Руководство по языку SQL СУБД Firebird 4.0

Дмитрий Филиппов, Александр Карпейкин, Алексей Ковязин, Дмитрий Кузьменко, Денис Симонов, Paul Vinkenoog, Дмитрий Еманов, Mark Rotteveel

Version 1.0.9, от 30.10.2023
Авторские права © 2017-2023 Firebird Project и всех участвующих авторов на Public Documentation License Version 1.0. Пожалуйста, обратитесь к License Notice in the Appendix
# Table of Contents

1. О руководстве по языку SQL Firebird 4.0 ................................................................. 27
   1.1. Что содержит данный документ ................................................................. 27
   1.2. Авторство ..................................................................................................... 27
       1.2.1. Авторы .................................................................................................... 27
   1.3. Благодарности ............................................................................................. 28
2. Структура языка SQL ......................................................................................... 29
   2.1. Общие сведения ........................................................................................... 29
       2.1.1. Подмножества SQL .............................................................................. 29
       2.1.2. Диалекты SQL ...................................................................................... 30
       2.1.3. Действия при ошибках ......................................................................... 31
   2.2. Основные сведения: операторы, предложения, ключевые слова .......... 31
   2.3. Идентификаторы .......................................................................................... 32
       2.3.1. Правила для обычных идентификаторов ............................................. 32
       2.3.2. Правила для идентификаторов с разделителями ............................... 33
   2.4. Литералы ...................................................................................................... 33
   2.5. Операторы и специальные символы ......................................................... 34
   2.6. Комментарии ............................................................................................... 34
3. Типы данных ......................................................................................................... 36
   3.1. Целочисленные типы данных ..................................................................... 39
       3.1.1. SMALLINT ............................................................................................ 39
       3.1.2. INTEGER ............................................................................................... 39
       3.1.3. BIGINT ................................................................................................ 39
       3.1.4. INT128 .................................................................................................. 40
       3.1.5. Шестнадцатеричный формат для целых чисел .................................. 40
   3.2. Типы данных с плавающей точкой ........................................................... 41
       3.2.1. Приблизительные числовые типы ....................................................... 41
            FLOAT ....................................................................................................... 42
            REAL ....................................................................................................... 43
            DOUBLE PRECISION ............................................................................. 43
            LONG FLOAT ......................................................................................... 43
       3.2.2. Десятичные типы с плавающей точкой .............................................. 43
            DECIMAL ................................................................................................. 44
            DECIMAL ................................................................................................. 44
   3.3. Типы данных с фиксированной точкой ...................................................... 49
       3.3.1. NUMERIC ............................................................................................. 49
       3.3.2. DECIMAL ............................................................................................. 50
       3.3.3. Точность арифметических операций .................................................... 51
   3.4. Типы данных для работы с датой и временем ......................................... 52
       3.4.1. DATE .................................................................................................... 53
3.4.2. TIME ................................................................. 54
3.4.3. TIMESTAMP ...................................................... 54
3.4.4. Часовой пояс сеанса ........................................... 55
  Получение часового пояса сеанса ................................ 56
3.4.5. Формат часового пояса ....................................... 56
  Региональная семантика TIME WITH TIME ZONE ............... 56
3.4.6. Литералы даты и времени .................................... 57
3.4.7. Операции, использующие значения даты и времени ....... 60
3.4.8. Дополнительные функции для поддержки часовочных поясов 63
  Виртуальная таблица RDB$TIME_ZONES ...................... 63
  Пакет RDB$TIME_ZONE_UTIL ..................................... 63
3.4.9. Обновление базы данных часовочных поясов ............... 63
3.5. Символьные типы данных ........................................ 63
  3.5.1. Unicode ..................................................... 64
  3.5.2. Набор символов клиента ................................... 64
  3.5.3. Специальные наборы символов ............................. 64
  3.5.4. Последовательность сортировки .......................... 65
    Независимый от регистра поиск ................................ 65
    Последовательности сортировки для UTF-8 .................... 66
  3.5.5. Индексирование символьных типов ....................... 66
  3.5.6. BINARY ................................................... 67
  3.5.7. CHAR ........................................................ 67
  3.5.8. VARBINARY ................................................ 68
  3.5.9. VARCHAR ................................................... 69
  3.5.10. NCHAR .................................................... 69
3.6. Логический тип данных ........................................ 69
  3.6.1. BOOLEAN .................................................. 69
    Оператор IS ..................................................... 70
    Примеры BOOLEAN ............................................... 70
    Использование Boolean с другими типами данных .......... 71
3.7. Бинарные типы данных .......................................... 72
  3.7.1. BLOB ...................................................... 72
    Подтипы BLOB .................................................. 72
    Особенности BLOB ............................................. 73
  3.7.2. Массивы ................................................... 74
    Указание явных границ для измерений ........................ 74
    Добавление дополнительных измерений ....................... 74
    Использование массивов ....................................... 75
3.8. Специальные типы данных ...................................... 75
  3.8.1. Тип данных SQL_NULL ..................................... 76
3.9. Преобразование типов данных .................................. 77
4. Общие элементы языка ........................................ 90
   4.1. Выражения ........................................ 90
      4.1.1. Литералы (константы) ....................... 92
         Строковые литералы (константы) ............. 92
         Числовые константы ............................ 95
         Логические литералы ............................ 97
      4.1.2. Операторы SQL ................................. 97
         Приоритет операторов ........................... 97
         Оператор конкатенации ......................... 98
         Арифметические операторы ..................... 98
         Операторы сравнения ............................ 98
         Логические операторы .......................... 99
      4.1.3. AT ........................................... 100
      4.1.4. NEXT VALUE FOR ............................. 100
      4.1.5. Условные выражения ....................... 101
         CASE ........................................ 101
      4.1.6. NULL в выражениях ......................... 103
         Выражения возвращающие NULL ................ 103
         NULL в логических выражениях ................. 103
   4.2. Подзапросы ........................................ 104
      4.2.1. Коррелированные подзапросы ............. 104
      4.2.2. Подзапросы возвращающие скалярный результат ........ 104
4.3. Предикаты ................................................................. 106
4.3.1. Утверждения .................................................... 106
4.3.2. Предикаты сравнения .......................................... 106
4.3.3. Другие предикаты сравнения ................................. 107
    BETWEEN ............................................................ 107
    LIKE ................................................................. 108
    STARTING WITH ................................................ 110
    CONTAINING ..................................................... 111
    SIMILAR TO ........................................................ 112
    IS DISTINCT FROM ............................................. 117
    Логический IS [NOT] ............................................. 118
    IS [NOT] NULL .................................................... 119
4.3.4. Предикаты существования .................................... 119
    EXISTS ............................................................. 120
    IN ................................................................. 121
    SINGULAR ........................................................ 123
4.3.5. Количественные предикаты подзапросов .................... 124
    ALL ................................................................. 124
    ANY и SOME ..................................................... 125
5. Операторы определения данных (DDL) ............................ 126
5.1. DATABASE .......................................................... 126
5.1.1. CREATE DATABASE ........................................... 126
    Использование псевдонимов БД ................................ 128
    Создание БД на удалённом сервере ............................ 128
    Необязательные параметры CREATE DATABASE .............. 129
    Диалект базы данных .......................................... 131
    Кто может создать базу данных? .............................. 131
    Примеры .......................................................... 131
5.1.2. ALTER DATABASE .............................................. 134
    Добавление вторичного файла .................................. 135
    Изменение пути и имени дельта файла ....................... 135
    Перевод базы данных в режим “безопасного копирования” 136
    Изменение набора символов по умолчанию .................. 137
    Изменение привилегий выполнения по умолчанию .......... 137
    LINGER ............................................................ 137
    Шифрование базы данных ...................................... 138
    Управление репликацией ....................................... 139
    Кто может выполнить ALTER DATABASE? ...................... 141
5.1.3. DROP DATABASE ................................................ 141
    Кто может удалить базу данных? ............................. 141
    Примеры .......................................................... 141
| 5.2. SHADOW | .......................................................... | 142 |
| 5.2.1. CREATE SHADOW | .................................................. | 142 |
| Режимы AUTO и MANUAL | .................................................. | 143 |
| Необязательные параметры CREATE SHADOW | .................. | 143 |
| Кто может создать теневую копию? | .......................... | 143 |
| Примеры | .................................................. | 144 |
| 5.2.2. DROP SHADOW | .................................................. | 144 |
| Кто может удалить теневую копию? | .......................... | 144 |
| Примеры | .................................................. | 145 |
| 5.3. DOMAIN | .......................................................... | 145 |
| 5.3.1. CREATE DOMAIN | .................................................. | 145 |
| Детали для конкретного типа | .......................... | 147 |
| Кто может создать домен? | .......................... | 149 |
| Примеры | .................................................. | 149 |
| 5.3.2. ALTER DOMAIN | .................................................. | 150 |
| Что не может изменить ALTER DOMAIN | .................. | 153 |
| Кто может изменить домен? | .......................... | 153 |
| Примеры | .................................................. | 153 |
| 5.3.3. DROP DOMAIN | .................................................. | 154 |
| Кто может удалить домен? | .......................... | 155 |
| Примеры | .................................................. | 155 |
| 5.4. TABLE | .......................................................... | 155 |
| 5.4.1. CREATE TABLE | .................................................. | 155 |
| Символьные столбцы | .......................... | 159 |
| Ограничение NOT NULL | .......................... | 159 |
| Значение по умолчанию | .......................... | 159 |
| Столбцы основанные на домене | .......................... | 159 |
| Столбцы идентификации (автоинкремент) | .......................... | 160 |
| Вычисляемые поля | .......................... | 161 |
| Столбцы типа массив | .......................... | 162 |
| Ограничения | .......................... | 162 |
| Привилегии выполнения | .......................... | 166 |
| Управление репликацией | .......................... | 167 |
| Кто может создать таблицу? | .......................... | 167 |
| Примеры | .................................................. | 167 |
| Глобальные временные таблицы (GTT) | .......................... | 171 |
| Внешние таблицы | .......................... | 172 |
| 5.4.2. ALTER TABLE | .................................................. | 175 |
| Счётчик форматов | .......................... | 179 |
| Предложение ADD | .......................... | 180 |
| Предложение DROP | .......................... | 181 |
Примеры
Кто может изменить представление?

Кто может создать представление с привилегиями выполнения WITH CHECK OPTIONS?

Обновляемые представления

Кто может обновить статистику?

Селективность индекса

Кто может удалить индекс?

Максимальное количество индексов на таблицу

Кто может создать индекс?

Управление репликацией

Предложение
Предложение
Предложение
Предложение

CREATE OR ALTER VIEW

ALTER VIEW

SET STATISTICS

DROP INDEX

ALTER INDEX

CREATE INDEX

DROP TABLE

ALTER SQL SECURITY

DROP SQL SECURITY

DROP CONSTRAINT

для индексов ограничений

Максимальное количество индексов на таблицу

Кто может создать таблицу?

Кто может изменить таблицу?

Управление репликацией

Предложение
Примеры

5.6.4. DROP VIEW

Кто может удалить представление?
Примеры

5.6.5. RECREATE VIEW

Примеры

5.7. TRIGGER

5.7.1. CREATE TRIGGER

Привилегии выполнения
Тело триггера
Терминатор оператора
DML триггеры (на таблицу или представление)
Триггеры на событие базы данных
Триггеры на события изменения метаданных

5.7.2. ALTER TRIGGER

Допустимые изменения
Кто может изменить триггеры?
Примеры

5.7.3. CREATE OR ALTER TRIGGER

Примеры

5.7.4. DROP TRIGGER

Кто может удалить триггеры?
Примеры

5.7.5. RECREATE TRIGGER

Примеры

5.8. PROCEDURE

5.8.1. CREATE PROCEDURE

Терминатор оператора
Параметры
Привилегии выполнения
Тело хранимой процедуры
Внешние хранимые процедуры
Кто может создать хранимую процедуру?
Примеры

5.8.2. ALTER PROCEDURE

Кто может изменить хранимую процедуру?
Примеры

5.8.3. CREATE OR ALTER PROCEDURE

Примеры

5.8.4. DROP PROCEDURE

Кто может удалить хранимую процедуру?
| Примеры | 235 |
| 5.8.5. RECREATE Procedure | 236 |
| Примеры | 236 |
| 5.9. FUNCTION | 237 |
| 5.9.1. CREATE FUNCTION | 237 |
| Терминатор оператора | 239 |
| Входные параметры | 240 |
| Использование доменов при объявлении параметров | 240 |
| Использование типа столбца при объявлении параметров | 240 |
| Возвращаемое значение | 240 |
| Детерминированные функции | 240 |
| Привилегии выполнения | 241 |
| Тело хранимой функции | 242 |
| Внешние функции | 242 |
| Кто может создать функцию? | 242 |
| Примеры | 243 |
| 5.9.2. ALTER FUNCTION | 245 |
| Кто может изменить функцию? | 246 |
| Примеры | 246 |
| 5.9.3. CREATE OR ALTER FUNCTION | 247 |
| Примеры | 247 |
| 5.9.4. DROP FUNCTION | 247 |
| Кто может удалить функцию? | 248 |
| Примеры | 248 |
| 5.9.5. RECREATE FUNCTION | 248 |
| Примеры | 249 |
| 5.10. PACKAGE | 249 |
| 5.10.1. CREATE PACKAGE | 249 |
| Привилегии выполнения | 252 |
| Терминатор оператора | 252 |
| Параметры процедур и функций | 252 |
| Детерминированные функции | 253 |
| Кто может создать пакет? | 253 |
| Примеры | 254 |
| 5.10.2. ALTER PACKAGE | 254 |
| Кто может изменить заголовок пакета? | 255 |
| 5.10.3. CREATE OR ALTER PACKAGE | 255 |
| Примеры | 256 |
| 5.10.4. DROP PACKAGE | 257 |
| Кто может удалить заголовок пакета? | 257 |
| Примеры | 257 |
5.10.5. RECREATE PACKAGE  ................................................................................. 257
   Примеры ............................................................................................................ 258
5.11. PACKAGE BODY ......................................................................................... 259
   5.11.1. CREATE PACKAGE BODY  ....................................................................... 259
      Кто может создать тело пакета? ............................................................... 262
      Примеры ............................................................................................................ 262
   5.11.2. DROP PACKAGE BODY ......................................................................... 263
      Кто может удалить тело пакета? ............................................................... 263
      Примеры ............................................................................................................ 263
   5.11.3. RECREATE PACKAGE BODY ................................................................. 263
      Примеры ............................................................................................................ 265
5.12. EXTERNAL FUNCTION ................................................................................ 265
   5.12.1. DECLARE EXTERNAL FUNCTION .......................................................... 266
      Кто может объявить внешнюю функцию? ........................................... 268
      Примеры ............................................................................................................ 269
   5.12.2. ALTER EXTERNAL FUNCTION ............................................................... 270
      Кто может изменить внешнюю функцию? ........................................... 270
      Примеры ............................................................................................................ 270
   5.12.3. DROP EXTERNAL FUNCTION ............................................................... 271
      Кто может удалить внешнюю функцию? ............................................ 271
      Примеры ............................................................................................................ 271
5.13. FILTER ......................................................................................................... 272
   5.13.1. DECLARE FILTER ................................................................................ 272
      Задание подтипов ....................................................................................... 273
      Параметры DECLARE FILTER ................................................................... 273
      Кто может создать BLOB фильтр? .......................................................... 273
      Примеры ............................................................................................................ 273
   5.13.2. DROP FILTER ....................................................................................... 274
      Кто может удалить BLOB фильтр? .......................................................... 274
      Примеры ............................................................................................................ 275
5.14. SEQUENCE (GENERATOR) ....................................................................... 275
   5.14.1. CREATE SEQUENCE ............................................................................. 275
      Кто может создать последовательность? ............................................ 276
      Примеры ............................................................................................................ 276
   5.14.2. ALTER SEQUENCE .............................................................................. 277
      Кто может изменить последовательность? ........................................ 278
      Примеры ............................................................................................................ 278
   5.14.3. CREATE OR ALTER SEQUENCE .......................................................... 279
      Примеры ............................................................................................................ 280
   5.14.4. DROP SEQUENCE ............................................................................... 280
      Кто может удалить генератор? ................................................................. 280
### Операторы обработки данных (DML)

<table>
<thead>
<tr>
<th>Статья</th>
<th>Страница</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1. SELECT</td>
<td>297</td>
</tr>
<tr>
<td>6.1.1. FIRST, SKIP</td>
<td>298</td>
</tr>
<tr>
<td>6.1.2. Список полей SELECT</td>
<td>300</td>
</tr>
<tr>
<td>5.14.5. RECREATE SEQUENCE</td>
<td>281</td>
</tr>
<tr>
<td>5.14.6. SET GENERATOR</td>
<td>282</td>
</tr>
<tr>
<td>5.15. EXCEPTION</td>
<td>283</td>
</tr>
<tr>
<td>5.16. COLLATION</td>
<td>287</td>
</tr>
<tr>
<td>5.17. CHARACTER SET</td>
<td>293</td>
</tr>
<tr>
<td>5.18. COMMENTS</td>
<td>294</td>
</tr>
</tbody>
</table>

---

Примеры

5.14.5. RECREATE SEQUENCE

Кто может изменить значение генератора?

Примеры

5.14.6. SET GENERATOR

Кто может создать сортировку?

5.15. EXCEPTION

Кто может добавить комментарий?

Примеры

5.16. COLLATION

Специфические атрибуты

Кто может создать сортировку?

Примеры

5.17. CHARACTER SET

Примеры

5.18. COMMENTS

Кто может добавить комментарий?

Примеры

6. Операторы обработки данных (DML)

6.1. SELECT

6.1.1. FIRST, SKIP

Особенности использования

Примеры

6.1.2. Список полей SELECT
Соединения

6.1.4. Соединения JOIN ................................................. 314
Внутренние (INNER) и внешние (OUTER) соединения . 316
Точные соединения ............................................. 318
Естественные соединения (NATURAL JOIN) .............. 322
Неявные соединения ............................................. 323
Смешивание явного и неявного соединения .............. 323
Перекрёстное соединение (CROSS JOIN) ................. 324
Неоднозначные имена полей в соединениях .............. 325
Соединения с хранимыми процедурами ............... 325
Соединения с LATERAL производными таблицами ... 325

6.1.5. WHERE ............................................................... 327

6.1.6. GROUP BY .......................................................... 330
HAVING ................................................................. 334

6.1.7. WINDOW ............................................................. 336

6.1.8. PLAN ................................................................. 337
Простые планы ....................................................... 339
Составные планы ..................................................... 342

6.1.9. UNION ............................................................... 346

6.1.10. ORDER BY ........................................................ 348
Направление сортировки ....................................... 349
Порядок сравнения ............................................ 350
Расположение NULL ............................................. 350
Сортировка частей UNION .................................... 350
Примеры ............................................................... 350

6.1.11. ROWS ............................................................... 352
Особенности при использовании ROWS с одним аргументом ............................................. 353
Особенности при использовании ROWS с двумя аргументами ............................................. 353
Замена FIRST ... SKIP ............................................ 353
Совместное использование FIRST ... SKIP и ROWS ..................................................... 353
Использование ROWS в UNION ................................ 353
Примеры ............................................................... 354

6.1.12. FETCH, OFFSET .................................................. 355
Примеры использования OFFSET и FETCH ............... 355

6.1.13. FOR UPDATE [OF] .................................................. 356

6.1.14. WITH LOCK ........................................................ 357
Использование предложения FOR UPDATE .............................................. 358
Как сервер работает с WITH LOCK ............................................... 359
Предостережения при использовании WITH LOCK .............................. 359
Примеры использования явной блокировки ...................................... 360
6.1.16. INTO ................................................................. 360
6.1.16. Общие табличные выражения CTE ("WITH ... AS ... SELECT") 362
6.1.16. Рекурсивные CTE ...................................................... 364
6.2. INSERT ................................................................. 366
6.2.1. INSERT ... VALUES ......................................................... 368
6.2.1. Ключевое слово DEFAULT .............................................. 368
6.2.2. INSERT ... SELECT ......................................................... 369
6.2.3. INSERT ... DEFAULT VALUES ......................................... 370
6.2.4. Директива OVERRIDING .............................................. 370
6.2.5. RETURNING ........................................................... 371
6.2.6. Вставка столбцов BLOB .................................................. 372
6.3. UPDATE ................................................................. 372
6.3.1. Использование псевдонима ................................................ 374
6.3.2. SET ........................................................................... 375
6.3.3. WHERE ....................................................................... 376
6.3.4. PLAN ........................................................................ 376
6.3.5. ORDER BY и ROWS ....................................................... 377
6.3.6. RETURNING ............................................................. 378
6.3.6. INTO ........................................................................ 378
6.3.7. Обновление столбцов BLOB ................................................ 379
6.4. UPDATE OR INSERT ......................................................... 379
6.4.1. Ключевое слово DEFAULT .............................................. 381
6.4.2. RETURNING ............................................................. 381
6.5. DELETE ................................................................. 382
6.5.1. WHERE ..................................................................... 383
6.5.2. PLAN ........................................................................ 384
6.5.3. ORDER BY и ROWS ....................................................... 384
6.5.4. RETURNING ............................................................. 385
6.6. MERGE ................................................................. 386
6.6.1. RETURNING ............................................................. 390
6.7. EXECUTE PROCEDURE ....................................................... 391
6.7.1. "Выполняемые" хранимые процедуры .................................... 392
6.8. EXECUTE BLOCK .............................................................. 393
6.8.1. Входные и выходные параметры ......................................... 395
6.8.2. Терминатор оператора ........................................................ 396
7. Операторы процедурного SQL (PSQL) ............................................. 397
7.1. Элементы PSQL .................................................. 397
  7.1.1. DML операторы с параметрами ............................. 397
  7.1.2. Транзакции ............................................... 397
  7.1.3. Структура модуля ......................................... 398
    Заголовок модуля ............................................. 398
    Привилегии выполнения PSQL кода ............................ 398
    Тело модуля .................................................. 398
7.2. Хранимые процедуры ............................................ 401
  7.2.1. Преимущества хранимых процедур ......................... 402
  7.2.2. Типы хранимых процедур .................................. 402
    Выполняемые хранимые процедуры ............................ 402
    Селективные хранимые процедуры ............................. 402
  7.2.3. Создание хранимой процедуры ............................. 403
  7.2.4. Изменение хранимой процедуры ......................... 403
  7.2.5. Удаление хранимой процедуры ............................ 403
7.3. Хранимые функции .............................................. 403
  7.3.1. Создание хранимой функции .............................. 403
  7.3.2. Изменение хранимой функции ............................. 403
  7.3.3. Удаление хранимой функции ............................. 404
7.4. PSQL блоки .................................................... 404
7.5. Пакеты ........................................................ 404
  7.5.1. Преимущества пакетов ................................... 405
  7.5.2. Создание пакета ........................................ 406
  7.5.3. Модификация пакета .................................... 406
  7.5.4. Удаление пакета ........................................ 406
7.6. Триггеры ........................................................ 406
  7.6.1. Порядок срабатывания .................................... 406
  7.6.2. DML триггеры ........................................... 406
    Варианты триггеров .......................................... 407
    Контекстные переменные NEW и OLD .......................... 407
  7.6.3. Триггеры на события базы данных ...................... 407
  7.6.4. DDL триггеры ........................................... 408
    Переменные доступные в пространстве имён DDL_TRIGGER .......................... 408
  7.6.5. Создание триггера ....................................... 409
  7.6.6. Изменение триггера ..................................... 409
  7.6.7. Удаление триггера ..................................... 409
7.7. Написание кода тела модуля ................................ 409
  7.7.1. Оператор присваивания ................................... 409
  7.7.2. DECLARE VARIABLE ...................................... 410
    Типы данных для переменных .................................. 412
    Ограничение NOT NULL ........................................ 412
<table>
<thead>
<tr>
<th>7.7.14. EXECUTE STATEMENT</th>
<th>432</th>
</tr>
</thead>
<tbody>
<tr>
<td>Параметризованные операторы</td>
<td>433</td>
</tr>
<tr>
<td>WITH {AUTONOMOUS</td>
<td>COMMON} TRANSACTION</td>
</tr>
<tr>
<td>WITH CALLER PRIVILEGES</td>
<td>436</td>
</tr>
<tr>
<td>ON EXTERNAL [DATA SOURCE]</td>
<td>436</td>
</tr>
<tr>
<td>AS USER, PASSWORD и ROLE</td>
<td>439</td>
</tr>
<tr>
<td>Предостережения</td>
<td>440</td>
</tr>
<tr>
<td>7.7.15. FOR SELECT</td>
<td>440</td>
</tr>
<tr>
<td>Необъявленный курсор</td>
<td>441</td>
</tr>
<tr>
<td>Примеры использования FOR SELECT</td>
<td>442</td>
</tr>
<tr>
<td>7.7.16. FOR EXECUTE STATEMENT</td>
<td>445</td>
</tr>
<tr>
<td>Примеры `FOR EXECUTE STATEMENT</td>
<td>446</td>
</tr>
<tr>
<td>7.7.17. OPEN</td>
<td>446</td>
</tr>
<tr>
<td>Примеры OPEN</td>
<td>447</td>
</tr>
<tr>
<td>7.7.18. FETCH</td>
<td>449</td>
</tr>
<tr>
<td>Примеры FETCH</td>
<td>451</td>
</tr>
<tr>
<td>7.7.19. CLOSE</td>
<td>454</td>
</tr>
</tbody>
</table>
### 8. Встроенные скалярные функции

#### 8.1. Функции для работы с контекстными переменными

<table>
<thead>
<tr>
<th>Функция</th>
<th>Описание</th>
<th>Страница</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$GET_CONTEXT()</td>
<td>Пространство имен SYSTEM</td>
<td>466</td>
</tr>
<tr>
<td>RDB$GET_CONTEXT()</td>
<td>Пространство имен DDL_TRIGGER</td>
<td>469</td>
</tr>
<tr>
<td>RDB$SET_CONTEXT()</td>
<td>Примеры</td>
<td>470</td>
</tr>
</tbody>
</table>

#### 8.2. Математические функции

<table>
<thead>
<tr>
<th>Функция</th>
<th>Описание</th>
<th>Страница</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS()</td>
<td></td>
<td>472</td>
</tr>
<tr>
<td>ACOS()</td>
<td></td>
<td>473</td>
</tr>
<tr>
<td>ACOSH()</td>
<td></td>
<td>473</td>
</tr>
<tr>
<td>ASIN()</td>
<td></td>
<td>474</td>
</tr>
<tr>
<td>ASINH()</td>
<td></td>
<td>474</td>
</tr>
<tr>
<td>ATAN()</td>
<td></td>
<td>475</td>
</tr>
<tr>
<td>ATAN2()</td>
<td></td>
<td>475</td>
</tr>
<tr>
<td>ATANH()</td>
<td></td>
<td>476</td>
</tr>
<tr>
<td>CEIL(), CEILING()</td>
<td></td>
<td>477</td>
</tr>
<tr>
<td>COS()</td>
<td></td>
<td>477</td>
</tr>
<tr>
<td>COSH()</td>
<td></td>
<td>478</td>
</tr>
<tr>
<td>COT()</td>
<td></td>
<td>478</td>
</tr>
<tr>
<td>EXP()</td>
<td></td>
<td>479</td>
</tr>
<tr>
<td>FLOOR()</td>
<td></td>
<td>479</td>
</tr>
<tr>
<td>LN()</td>
<td></td>
<td>479</td>
</tr>
<tr>
<td>LOG()</td>
<td></td>
<td>480</td>
</tr>
<tr>
<td>LOG10()</td>
<td></td>
<td>481</td>
</tr>
<tr>
<td>MOD()</td>
<td></td>
<td>481</td>
</tr>
<tr>
<td>PI()</td>
<td></td>
<td>481</td>
</tr>
</tbody>
</table>
8.3. Функции для работы со строками

8.3.1. ASCII_CHAR() ................................................. 488
8.3.2. ASCII_VAL() ................................................. 488
8.3.3. BASE64_DECODE() ........................................ 489
   Примеры BASE64_DECODE .................................. 489
8.3.4. BASE64_ENCODE() ........................................ 490
   Примеры BASE64_ENCODE .................................. 490
8.3.5. BIT_LENGTH() ............................................... 491
   Примеры BIT_LENGTH ........................................ 491
8.3.6. CHAR_LENGTH(), CHARACTER_LENGTH() ............. 492
8.3.7. HASH() ..................................................... 492
   Примеры HASH ............................................... 493
8.3.8. HEX DECODE() ............................................ 493
   Примеры HEX DECODE ...................................... 494
8.3.9. HEX_ENCODE() ............................................ 494
   Примеры HEX_ENCODE ...................................... 495
8.3.10. LEFT() ................................................... 495
8.3.11. LOWER() .................................................. 496
   Примеры LOWER .............................................. 497
8.3.12. LPAD() .................................................... 497
   Примеры LPAD ............................................... 498
8.3.13. OCTET_LENGTH() ........................................ 498
   Примеры OCTET_LENGTH .................................... 499
8.3.14. OVERLAY() ................................................ 499
   Примеры OVERLAY ........................................... 500
8.3.15. POSITION() ............................................... 501
   Примеры POSITION .......................................... 501
8.3.16. REPLACE() ............................................... 502
   Примеры REPLACE ............................................ 503
8.3.17. REVERSE() ............................................... 503
8.3.18. RIGHT() .......................................................... 504
8.3.19. RPAD() ......................................................... 505
Примеры RPAD .................................................. 506
8.3.20. SUBSTRING() .................................................. 506
Позиционный SUBSTRING ........................................ 507
SUBSTRING по регулярному выражению ..................... 508
8.3.21. TRIM() .......................................................... 509
Примеры TRIM .................................................... 509
8.3.22. UPPER() ......................................................... 510
Примеры UPPER .................................................. 510

8.4. Функции для работы с датой и временем ..................... 511
8.4.1. DATEADD() .................................................... 511
Примеры DATEADD .............................................. 512
8.4.2. DATEDIFF() .................................................... 512
Примеры DATEDIFF ............................................. 513
8.4.3. EXTRACT() .................................................... 514
8.4.4. FIRST_DAY() .................................................. 515
Примеры FIRST_DAY ........................................... 516
8.4.5. LAST_DAY() ................................................... 516
Примеры LAST_DAY ............................................ 516

8.5. Функции для работы с типом BLOB ......................... 517
8.5.1. BLOB_APPEND() ............................................. 517

8.6. Функции для работы с типом DECFLOAT .................. 520
8.6.1. COMPARE_DecFLOAT() .................................. 520
8.6.2. NORMALIZE_DecFLOAT() .................................. 521
Примеры NORMALIZE_DecFLOAT .............................. 521
8.6.3. QUANTIZE() .................................................. 521
8.6.4. TOTALORDER() ............................................ 523

8.7. Криптографические функции ...................................... 524
8.7.1. CRYPT_HASH() ............................................ 524
Примеры CRYPT_HASH ........................................... 524
8.7.2. DECRYPT() .................................................. 524
8.7.3. ENCRYPT() .................................................. 524
8.7.4. RSA_PRIVATE() ............................................ 526
8.7.5. RSA_PUBLIC() ............................................ 528
8.7.6. RSA_ENCRYPT() ............................................ 529
8.7.7. RSA_DECRYPT() ............................................ 529
8.7.8. RSA_SIGN_HASH() ............................................ 530
8.7.9. RSA_VERIFY_HASH() ............................................ 530

8.8. Функции преобразования типов ................................. 533
8.8.1. CAST() ................................................................. 533
    Преобразование к домену или к его базовому типу ........................................ 534
    Преобразование к типу столбца ....................................................................... 535
    Примеры приведения типов .............................................................................. 536
8.9. Функции побитовых операций ...................................................................... 536
  8.9.1. BIN_AND() .......................................................... 536
  8.9.2. BIN_NOT() ........................................................................................................ 537
  8.9.3. BIN_OR() ......................................................................................................... 537
  8.9.4. BIN_SHL() .............................................................. 538
  8.9.5. BIN_SHR() ..................................................................................................... 538
  8.9.6. BIN_XOR() .................................................................................................... 539
8.10. Функции для работы с UUID ..................................................................... 539
   8.10.1. CHAR_TO_UUID() .................................................. 539
           Примеры CHAR_TO_UUID ......................................................... 540
   8.10.2. GEN_UUID() ........................................................... 540
           Примеры GEN_UUID ................................................................. 540
   8.10.3. UUID_TO_CHAR() ...................................................... 541
           Примеры UUID_TO_CHAR .......................................................... 541
8.11. Функции для работы с генераторами (последовательностями) ............... 542
   8.11.1. GEN_ID() ................................................................. 542
           Примеры GEN_ID ............................................................... 542
8.12. Условные функции ................................................................................. 543
   8.12.1. COALESCE() ............................................................ 543
           Примеры COALESCE ............................................................. 543
   8.12.2. DECODE() .................................................................................................. 544
           Примеры DECODE ................................................................. 545
   8.12.3. IIF() .......................................................................................................... 545
           Примеры IIF ................................................................. 546
   8.12.4. MAXVALUE() .......................................................... 546
           Примеры MAXVALUE ............................................................ 546
   8.12.5. MINVALUE() .......................................................... 547
           Примеры MINVALUE ............................................................. 547
   8.12.6. NULLIF() ................................................................................................... 547
           Примеры NULLIF ............................................................ 548
8.13. Другие функции ....................................................................................... 548
   8.13.1. MAKE_DBKEY() ..................................................... 548
   8.13.2. RDB$ERROR() ......................................................... 550
   8.13.3. RDB$GET_TRANSACTION_CN() ........................................... 552
   8.13.4. RDB$ROLE_IN_USE() .................................................. 553
   8.13.5. RDB$SYSTEM_PRIVILEGE() .............................................. 554
9. Агрегатные функции .................................................................................... 555
9.1. Предложение FILTER .................................................. 555
9.2. Основные агрегатные функции .................................. 556
  9.2.1. AVG() ................................................................. 556
     Примеры AVG .......................................................... 557
  9.2.2. COUNT() ............................................................ 557
     Примеры COUNT ........................................................ 558
  9.2.3. LIST() ................................................................. 558
     Примеры LIST .......................................................... 559
  9.2.4. MAX() ................................................................. 560
     Примеры MAX .......................................................... 560
  9.2.5. MIN() ................................................................. 560
     Примеры MIN .......................................................... 561
  9.2.6. SUM() ................................................................. 561
     Примеры SUM .......................................................... 562
9.3. Статистические функции ............................................. 562
  9.3.1. CORR() ............................................................... 563
     Примеры CORR ........................................................ 563
  9.3.2. COVAR_POP() ...................................................... 564
     Примеры COVAR_POP ................................................ 564
  9.3.3. COVAR_SAMP() ..................................................... 564
     Примеры COVAR_SAMP ............................................. 565
  9.3.4. STDDEV_POP() ..................................................... 565
     Примеры STDDEV_POP ............................................. 566
  9.3.5. STDDEV_SAMP() ................................................... 566
     Примеры STDDEV_SAMP ........................................... 567
  9.3.6. VAR_POP() .......................................................... 567
     Примеры VAR_POP .................................................... 568
  9.3.7. VAR_SAMP() ......................................................... 568
     Примеры VAR_SAMP ................................................ 569
9.4. Функции линейной регрессии .................................... 569
  9.4.1. REGR_AVGX() ...................................................... 570
  9.4.2. REGR_AVGY() ...................................................... 570
  9.4.3. REGR_COUNT() .................................................... 571
  9.4.4. REGR_INTERCEPT() ............................................. 572
     Примеры REGR_INTERCEPT ......................................... 573
  9.4.5. REGR_R2() .......................................................... 573
  9.4.6. REGR_SLOPE() ..................................................... 574
  9.4.7. REGR_SXX() ........................................................ 575
  9.4.8. REGR_SXY() ........................................................ 576
  9.4.9. REGR_SYY() ........................................................ 577
10. Оконные (аналитические) функции ................................. 578
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1. Агрегатные функции</td>
<td>580</td>
</tr>
<tr>
<td>10.2. Секционирование</td>
<td>581</td>
</tr>
<tr>
<td>10.3. Сортировка</td>
<td>582</td>
</tr>
<tr>
<td>10.4. Рамка окна</td>
<td>584</td>
</tr>
<tr>
<td>10.4.1. Окна диапазона</td>
<td>585</td>
</tr>
<tr>
<td>10.4.2. Окна строк</td>
<td>586</td>
</tr>
<tr>
<td>10.5. Именованные окна</td>
<td>587</td>
</tr>
<tr>
<td>10.6. Ранжирующие функции</td>
<td>588</td>
</tr>
<tr>
<td>10.6.1. DENSE_RANK()</td>
<td>588</td>
</tr>
<tr>
<td>10.6.2. RANK()</td>
<td>589</td>
</tr>
<tr>
<td>10.6.3. PERCENT_RANK()</td>
<td>590</td>
</tr>
<tr>
<td>10.6.4. CUME_DIST()</td>
<td>591</td>
</tr>
<tr>
<td>10.6.5. NTILE()</td>
<td>592</td>
</tr>
<tr>
<td>10.6.6. ROW_NUMBER()</td>
<td>593</td>
</tr>
<tr>
<td>10.7. Навигационные функции</td>
<td>594</td>
</tr>
<tr>
<td>10.7.1. FIRST_VALUE()</td>
<td>595</td>
</tr>
<tr>
<td>10.7.2. LAG()</td>
<td>596</td>
</tr>
<tr>
<td>10.7.3. LAST_VALUE()</td>
<td>597</td>
</tr>
<tr>
<td>10.7.4. LEAD()</td>
<td>598</td>
</tr>
<tr>
<td>10.7.5. NTH_VALUE()</td>
<td>599</td>
</tr>
<tr>
<td>10.8. Агрегатные функции внутри окончных</td>
<td>599</td>
</tr>
<tr>
<td>11. Системные пакеты</td>
<td>601</td>
</tr>
<tr>
<td>11.1. Пакет RDB$TIME_ZONE_UTIL</td>
<td>601</td>
</tr>
<tr>
<td>11.1.1. Функция RDB$TIME_ZONE_UTIL.DATABASE_VERSION()</td>
<td>601</td>
</tr>
<tr>
<td>11.1.2. Процедура RDB$TIME_ZONE_UTIL.TRANSITIONS()</td>
<td>601</td>
</tr>
<tr>
<td>12. Контекстные переменные</td>
<td>603</td>
</tr>
<tr>
<td>12.1. CURRENT_CONNECTION</td>
<td>603</td>
</tr>
<tr>
<td>12.2. CURRENT_DATE</td>
<td>603</td>
</tr>
<tr>
<td>12.3. CURRENT_ROLE</td>
<td>604</td>
</tr>
<tr>
<td>12.4. CURRENT_TIME</td>
<td>605</td>
</tr>
<tr>
<td>12.5. CURRENT_TIMESTAMP</td>
<td>605</td>
</tr>
<tr>
<td>12.6. CURRENT_TRANSACTION</td>
<td>606</td>
</tr>
<tr>
<td>12.7. CURRENT_USER</td>
<td>607</td>
</tr>
<tr>
<td>12.8. DELETING</td>
<td>607</td>
</tr>
<tr>
<td>12.9. GDSCODE</td>
<td>608</td>
</tr>
<tr>
<td>12.10. INSERTING</td>
<td>609</td>
</tr>
<tr>
<td>12.11. LOCALTIME</td>
<td>610</td>
</tr>
<tr>
<td>12.12. LOCALTIME_TIMESTAMP</td>
<td>610</td>
</tr>
<tr>
<td>12.13. NEW</td>
<td>611</td>
</tr>
<tr>
<td>12.14. OLD</td>
<td>612</td>
</tr>
<tr>
<td>12.15. RESETTING</td>
<td>613</td>
</tr>
</tbody>
</table>
12.16. ROW_COUNT
12.17. SQLCODE
12.18. SQLSTATE
12.19. UPDATING
12.20. USER
13. Управление транзакциями
13.1. SET TRANSACTION
13.1.1. Параметры транзакции
   Имя транзакции
   Режим доступа
   Режим разрешения блокировок
   ISOLATION LEVEL
   NO AUTO UNDO
   IGNORE LIMBO
   AUTO COMMIT
   RESERVING
13.2. COMMIT
13.3. ROLLBACK
13.3.1. ROLLBACK TO SAVEPOINT
13.4. SAVEPOINT
13.5. RELEASE SAVEPOINT
13.6. Внутренние точки сохранения
13.7. Точки сохранения и PSQL
14. Безопасность
14.1. Аутентификация пользователя
14.1.1. Специальные учётные записи
   SYSDBA
   Особенности POSIX
   Особенности Windows
14.1.2. Владелец базы данных
14.1.3. Роль RDB$ADMIN
   Предоставление роли RDB$ADMIN в обычной базе данных
   Использование роли RDB$ADMIN в обычной базе данных
   Предоставление роли RDB$ADMIN в базе данных пользователей
   Использование роли RDB$ADMIN в базе данных пользователей
   Использование роли RDB$ADMIN в gsec
   AUTO ADMIN MAPPING
14.1.4. Администраторы
14.2. Управление пользователями
14.2.1. CREATE USER
   Кто может создать пользователя
Операторы для отзыва привилегий и ролей

Операторы для предоставления привилегий и назначения ролей

Роли

14.4. SQL привилегии

14.4.3. Привилегии выполнения SQL кода

Примеры

14.4.4. DROP USER

14.4.5. RECREATE USER

Примеры

14.2.6. Получение списка пользователей

14.3. SQL привилегии

14.3.1. Владелец объекта базы данных

14.3.2. Привилегии выполнения SQL кода

Примеры

14.4. Роли

14.4.1. CREATE ROLE

Роли с системными привилегиями

Кто может создать роль

Примеры CREATE ROLE

14.4.2. ALTER ROLE

14.4.3. DROP ROLE

Кто может удалить роль

Примеры DROP ROLE

14.5. Операторы для предоставления привилегий и назначения ролей

14.5.1. GRANT

Предложение TO

Пользователь PUBLIC

Предложение WITH GRANT OPTION

Предложение GRANTED BY

Табличные привилегии

Привилегия EXECUTE

Привилегия USAGE

DDL привилегии

DDL привилегии для базы данных

Примеры предоставления DDL привилегий на базу данных

Предоставление прав системным привилегиям

Примеры предоставления прав системным привилегиям

Назначение ролей

14.6. Операторы для отзыва привилегий и ролей
14.6.1. REVOKE .......................... 677
Предложение FROM .......................... 678
Примеры отзыва привилегий .......................... 678
Предложение GRANT OPTION FOR .......................... 680
Отзыв привилегий с использованием GRANT OPTION FOR .......................... 680
Отмена назначенных ролей .......................... 680
Предложение GRANTED BY .......................... 681
REVOKE ALL ON ALL .......................... 682
14.7. Отображение объектов безопасности .......................... 682
14.7.1. CREATE MAPPING .......................... 683
Кто может создать отображение .......................... 685
Примеры CREATE MAPPING .......................... 685
14.7.2. ALTER MAPPING .......................... 687
Кто может изменить отображение .......................... 687
Примеры ALTER MAPPING .......................... 687
14.7.3. CREATE OR ALTER MAPPING .......................... 688
Примеры CREATE OR ALTER MAPPING .......................... 688
14.7.4. DROP MAPPING .......................... 688
Кто может удалить отображение .......................... 689
Примеры DROP MAPPING .......................... 689
14.8. Шифрование базы данных .......................... 689
15. Управляющие операторы .......................... 691
15.1. Поведение типов данных .......................... 691
15.1.1. SET BIND .......................... 691
15.1.2. SET DECFLOAT .......................... 694
SET DECFLOAT ROUND .......................... 694
SET DECFLOAT TRAPS .......................... 696
15.2. Тайм-ауты .......................... 696
15.2.1. Тайм-аут выполнения SQL оператора .......................... 697
SET STATEMENT TIMEOUT .......................... 698
15.2.2. Тайм-аут простого соединения .......................... 699
SET SESSION IDLE TIMEOUT .......................... 700
15.3. Пул внешних соединений .......................... 701
15.3.1. ALTER EXTERNAL CONNECTIONS POOL SET SIZE .......................... 701
15.3.2. ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME .......................... 701
15.3.3. ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL .......................... 702
15.3.4. ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST .......................... 702
15.4. Изменение текущей роли .......................... 702
15.4.1. SET ROLE .......................... 702
15.4.2. SET TRUSTED ROLE .......................... 703
15.5. Управление часовым поясом сеанса .......................... 704
<table>
<thead>
<tr>
<th>Appendix D: Системные таблицы</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$AUTH_MAPPING</td>
<td>784</td>
</tr>
<tr>
<td>RDB$BACKUP_HISTORY</td>
<td>786</td>
</tr>
<tr>
<td>RDB$CHARACTER_SETS</td>
<td>787</td>
</tr>
<tr>
<td>RDB$CHECK_CONSTRAINTS</td>
<td>787</td>
</tr>
<tr>
<td>RDB$COLLATIONS</td>
<td>788</td>
</tr>
<tr>
<td>RDB$CONFIG</td>
<td>789</td>
</tr>
<tr>
<td>RDB$DATABASE</td>
<td>790</td>
</tr>
<tr>
<td>RDB$DBCreators</td>
<td>791</td>
</tr>
<tr>
<td>RDB$DEPENDENCIES</td>
<td>792</td>
</tr>
<tr>
<td>RDB$EXCEPTIONS</td>
<td>793</td>
</tr>
<tr>
<td>RDB$FIELD_DIMENSIONS</td>
<td>794</td>
</tr>
<tr>
<td>RDB$FIELDS</td>
<td>795</td>
</tr>
<tr>
<td>RDB$FILES</td>
<td>795</td>
</tr>
<tr>
<td>RDB$FILTERS</td>
<td>800</td>
</tr>
<tr>
<td>RDB$FORMATS</td>
<td>801</td>
</tr>
<tr>
<td>RDB$FUNCTION_ARGUMENTS</td>
<td>802</td>
</tr>
<tr>
<td>RDB$FUNCTIONS</td>
<td>803</td>
</tr>
<tr>
<td>RDB$GENERATORS</td>
<td>805</td>
</tr>
<tr>
<td>RDB$INDEX_SEGMENTS</td>
<td>807</td>
</tr>
<tr>
<td>RDB$INDICES</td>
<td>808</td>
</tr>
<tr>
<td>RDB$LOG_FILES</td>
<td>808</td>
</tr>
<tr>
<td>RDB$PACKAGES</td>
<td>810</td>
</tr>
<tr>
<td>RDB$PAGES</td>
<td>810</td>
</tr>
<tr>
<td>RDB$PROCEDURE_PARAMETERS</td>
<td>811</td>
</tr>
<tr>
<td>RDB$PROCEDURES</td>
<td>813</td>
</tr>
<tr>
<td>RDB$PUBLICATION_TABLES</td>
<td>814</td>
</tr>
<tr>
<td>RDB$PUBLICATIONS</td>
<td>815</td>
</tr>
</tbody>
</table>
### Appendix E: Таблицы мониторинга

- MON$ATTACHMENTS .................................................. 832
  - Использование MON$ATTACHMENTS для закрытия подключений ... 835
- MON$CALL_STACK .................................................. 836
- MON$CONTEXT_VARIABLES ....................................... 837
- MON$DATABASE .................................................... 838
- MON$IO_STATS ..................................................... 840
- MON$MEMORY_USAGE .............................................. 841
- MON$RECORD_STATS ............................................... 842
- MON$STATEMENTS .................................................. 844
  - Использование MON$STATEMENTS для отмены запросов ....... 845
- MON$TABLE_STATS ................................................. 846
- MON$TRANSACTIONS ............................................... 847

### Appendix F: Таблицы безопасности

- SEC$GLOBAL_AUTH_MAPPING ..................................... 850
- SEC$USERS .......................................................... 850
- SEC$USER_ATTRIBUTES ........................................... 851

### Appendix G: Таблицы плагинов

- Плагин управления пользователями Srp .......................... 853
  - Таблица PLG$SRP ............................................... 853
  - Представление PLG$SRP_VIEW ................................ 853
- Плагин управления пользователями Legacy_UserManager .... 854
  - Таблица PLG$USERS ............................................ 854
  - Представление PLG$VIEW_USERS ............................. 855

### Appendix H: Наборы символов и порядки сортировки

- Алфавитный указатель .......................................... 864

<table>
<thead>
<tr>
<th>Table of Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$REF_CONSTRAINTS</td>
<td>815</td>
</tr>
<tr>
<td>RDB$RELATION_CONSTRAINTS</td>
<td>816</td>
</tr>
<tr>
<td>RDB$RELATION_FIELDS</td>
<td>817</td>
</tr>
<tr>
<td>RDB$RELATIONS</td>
<td>818</td>
</tr>
<tr>
<td>RDB$ROLES</td>
<td>820</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASSES</td>
<td>821</td>
</tr>
<tr>
<td>RDB$TIME_ZONES</td>
<td>822</td>
</tr>
<tr>
<td>RDB$TRANSACTIONS</td>
<td>822</td>
</tr>
<tr>
<td>RDB$TRIGGER_MESSAGES</td>
<td>822</td>
</tr>
<tr>
<td>RDB$TRIGGERS</td>
<td>823</td>
</tr>
<tr>
<td>RDB$TYPES</td>
<td>827</td>
</tr>
<tr>
<td>RDB$USER_PRIVILEGES</td>
<td>827</td>
</tr>
<tr>
<td>RDB$VIEW_RELATIONS</td>
<td>830</td>
</tr>
<tr>
<td>MON$ATTACHMENTS</td>
<td>832</td>
</tr>
<tr>
<td>Использование MON$ATTACHMENTS для закрытия подключений</td>
<td>835</td>
</tr>
<tr>
<td>MON$CALL_STACK</td>
<td>836</td>
</tr>
<tr>
<td>MON$CONTEXT_VARIABLES</td>
<td>837</td>
</tr>
<tr>
<td>MON$DATABASE</td>
<td>838</td>
</tr>
<tr>
<td>MON$IO_STATS</td>
<td>840</td>
</tr>
<tr>
<td>MON$MEMORY_USAGE</td>
<td>841</td>
</tr>
<tr>
<td>MON$RECORD_STATS</td>
<td>842</td>
</tr>
<tr>
<td>MON$STATEMENTS</td>
<td>844</td>
</tr>
<tr>
<td>Использование MON$STATEMENTS для отмены запросов</td>
<td>845</td>
</tr>
<tr>
<td>MON$TABLE_STATS</td>
<td>846</td>
</tr>
<tr>
<td>MON$TRANSACTIONS</td>
<td>847</td>
</tr>
<tr>
<td>SEC$GLOBAL_AUTH_MAPPING</td>
<td>850</td>
</tr>
<tr>
<td>SEC$USERS</td>
<td>850</td>
</tr>
<tr>
<td>SEC$USER_ATTRIBUTES</td>
<td>851</td>
</tr>
<tr>
<td>Appendix G: Таблицы плагинов</td>
<td>853</td>
</tr>
<tr>
<td>Плагин управления пользователями Srp</td>
<td>853</td>
</tr>
<tr>
<td>Таблица PLG$SRP</td>
<td>853</td>
</tr>
<tr>
<td>Представление PLG$SRP_VIEW</td>
<td>853</td>
</tr>
<tr>
<td>Плагин управления пользователями Legacy_UserManager</td>
<td>854</td>
</tr>
<tr>
<td>Таблица PLG$USERS</td>
<td>854</td>
</tr>
<tr>
<td>Представление PLG$VIEW_USERS</td>
<td>855</td>
</tr>
<tr>
<td>Appendix H: Наборы символов и порядки сортировки</td>
<td>857</td>
</tr>
<tr>
<td>Appendix I: License notice</td>
<td>863</td>
</tr>
<tr>
<td>Алфавитный указатель</td>
<td>864</td>
</tr>
</tbody>
</table>
Chapter 1. О руководстве по языку SQL Firebird 4.0

Это руководство описывает язык SQL, поддерживаемый СУБД Firebird 4.0.4.

В руководстве также приводятся практические примеры использования SQL, многие из которых взяты из реальной практики.

1.1. Что содержит данный документ

Данный документ содержит описание языка SQL Firebird. Firebird полностью соответствует международным стандартам SQL, от поддержки типов данных, структур хранения данных, механизмов ссылочной целостности до возможностей управления данными и прав доступа. В СУБД Firebird также реализован надежный процедурный язык — процедурный SQL (PSQL) — для хранимых процедур, триггеров и динамически исполняемых блоков кода. Это те области, о которых идет речь в этом руководстве.

В этом документе не рассматриваются вопросы конфигурация, инструменты командной строки и описание API, и другое не относящееся к языку SQL.

1.2. Авторство

1.2.1. Авторы

Прямой контент

- Дмитрий Филиппов (писатель)
- Александр Карпейкин (писатель)
- Алексей Ковязин (писатель, редактор)
- Дмитрий Кузьменко (писатель, редактор)
- Денис Симонов (писатель, редактор)
- Paul Vinkenoog (писатель, дизайнер)
- Mark Rotteveel (писатель)

Ресурсы

- Дмитрий Еманов
- Adriano dos Santos Fernandes
- Александ Пешков
- Владислав Хорсун
- Claudio Valderrama
- Helen Borrie
• и другие

1.3. Благодарности

Спонсоры

Смотри список спонсоров Firebird 2.5 Language Reference.

Спонсоры Руководства по языку SQL на русском языке

Московская биржа

Московская Биржа — крупнейший в России и Восточной Европе биржевой холдинг, образованный 19 декабря 2011 года в результате слияния биржевых групп ММВБ (основана в 1992) и РТС (основана в 1995). Московская Биржа входит в двадцатку ведущих мировых площадок по объёму торгов ценными бумагами, суммарной капитализации торгемых акций и в десятку крупнейших бирж производных финансовых инструментов.

IBSurgeon (ibase.ru)

Техническая поддержка и инструменты разработчика и администратора для СУБД Firebird.

Есть несколько способов внести свой вклад в документацию Firebird или Firebird в целом:

• Участвуйте в списках рассылки (см. https://www.firebirdsql.org/en/mailing-lists/)

• Сообщайте об ошибках или отправляйте запросы на включение на GitHub (https://github.com/FirebirdSQL/)

• Станьте разработчиком (для документации свяжитесь с нами по firebird-docs, для Firebird в целом используйте список рассылки Firebird-devel)

• Пожертвуйте в Firebird Foundation (см. https://www.firebirdsql.org/en/donate/)

• Станьте платным членом или спонсором Firebird Foundation (см. https://www.firebirdsql.org/en/firebird-foundation/)
Chapter 2. Структура языка SQL

В этом справочнике описан язык SQL, поддерживаемый Firebird.

2.1. Общие сведения

Для начала ознакомьтесь с несколькими замечаниями о некоторых характеристиках, лежащих в основе языковой реализации Firebird.

2.1.1. Подмножества SQL

SQL имеет четыре подмножества SQL, используемых в различных областях применения:

- Динамический SQL (DSQL, Dynamic SQL)
- Процедурный SQL (PSQL, Procedural SQL)
- Встроенный SQL (ESQL, Embedded SQL)
- Интерактивный SQL (ISQL, Interactive SQL)

Динамический SQL является основной частью языка, которая соответствует Части 2 (SQL/Foundation – SQL/Основы) спецификации SQL. DSQL представляет собой конструкции, которые передаются клиентскими приложениями с помощью Firebird API и обрабатываются сервером базы данных.

Процедурный SQL является расширением Динамического SQL, в котором дополнительно присутствуют составные операторы, содержащие локальные переменные, присваивание, циклы и другие процедурные конструкции. PSQL относится к Части 4 (SQL/PSM) спецификации SQL. Изначально расширение PSQL было доступно только лишь в постоянно хранимых в базе модулях (процедурах и триггерах), но сравнительно недавно они стали также доступны в Динамическом SQL (смотри EXECUTE BLOCK).

Встроенный SQL определяет подмножество DSQL, поддерживаемое средством Firebird GPRE. GPRE — приложение-препроцессор, которое позволяет вам внедрять SQL конструкции в ваш непосредственный язык программирования (C, C++, Pascal, Cobol и так далее) и производить обработку этих внедрённых конструкций в правильные вызовы Firebird API. Обратите внимание, что ESQL поддерживает только часть конструкций и выражений DSQL.

Интерактивный SQL подразумевает собой язык, который может быть использован для работы с приложением командной строки Firebird ISQL для интерактивного доступа к базам данных. isql является обычным клиентским приложением. Для него обычный язык — это язык DSQL. Однако приложение поддерживает несколько дополнительных команд.

Оба языковых подмножества, как DSQL, так и PSQL полностью представлены в данном руководстве. Из набора инструментария ни ESQL, ни ISQL не описаны здесь отдельно, за исключением тех мест, где это указано явно.
2.1.2. Диалекты SQL

SQL диалект — это термин, определяющий специфические особенности языка SQL, которые доступны во время доступа с его помощью к базе данных. SQL диалект может быть определён как на уровне базы данных, так и на уровне соединения с базой данных. В настоящее время доступны три диалекта:

- Диалект 1 предназначен исключительно для обеспечения обратной совместимости с устаревшими базами данных из очень старых версий InterBase, v.5 и ниже. Базы данных Dialect 1 сохраняют определенные языковые особенности, которые отличаются от Dialect 3, используемого по умолчанию для баз данных Firebird.
  - Информация о дате и времени хранится в типе данных `DATE`. Имеется тип данных `TIMESTAMP`, который идентичен `DATE`.
  - Двойные кавычки могут использоваться как альтернатива апострофам для разделения строковых данных. Это противоречит стандарту SQL - двойные кавычки зарезервированы для особых синтаксических целей как в стандартном SQL, так и в диалекте 3. Поэтому строки с двойными кавычками следует избегать.
  - Точность типов данных `NUMERIC` и `DECIMAL` меньше, чем в 3-м диалекте и в случае, если значение точности более 9, Firebird хранит такие значения как длинные значения с плавающей точкой.
  - `BIGINT` не является доступным типом данных.
  - Идентификаторы нечувствительны к регистру и всегда должны соответствовать правилам для обычных идентификаторов — см. Раздел Идентификаторы ниже.
  - Хотя значения генератора хранятся как 64-битные целые числа, запрос клиента Dialect 1, например, `SELECT GEN_ID (MyGen, 1)`, вернет значение генератора, усеченное до 32 бит.
- Диалект 2 доступен только в клиентском соединении к Firebird и не может быть применён к базе данных. Он предназначен для того, чтобы помочь в отладке в случае возможных проблем с целостностью данных при проведении миграции с диалекта 1 на 3.
- В базах данных Диалекта 3:
  - Числа с типами данных `DECIMAL` и `NUMERIC` хранятся как длинные значения с фиксированной точкой (масштабируемое целое число) в случае если точность числа больше 9.
  - Тип данных `TIME` доступен и используется для хранения значения только времени.
  - Тип данных `DATE` хранит информацию только о дате.
  - Тип данных `BIGINT` доступен в качестве целого 64-битного типа данных.
  - Двойные кавычки могут использоваться, но только для идентификаторов, которые являются зависимыми от регистра, а не для строковых данных.
  - Все строки должны быть разделены одинарными кавычками (апострофам).
  - Значения генераторов возвращаются как 64-битное целое.
Для вновь разрабатываемых баз данных и приложений настоятельно рекомендуется использовать 3-й диалект. Диалект при соединении с базой данных должен быть таким же, как и базы данных. Исключением является случай миграции с 1-го в 3-й диалект, когда в строке соединения с базой данных используется 2-й диалект.

По умолчанию это руководство описывает семантику SQL третьего диалекта, если только в тексте явно не указывается диалект.

2.1.3. Действия при ошибках

Обработка любого оператора либо успешно завершается, либо прерывается из-за вызванной определёнными условиями ошибки. Обработку ошибок можно производить, как в клиентском приложении, так и на стороне сервера средствами SQL.

2.2. Основные сведения: операторы, предложения, ключевые слова

Основная конструкция SQL — оператор (statement). Оператор описывает, что должна выполнить система управления базами данных с конкретным объектом данных или метаданных, обычно не указывая, как именно это должно быть выполнено. Достаточно сложные операторы содержат более простые конструкции — предложения (clause) и варианты, альтернативы (options).

Предложения (clause)

Предложение описывает некую законченную конструкцию в операторе. Например, предложение WHERE в операторе SELECT и в ряде других операторов (UPDATE, DELETE) задаёт условия поиска данных в таблице (таблицах), подлежащих выборке, изменению, удалению. Предложение ORDER BY задаёт характеристики упорядочения выходного, результирующего, набора данных.

Альтернативы (options)

Альтернативы, будучи наиболее простыми конструкциями, задаются при помощи конкретных ключевых слов и определяют некоторые дополнительные характеристики элементов предложения (допустимость дублирования данных, варианты использования и др.).

Ключевые слова

В SQL существуют ключевые слова и зарезервированные слова. Ключевые слова — это все слова, входящие в лексику (словарь) языка SQL. Ключевые слова можно (но не рекомендуется) использовать в качестве имён, идентификаторов объектов базы данных, внутренних переменных и параметров. Зарезервированные слова — это те ключевые слова, которые нельзя использовать в качестве имён объектов базы данных, переменных или параметров.

Например, следующий оператор будет выполнен без ошибок потому, что ABS является ключевым, но не зарезервированным словом.
CREATE TABLE T (ABS INT NOT NULL);

При выполнении такого оператора будет выдана ошибка потому, что ADD является ключевым и зарезервированным словом.

CREATE TABLE T (ADD INT NOT NULL);

Список зарезервированных и ключевых слов представлен в приложении Зарезервированные и ключевые слова.

2.3. Идентификаторы

Все объекты базы данных имеют имена, которые иногда называют идентификаторами. Максимальная длина идентификатора составляет 63 символа. Существует два типа идентификаторов — имена, похожие по форме на имена переменных в обычных языках программирования, и имена с разделителями (delimited name), которые являются отличительной особенностью языка SQL.

2.3.1. Правила для обычных идентификаторов

- Длина идентификатора не должна превышать 63 символа
- Обычное имя должно начинаться с букв латинского алфавита (первые 7 бит таблицы ASCII), за которой могут следовать буквы (латинского алфавита), цифры, символ подчёркивания и знак доллара. В имени нельзя использовать буквы кириллицы, пробелы, другие специальные символы. Такое имя нечувствительно к регистру, его можно записывать как строчными, так и прописными буквами. Следующие имена с точки зрения системы являются одинаковыми:

```
fullname
FULLNAME
FuLlNaMe
FullName
```

Синтаксис обычных идентификаторов

```
<name> ::=<name><letter> | <name><digit> | <name>_ | <name>$

<letter> ::= <upper letter> | <lower letter>

<upper letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<lower letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
```
2.3.2. Правила для идентификаторов с разделителями

• Длина идентификатора не должна превышать 63 символа.
• Имя должно быть заключено в двойные кавычки, например "anIdentifier".
• Идентификатор может содержать любой символ из набора символов UTF-8, включая символы с диакритическими знаками, пробелы и специальные символы.
• Идентификатор может быть зарезервированным словом.
• Идентификаторы с разделителями чувствительны к регистру во всех контекстах.
• Завершающие пробелы в именах с разделителями игнорируются, как и в случае любой строковой константы.
• Идентификаторы с разделителями доступны только в Диалекте 3. Подробнее о диалектах см. Диалекты SQL.

Синтаксис идентификаторов с разделителями

<delimited name> ::= "<permitted_character>[<permitted_character> ...]"

Идентификатор с разделителями, например "FULLNAME", совпадает с обычными идентификаторами FULLNAME,fullname, FullName и т. д. Причина в том, что Firebird хранит обычные идентификаторы в верхнем регистре, независимо от того, как они были определены или объявлены. Идентификаторы с разделителями всегда сохраняются так как их определили или объявили. Таким образом, идентификатор "FullName" (в кавычках) отличается от FullName (без кавычек), который хранится в метаданных как FULLNAME.

2.4. Литералы

Литералы служат для непосредственного представления данных. Ниже приведены примеры стандартных литералов:

• целочисленные — 0,-34, 45, 0X080000000;
• числа с фиксированной точкой — 0.0, -3.14;
• вещественные — 3.23e-23;
• строковые — 'текст','don''t!';
• двоичные строки — x'48656C6C6F20776F726C64';
• дата — DATE '10.01.2014';
• время — TIME '15:12:56';
• временная отметка — TIMESTAMP '10.01.2014 13:32:02';
• логические — TRUE, FALSE, UNKNOWN;
• неопределённое состояние — null.

Подробней о литералах для каждого из типов данных см. Типы и подтипы данных.

### 2.5. Операторы и специальные символы

Существует набор специальных символов, используемых в качестве разделителей.

```plaintext
<special char> ::= <space> | " | % | & | ' | ( | ) | * | + | , | - |
                  | . | / | : | ; | < | = | > | ? | [ | ] | ^ | { | }
```

Часть этих символов, а так же их комбинации могут быть использованы как операторы (арифметические, строковые, логические), как разделители команд SQL, для квотирования идентификаторов, и для обозначения границ строковых литералов или комментариев.

**Синтаксис операторов**

```plaintext
<operator> ::= <string concatenation operator> |
              <arithmetic operator> |
              <comparison operator> |
              <logical operator>

<string concatenation operator> ::= "||"

<arithmetic operator> ::= * | / | + | - |

<comparison operator> ::= = | <> | != | ^= | ^= | > | < | >= | <= |

<logical operator> ::= NOT | AND | OR
```

Подробнее об операторах см. Выражения.

### 2.6. Комментарии

В SQL скриптах, операторах SQL и PSQL модулях могут встречаться комментарии. Комментарий — это произвольный текст заданный пользователем, предназначенный для пояснения работы отдельных частей программы. Синтаксический анализатор игнорирует текст комментариев.

В Firebird поддерживается два типа комментариев: блочные и однострочные.
Синтаксис

<comment> ::= <block comment> | <single-line comment>

<block comment> ::= /* <character>[<character> ...] */

<single-line comment> ::= -- <character>[<character> ...]<end line>

Блочные комментарии начинаются с символов /* и заканчиваются символами */. Блочные комментарии могут содержать текст произвольной длины и занимать несколько строк.

Однострочные комментарии начинаются с символов -- и действуют до конца текущей строки.

Example 1. Комментарии

```
CREATE PROCEDURE P(APARAM INT)
    RETURNS (B INT)
AS
BEGIN
/* Данный текст не будет учитываться
   при работе процедуры, т.к. является комментарием
*/
    B = A + 1; -- Однострочный комментарий
    SUSPEND;
END
```
Chapter 3. Типы данных

Типы данных используются в случае:

• определения столбца в таблице базы данных в операторе CREATE TABLE или для его изменения с использованием ALTER TABLE;

• при объявлении и редактировании домена оператором CREATE DOMAIN/ALTER DOMAIN;

• при объявлении локальных переменных в хранимых процедурах, функциях, PSQL-блоках и триггерах, при указании аргументов хранимых процедур и функций;

• при описании внешних функций (UDF – функций, определённых пользователем) для указания аргументов и возвращаемых значений;

• при явном преобразовании типов данных в качестве аргумента для функции CAST.

<table>
<thead>
<tr>
<th>Наименование</th>
<th>Размер</th>
<th>Точность и ограничения</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>64 бита</td>
<td>от -2(^{63}) до (2(^{63}) - 1)</td>
<td>64 битное целое. Тип данных доступен только в 3 диалекте.</td>
</tr>
<tr>
<td>BINARY((n))</td>
<td>(n) байт</td>
<td>от 1 до 32 767 байт</td>
<td>Бинарный тип данных фиксированной длины. Является псевдонимом типа CHAR((n)) CHARACTER SET OCTETS.</td>
</tr>
<tr>
<td>BLOB</td>
<td>Переменный</td>
<td>Размер сегмента BLOB ограничивается 64К. Максимальный размер поля BLOB 32 Гб. Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб.</td>
<td>Тип данных с динамически изменяемым размером для хранения больших данных, таких как графика, тексты, оцифрованные звуки. Для сегментированных BLOB базовой структурной единицей является сегмент. Подтип BLOB описывает содержимое.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>8 бит</td>
<td>false, true, unknown</td>
<td>Логический тип данных.</td>
</tr>
<tr>
<td>Наименование</td>
<td>Размер</td>
<td>Точность и ограничения</td>
<td>Описание</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>CHAR(n), CHARACTER(n)</td>
<td>n символов. Размер в байтах зависит от кодировки и количества байт на символ.</td>
<td>от 1 до 32,767 байт</td>
<td>Символьный тип данных фиксированной длины. При извлечении данных, строка дополняется пробелами справа до указанной длины. Если количество символов n не указано, то по умолчанию принимается 1.</td>
</tr>
<tr>
<td>DATE</td>
<td>32 бита</td>
<td>от 01.01.0001 до 31.12.9999</td>
<td>ISC_DATE. Только дата без временной части.</td>
</tr>
<tr>
<td>DECIMAL (precision, scale)</td>
<td>16, 32, 64 или 128 бит в зависимости от точности</td>
<td>precision = от 1 до 38, количество цифр для хранения; scale = от 0 до 38, задаёт количество знаков после десятичной точки.</td>
<td>Число с десятичной точкой, которое после десятичной точки имеет scale разрядов. scale должно быть меньше или равно precision. Пример: DECIMAL(10, 3) содержит число точно в следующем формате: pppppppp.sss.</td>
</tr>
<tr>
<td>DEC_FLOAT(precision)</td>
<td>64 или 128 бит в зависимости от точности</td>
<td>precision = 16 или 34, количество значащих цифр (точность)</td>
<td>SQL:2016 совместимый тип данных точно хранящий десятичные числа с плавающей запятой, основанный на стандарте IEEE 754-2008.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>64 бита</td>
<td>от 2.225 * 10^-308 до 1.797 * 10^308</td>
<td>IEEE двойной точности, 15 цифр, размер зависит от платформы.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>32 бита</td>
<td>от 1.175 * 10^-38 до 3.402 * 10^38</td>
<td>IEEE одинарной точности, 7 цифр</td>
</tr>
<tr>
<td>FLOAT(precision)</td>
<td>32 или 64 бита в зависимости от точности</td>
<td>precision — точность в двоичных числах, может находиться в диапазоне от 1 до 53.</td>
<td>Если precision от 1 до 24 — 32-битное одинарной точности (синоним типа FLOAT). Если precision от 25 до 53 — 64-битное двойной точности (синоним типа DOUBLE PRECISION).</td>
</tr>
<tr>
<td>INTEGER, INT</td>
<td>32 бита</td>
<td>от -2147483648 до 2147483647</td>
<td>Знаковое целое</td>
</tr>
<tr>
<td>INT128</td>
<td>128 бит</td>
<td>от -2^127 до 2^128-1</td>
<td>128-битное целое.</td>
</tr>
<tr>
<td>Наименование</td>
<td>Размер</td>
<td>Точность и ограничения</td>
<td>Описание</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------</td>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUMERIC (precision, scale)</td>
<td>16, 32, 64 или 128 бит в зависимости от точности</td>
<td>precision = от 1 до 38, scale = от 0 до 38, задаёт количество знаков после десятичной точки.</td>
<td>Число с десятичной точкой, которое после десятичной точки имеет scale разрядов. scale должно быть меньше или равно precision. Пример: NUMERIC(10, 3) содержит число точно в следующем формате: pppppppp.sss.</td>
</tr>
<tr>
<td>REAL</td>
<td>32 бита</td>
<td>от 1.175 * 10^{38} до 3.402 * 10^{38}</td>
<td>Является синонимом типа FLOAT.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>16 бита</td>
<td>от -32,768 до 32,767</td>
<td>Короткое знаковое целое.</td>
</tr>
<tr>
<td>TIME [WITHOUT TIME ZONE]</td>
<td>32 бита</td>
<td>0:00 до 23:59:59.9999</td>
<td>ISC_TIME. Время дня без информации о часовом поясе.</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>6 байт</td>
<td>0:00 до 23:59:59.9999</td>
<td>Время дня с информацией о часовом поясе.</td>
</tr>
<tr>
<td>TIMESTAMP [WITHOUT TIME ZONE]</td>
<td>64 бита (2 Х 32 бита)</td>
<td>от 01.01.0001 до 31.12.9999</td>
<td>Дата включающая время без информации о часовом поясе.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>10 байт</td>
<td>от 01.01.0001 до 31.12.9999</td>
<td>Дата включающая время с информацией о часовом поясе.</td>
</tr>
<tr>
<td>VARBINARY(n), BINARY VARYING(n)</td>
<td>n байт.</td>
<td>от 1 до 32,765 байт</td>
<td>Бинарный тип данных переменной длины. Является псевдонимом типа VARCHAR(n) CHARACTER SET OCTETS.</td>
</tr>
<tr>
<td>VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)</td>
<td>n символов.</td>
<td>Размер в байтах зависит от кодировки и количества байт на символ.</td>
<td>Размер символов в байтах с учётом их кодировки не может быть больше 32765. Для этого типа данных, в отличие от CHAR (где по умолчанию предполагается количество символов 1), количество символов n обязательно должно быть указано.</td>
</tr>
</tbody>
</table>

Следует иметь в виду, что временной ряд из дат прошлых веков рассматривается без учёта реальных исторических фактов и так, как будто бы во всем этом диапазоне ВСЕГДА действовал только Григорианский календарь.
3.1. Целочисленные типы данных

Для целых чисел используют целочисленные типы данных SMALLINT, INTEGER, BIGINT (в 3-м диалекте) и INT128. Firebird не поддерживает беззнаковый целочисленный тип данных.

3.1.1. SMALLINT

Тип данных SMALLINT представляет собой 16-битное целое. Он применяется в случаях, когда не требуется широкий диапазон возможных значений для хранения данных.

Числа типа SMALLINT находятся в диапазоне от $-2^{15}$ до $2^{15} - 1$, или от -32768 до 32767.

*Example 2. Использование SMALLINT*

```sql
CREATE DOMAIN DFLAG AS SMALLINT DEFAULT 0 NOT NULL CHECK (VALUE=-1 OR VALUE=0 OR VALUE=1);
CREATE DOMAIN RGB_VALUE AS SMALLINT;
```

3.1.2. INTEGER

Тип данных INTEGER представляет собой 32-битное целое. Сокращённый вариант записи типа данных INT.

Числа типа INTEGER находятся в диапазоне от $-2^{31}$ до $2^{31} - 1$, или от -2,147,483,648 до 2,147,483,647.

*Example 3. Использование INTEGER*

```sql
CREATE TABLE CUSTOMER (  
  CUST_NO INTEGER NOT NULL,  
  CUSTOMER VARCHAR(25) NOT NULL,  
  CONTACT_FIRST VARCHAR(15),  
  CONTACT_LAST VARCHAR(20),  
  ...  
  PRIMARY KEY (CUST_NO)  
);
```

3.1.3. BIGINT

BIGINT — это SQL:99-совместимый 64-битный целочисленный тип данных. Он доступен только в 3-м диалекте. При использовании клиентом диалекта 1, передаваемое сервером значение генератора усекается до 32-х битного целого (INTEGER). При подключении в 3-м диалекте значение генератора имеет тип BIGINT.
Числа типа BIGINT находятся в диапазоне от \(-2^{63}\) до \(2^{63} - 1\), или от \(-9,223,372,036,854,775,808\) до \(9,223,372,036,854,775,807\).

**Example 4. Использование BIGINT**

```sql
CREATE TABLE WHOLELOTTARECORDS (
    ID BIGINT NOT NULL PRIMARY KEY,
    DESCRIPTION VARCHAR(32)
);
```

### 3.1.4. INT128

INT128 — не стандартный 128 битный целочисленный тип данных.

Числа типа INT128 находятся в диапазоне от \(-2^{127}\) до \(2^{127} - 1\).

**Example 5. Использование INT128**

```sql
CREATE PROCEDURE PROC1 (PAR1 INT128)
AS
BEGIN
    -- текст процедуры
END
```

### 3.1.5. Шестнадцатеричный формат для целых чисел

Константы целочисленных типов можно указать в шестнадцатеричном формате.

Начиная с Firebird 4.0.1 числа состоящие из 17-32 шестнадцатеричных цифр будут интерпретированы как INT128.

**Table 2. Константы целочисленных типов в шестнадцатеричном формате**

<table>
<thead>
<tr>
<th>Количество шестнадцатеричных цифр</th>
<th>Тип данных</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>INTEGER</td>
</tr>
<tr>
<td>9-16</td>
<td>BIGINT</td>
</tr>
<tr>
<td>17-32</td>
<td>INT128</td>
</tr>
</tbody>
</table>

Запись SMALLINT в шестнадцатеричном представлении не поддерживается в явном виде, но Firebird будет прозрачно преобразовывать шестнадцатеричное число в SMALLINT, если это необходимо, при условии что оно попадает в допустимый диапазон положительных и отрицательных значений для SMALLINT.

Использование и диапазоны значений чисел шестнадцатеричной нотации более подробно описаны в ходе обсуждения числовых констант в главе под названием “Общие элементы”.
языка”.

Example 6. Использование целых чисел заданных шестнадцатеричном виде

```sql
INSERT INTO MYBIGINTS VALUES (
  -236453287458723,
  328832607832,
  22,
  -56786237632476,
  0x6f55a09d42, -- 478177959234
  0x7fffffffffffff, -- 92237036854775807
  0x80000000, -- -2147483648, t.e. INTEGER
  0x80000000, -- 2147483648, t.e. BIGINT
  0x80000000, -- -2147483648, t.e. INTEGER
  0x80000000, -- 2147483648, t.e. BIGINT
);
```

Шестнадцатеричный INTEGER автоматически приводится к типу BIGINT перед вставкой в таблицу. Однако это происходит после установки численного значения, так 0x80000000 (8 цифр) и 0x08000000 (9 цифр) будут сохранены в разных форматах. Значение 0x80000000 (8 цифр) будет сохранено в формате INTEGER, а 0x080000000 (9 цифр) как BIGINT.

3.2. Типы данных с плавающей точкой

Типы данных с плавающей точкой хранятся в двоичном формате IEEE 754, который включает в себя знак, показатель степени и мантиссу. Firebird имеет две формы типов с плавающей точкой:

- приблизительные числовые типы (или двоичные типы с плавающей точкой);
- десятичные типы с плавающей точкой.

3.2.1. Приблизительные числовые типы

Приблизительные числовые типы плавающей запятой, поддерживаемые Firebird представлены типами 32-битной одинарной точностью и 64-битной двойной точности. Эти типы доступны со следующими именами стандартных типов SQL:

- REAL — 32-битный одинарной точности (синоним типа FLOAT);
- FLOAT — 32-битный одинарной точности;
- DOUBLE PRECISION — 64-битный двойной точности;
- FLOAT(p), где p — точность в двоичных числах
  - 1 <= p <= 32 — 32-битное одинарной точности (синоним типа FLOAT)
  - 33 <= p <= 53 — 64-битное двойной точности (синоним типа DOUBLE PRECISION)
Кроме того, в Firebird имеются нестандартные имена типов:

- **LONG FLOAT** — 64-двойной точности (синоним типа DOUBLE PRECISION);
- **LONG FLOAT(p)**, где \( p \) — точность в двоичных числах. \( 1 \leq p \leq 53 \) — 64-битное двойной точности (синоним типа DOUBLE PRECISION)

Точность этого типов FLOAT и DOUBLE PRECISION является динамической, что соответствует физическому формату хранения, который составляет 4 байта для типа FLOAT и 8 байт для типа DOUBLE PRECISION.

Учитывая особенности хранения чисел с плавающей точкой, этот тип данных не рекомендуется использовать для хранения денежных данных. По тем же причинам не рекомендуется использовать столбцы с данными такого типа в качестве ключей и применять к ним ограничения уникальности.

При проверке данных столбцов с типами данных с плавающей точкой рекомендуется вместо точного равенства использовать выражения проверки вхождения в диапазон, например BETWEEN.

При использовании таких типов данных в выражениях рекомендуется крайне внимательно и серьёзно подойти к вопросу округления результатов расчётов.

**FLOAT**

FLOAT — тип данных для хранения чисел с плавающей точкой.

**Синтаксис**

FLOAT [(bin_prec)]

**Table 3. Параметры типа FLOAT**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_prec</td>
<td>Точность в двоичных цифрах, по умолчанию рано 24</td>
</tr>
<tr>
<td></td>
<td>1 - 24: 32-битное одинарной точности (FLOAT без указания точности)</td>
</tr>
<tr>
<td></td>
<td>25 - 53: 64-битное двойной точности (синоним типа DOUBLE PRECISION)</td>
</tr>
</tbody>
</table>

Тип данных FLOAT по умолчанию представляет собой 32-битный тип с плавающей запятой одинарной точности с приблизительной точностью 7 десятичных знаков после десятичной точки (24 двоичных знака). Числа типа FLOAT находятся в диапазоне от \( 1.175 \times 10^{-38} \) до \( 3.402 \times 10^{38} \).

**FLOAT с указанием точности**

У типа FLOAT может быть указана точность в двоичных числах

Указанная точность \( bin\_prec \) влияет на способ хранения числа.
• 1 <= \textit{bin\_prec} <= 24: 32-битное одинарной точности (синоним типа \texttt{FLOAT} без указания точности)

• 25 <= \textit{bin\_prec} <= 53: 64-битное двойной точности (синоним типа \texttt{DOUBLE\ PRECISION})

В Firebird 3.0 и более ранних версиях поддерживался синтаксис \texttt{FLOAT(\textit{dec\_prec})}, где \textit{dec\_prec} — приблизительная точность в десятичных знаках. Если 0 <= \textit{dec\_prec} <= 7, то тип отображался на 32-битный одинарной точности. Если \textit{dec\_prec} > 7, то отображался на 64-битный двойной точности. Это нестандартное поведение. Данный синтаксис не был документирован ранее.

**REAL**

Тип REAL является синонимом типа \texttt{FLOAT}.

**DOUBLE\ PRECISION**

DOUBLE\ PRECISION — 64-битный тип данных для хранения чисел с плавающей точкой. Он обладает приблизительной точностью 15 цифр после запятой. Числа типа DOUBLE\ PRECISION находятся в диапазоне от \(2.225 \times 10^{-308}\) до \(1.797 \times 10^{308}\).

**LONG\ FLOAT**

Синтаксис:

\[
\text{LONG\ FLOAT}[(\textit{bin\_prec})]
\]

\[
\textit{precision} ::= 1..53
\]

Тип LONG\ FLOAT является синонимом типа DOUBLE\ PRECISION или \texttt{FLOAT(\textit{bin\_prec})}, где 25 <= \textit{bin\_prec} <= 53.

У типа LONG\ FLOAT может быть указана точность в двоичных числах. Указанная точность 1 <= \textit{bin\_prec} <= 53 не влияет на способ хранения — число всегда хранится как 64-битное двойной точности.

В Firebird 3.0 и более ранних версиях поддерживался синтаксис LONG\ FLOAT(\textit{dec\_prec}), где \textit{dec\_prec} — приблизительная точность в десятичных знаках. Независимо от указанной точности число всегда хранится как 64-битное двойной точности. Данный синтаксис не был документирован ранее.

Эти нестандартные имена типов устарели и могут быть удалены в будущей версии.

**3.2.2. Десятичные типы с плавающей точкой**

Начиная с Firebird 4.0 поддерживаются типы десятичных чисел с плавающей запятой.
DECFLOAT

DECFLOAT является числовым типом из стандарта SQL:2016, который точно хранит числа с плавающей запятой. В отличие от DECFLOAT типы FLOAT или DOUBLE PRECISION обеспечивают двоичное приближение предполагаемой точности.

Firebird в соответствии со стандартом IEEE 754-1985 (IEEE 754-2008) реализует типы DECFLOAT64 (DECFLOAT(16)) и DECFLOAT128 (DECFLOAT(34)).

Все промежуточные вычисления осуществляются с использованием 34-значными значениями.

16-значное и 34-значное

“16” и “34” относятся к максимальной точности десятичных цифр.


Синтаксис

```
DECFLOAT[(precision)]
precision ::= 16 | 34
```

Table 4. Диапазон значений DECFLOAT

<table>
<thead>
<tr>
<th>Тип</th>
<th>Максимальная точность</th>
<th>Минимальная экспонента</th>
<th>Максимальная экспонента</th>
<th>Наименьшее значение</th>
<th>Наибольшее значение</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECFLOAT(16)</td>
<td>16</td>
<td>-383</td>
<td>+384</td>
<td>1E-398</td>
<td>9.9..9E+384</td>
</tr>
<tr>
<td>DECFLOAT(34)</td>
<td>34</td>
<td>-6143</td>
<td>+6144</td>
<td>1E-6176</td>
<td>9.9..9E+6144</td>
</tr>
</tbody>
</table>

Обратите внимание, что хотя наименьший показатель степени для DECFLOAT (16) равен -383, наименьшее значение имеет показатель степени -398, что на 15 цифр меньше. И аналогично для DECFLOAT (34), наименьший показатель степени равен -6143, но наименьшее значение имеет показатель степени -6176, что на 33 цифры меньше. Причина заключается в том, что точность была “принесена в жертву”, чтобы можно было хранить меньшее значение.

Это результат того, как хранится значение: как десятичное значение из 16 или 34 цифр и показатель степени. Например, 1.234567890123456e-383 фактически сохраняется как коэффициент 1234567890123456 и показатель степени -398, а 1E-398 сохраняется как коэффициент 1, показатель степени -398.

Тип DECFLOAT следует использовать если вам необходимы вычисления и хранение чисел с большой точностью.
**Chapter 3. Типы данных**

**Example 7. Использование типа DECFLOAT при определении таблицы**

```
CREATE TABLE StockPrice (  
id INT NOT NULL PRIMARY KEY,  
stock DECFLOAT(16),  
...  
);
```

**Example 8. Использование типа DECFLOAT в PSQL**

```
DECLARE VARIABLE v DECFLOAT(34);
```

**Поведение операций с DECFLOAT**

Поведение операций с DECFLOAT, в частности округление и поведение при ошибках, можно настроить с помощью оператора управления `SET DECFLOAT`.

**Длина литералов DECFLOAT**

Значение типа DECFLOAT можно задать числовым литералом в научной нотации, только если мантисса состоит из 20 или более цифр, или абсолютный показатель степени больше 308. В противном случае такие литералы интерпретируются как DOUBLE PRECISION. Точные числовые литералы с 40 или более цифрами — фактически 39 цифр, если они больше максимального значения INT128 также обрабатываются как DECFLOAT (34).

В качестве альтернативы можно использовать строковый литерал и явно привести к желаемому типу DECFLOAT.

Длина литералов типа DECFLOAT ограничена 1024 символами. Для более длинных значений вам придётся использовать научную нотацию. Например, значение 0.0<1020 zeroes>11 не может быть записано как литерал, вместо него вы можете использовать аналогичную научную нотацию: 1.1e-1022. Аналогично 10<1022 zeroes>0 может быть записано как 1.0e1024.

Литералы, содержащие более 34 значащих цифр, округляются с использованием режима округления DECFLOAT установленного для сеанса.

**DECFLOAT и функции**

**Использование обычных функций**

Ряд стандартных скалярных функций можно использовать с выражениями и значениями типа DECFLOAT. Это относится к следующим математическим функциям:

<table>
<thead>
<tr>
<th>Функция</th>
<th>Функция</th>
<th>Функция</th>
<th>Функция</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>CEILING</td>
<td>EXP</td>
<td>FLOOR</td>
</tr>
<tr>
<td>LOG</td>
<td>LOG10</td>
<td>POWER</td>
<td>SIGN</td>
</tr>
<tr>
<td>SQRT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Агрегатные функции SUM, AVG, MIN и MAX тоже работают с типом DECFLOAT. Все статистические агрегатные функции (такие как STDEV или CORR, но не ограничено ими) могут работать с данными типа DECFLOAT.

Специальные функции для DECFLOAT

Firebird поддерживает 4 функции, которые созданы специально для поддержки типа DECFLOAT:

**COMPARE_DECFLOAT**

сравнивает два значения DECFLOAT как равные, разные или неупорядоченные

**NORMALIZE_DECFLOAT**

принимает единственный аргумент DECFLOAT и возвращает его в простейшей форме

**QUANTIZE**

принимает два аргумента DECFLOAT и возвращает первый аргумент, масштабированный с использованием второго значения в качестве образца

**TOTALORDER**

выполняет точное сравнение двух значений DECFLOAT

Семантика сравнения

Замыкающие нули в значениях десятичных чисел с плавающей запятой сохраняются. Например, 1.0 и 1.00 — это два различных представления. Это порождает различные семантики сравнения для типа данных DECFLOAT, как показано ниже.

Сравнение числовых значений

Замыкающие нули игнорируются в сравнениях. Например, 1.0 равно 1.00. По умолчанию такой тип сравнения используется для индексирования, сортировки, разбивки таблицы, оценки предикатов и других функций — короче говоря, везде, где сравнение выполняется неявно или в предикатах.

*Example 9. Сравнение числовых значений*

```sql
create table stockPrice (stock DECFLOAT(16));

insert into stockPrice values (4.2);

insert into stockPrice values (4.2000);

insert into stockPrice values (4.6125);

insert into stockPrice values (4.20);
```

Chapter 3. Типы данных

46
Сравнение TotalOrder

Замыкающие нули учитываются при сравнении. Например, 1.0 > 1.00. Каждое значение DECIMAL имеет порядок в семантике сравнения TotalOrder.

Согласно семантике TotalOrder, порядок различных значений определяется так, как показано в следующем примере:

```
-nan < -snan < -inf < -0.1 < -0.10 < -0 < 0 < 0.10 < 0.1 < inf < snan < nan
```

Обратите внимание на то, что отрицательный нуль меньше положительного нуля при сравнении TotalOrder.

Запросить сравнение TotalOrder в предикатах можно при помощи встроенной функции TOTALORDER().

Example 10. Сравнение TotalOrder

Для курсов акций может быть важным знать точность данных. Например, если курсы обычно указываются с точностью в пять знаков после запятой, а курс равен $4.2$, тогда неясно, равна цена $4.2000$, $4.2999$ или чему-то, лежащему между этими двумя значениями.

```sql
create table stockPrice (stock DECIMAL(16));

insert into stockPrice values (4.2);
insert into stockPrice values (4.2000);
insert into stockPrice values (4.6125);
```
Порядок, в котором возвращаются арифметически одинаковые значения, имеющие различное количество замыкающих нулей, не определен. Следовательно, ORDER BY по столбцу DECFLOAT со значениями 1.0 и 1.00 возвращает два значения в произвольном порядке. Аналогично, DISTINCT возвращает либо 1.0, либо 1.00.

**Поддержка в клиентских приложениях**

Библиотека fbclient версии 4.0 имеет нативную поддержку типа DECFLOAT. Однако более старые версии клиентской библиотеки ничего не знают о типе DECFLOAT. Для того чтобы старые приложения умели работать с типом DECFLOAT вы можете настроить отображение значений DECFLOAT на другие доступные типы данных с помощью оператора `SET BIND`.

**Примеры:**

```sql
SET BIND OF DECFLOAT TO LEGACY;
-- значения столбцов типа DECFLOAT будут преобразованы в тип DOUBLE PRECISION

SET BIND OF DECFLOAT TO DOUBLE PRECISION;
-- другой вариант

SET BIND OF DECFLOAT(16) TO CHAR;
-- значения столбцов типа DECFLOAT(16) будут преобразованы в тип CHAR(23)

SET BIND OF DECFLOAT(34) TO CHAR;
-- значения столбцов типа DECFLOAT(34) будут преобразованы в тип CHAR(42)

SET BIND OF DECFLOAT TO NUMERIC(18, 4);
-- значения столбцов типа DECFLOAT будут преобразованы в тип NUMERIC(18, 4)

SET BIND OF DECFLOAT TO NATIVE;
-- возвращает значения столбцов типа DECFLOAT в нативном типе
```

Различные привязки полезны, если вы планируете использовать значения DECFLOAT со старым клиентом, не поддерживающим собственный формат. Можно выбирать между строками (идеальная точность, но плохая поддержка для дальнейшей обработки), значения с плавающей запятой (идеальная поддержка для дальнейшей обработки, но с плохой точностью) или масштабированные целые числа (хорошая поддержка дальнейшей
обработки и требуемая точность, но диапазон значений очень ограничен). Когда
используется инструмент, подобный универсальному GUI-клиенту, выбор привязки к CHAR
подходит в большинстве случаев.

3.3. Типы данных с фиксированной точкой

Данные типы данных позволяют применять их для хранения денежных значений и
обеспечивают предсказуемость операций умножения и деления.

Firebird предлагает два типа данных с фиксированной точкой: NUMERIC и DECIMAL. В
соответствии со стандартом оба типа ограничивают хранимое число объявленным
масштабом (количество чисел после запятой). При этом подход к тому, как ограничивается
точность для типов разный: для столбцов NUMERIC точность является такой, “как объявлено”,
в то время, как DECIMAL столбцы могут получать числа, чья точность, по меньшей мере,
равна тому, что было объявлено.

Например, NUMERIC(4, 2) описывает число, состоящее в общей сложности из четырёх цифр,
включая 2 цифры после запятой; итого 2 цифры до запятой, 2 после. При записи в столбец с
этим типом данных значений 3.1415 в столбце NUMERIC(4, 2) будет сохранено значение 3,14.

Для данных с фиксированной точкой общим является форма декларации, например
NUMERIC(p, s). Здесь важно понять, что в этой записи s— это масштаб, а не интуитивно
предсказываемое “количество знаков после запятой”. Для “визуализации” механизма
хранения данных запомните для себя процедуру:

• При сохранении в базу данных число умножается на 10 (10^s), превращаясь в целое;
• При чтении данных происходит обратное преобразование числа.

Способ физического хранения данных в СУБД зависит от нескольких факторов:
декларируемой точности, диалекта базы данных, типа объявления.

Table 5. Способ физического хранения чисел с фиксированной точкой

<table>
<thead>
<tr>
<th>Точность</th>
<th>Тип данных</th>
<th>Диалект 1</th>
<th>Диалект 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>NUMERIC</td>
<td>SMALLINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>1 - 4</td>
<td>DECIMAL</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>5 - 9</td>
<td>NUMERIC и DECIMAL</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>10 - 18</td>
<td>NUMERIC и DECIMAL</td>
<td>DOUBLE PRECISION</td>
<td>BIGINT</td>
</tr>
<tr>
<td>19 - 38</td>
<td>NUMERIC и DECIMAL</td>
<td>INT128</td>
<td>INT128</td>
</tr>
</tbody>
</table>

3.3.1. NUMERIC

Формат объявления данных

NUMERIC
| НУМЕРИЧИЧЕСКИЙ(precision) |
| НУМЕРИЧИЧЕСКИЙ(precision, scale) |

Table 6. Параметры типа NUMERIC

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Точность. Может быть в диапазоне от 1 до 38. По умолчанию 9.</td>
</tr>
<tr>
<td>scale</td>
<td>Масштаб. Может быть в диапазоне от 0 до precision. По умолчанию 0.</td>
</tr>
</tbody>
</table>

В зависимости от точности precision и масштаба scale СУБД хранит данные по-разному.

Приведём примеры того, как СУБД хранит данные в зависимости от формы их объявления:

- **NUMERIC(4)** stored as **SMALLINT** (exact data)
- **NUMERIC(4,2)** stored as **SMALLINT** (data * 10^2)
- **NUMERIC(10,4)** (Dialect 1) **DOUBLE PRECISION**
  
  (Dialect 3) **BIGINT** (data * 10^4)
- **NUMERIC(38, 6)** stored as **INT128** (data * 10^6)

Всегда надо помнить, что формат хранения данных зависит от точности. Например, вы задали тип столбца NUMERIC(2, 2), предполагая, что диапазон значений в нем будет -0.99...0.99. Однако в действительности диапазон значений в столбце будет -327.68..327.67, что объясняется хранением типа данных NUMERIC(2, 2) в формате SMALLINT. Фактически типы данных NUMERIC(4, 2), NUMERIC(3, 2) и NUMERIC(2, 2) являются одинаковыми.

Таким образом, для реального хранения данных в столбце с типом данных NUMERIC(2, 2) в диапазоне -0.99...0.99 для него надо создавать ограничение.

### 3.3.2. DECIMAL

#### Формат объявления данных

| DECIMAL |
| DECIMAL(precision) |
| DECIMAL(precision, scale) |

Table 7. Параметры типа DECIMAL

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Точность. Может быть в диапазоне от 1 до 38. По умолчанию 9.</td>
</tr>
<tr>
<td>scale</td>
<td>Масштаб. Может быть в диапазоне от 0 до precision. По умолчанию 0.</td>
</tr>
</tbody>
</table>

Формат хранения данных в базе во многом аналогичен NUMERIC, хотя существуют некоторые особенности, которые проще всего пояснить на примере.
Приведём примеры того, как СУБД хранит данные в зависимости от формы их объявления:

<table>
<thead>
<tr>
<th>DECIMAL(4)</th>
<th>stored as</th>
<th>INTEGER (exact data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL(4,2)</td>
<td></td>
<td>INTEGER (data * 10^2)</td>
</tr>
<tr>
<td>DECIMAL(10,4) (Dialect 1)</td>
<td>DOUBLE PRECISION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Dialect 3)</td>
<td>BIGINT (data * 10^4)</td>
</tr>
<tr>
<td>DECIMAL(38, 6)</td>
<td></td>
<td>INT128 (data * 10^6)</td>
</tr>
</tbody>
</table>

3.3.3. Точность арифметических операций

Функции MIN, MAX, SUM, AVG работают со всеми точными числовыми типами. SUM и AVG являются точными, если обрабатываемая запись имеет точный числовой тип, а масштабированная сумма соответствует 64 или 128 битам: в противном случае возникает исключение переполнения. SUM и AVG никогда не вычисляются с использованием арифметики с плавающей запятой, если тип данных столбца не является приблизительным числом.

Функции MIN и MAX для точного числового столбца возвращают точный числовой результат, имеющий ту же точность и масштаб, что и столбец. SUM и AVG для точного числового типа возвращает результат типа NUMERIC (18 | 38) S) или DECIMAL (18 | 38), где S - масштаб столбца. Стандарт SQL определяет масштаб результата в таких случаях, в то время как точность SUM или AVG для столбцов с фиксированной точкой определяется реализацией: мы определяем его как 18 или 38 (если точность аргумента 18 или 38).

Если два операнда OP1 и OP2 являются точными числами с масштабами S1 и S2 соответственно, то OP1 + OP2 и OP1 - OP2 являются точными числами с точностью 18 или 38 (если один из аргументов с точностью 38) и масштабом равному наибольшему из значений S1 и S2, тогда как для OP1 * OP2 и OP1 / OP2 являются точными числами с точностью 18 или 38 (если точность аргументов 18 или 38) и шкалой S1 + S2. Масштабы этих операций, кроме деления, определяются стандартом SQL. Точность всех этих операций и масштаб при делении стандартом не регламентируются, а определяются реализацией: Firebird определяет точность как 18 или 38 (если точность аргументов 18 или 38), а масштаб деления как S1 + S2, такой же, что определён стандартом в для умножения.

Всякий раз, когда выполняются арифметические операции с точными числами типами, в случае потери точности будет сообщено об ошибке переполнения, а не возвращено неправильное значение. Например, если столбец DECIMAL (18,4) содержит наиболее отрицательное значение этого типа, -922337203685477.5808, попытка разделить этот столбец на -1 будет сообщать об ошибке переполнения, поскольку истинный результат превышает наибольшее положительное значение, которое может быть представлено в типе, а именно 922337203685477.5807.

Если один operand является точным числом, а другой приблизительным числом, то результатом любого из четырёх диадических операторов будет типа DOUBLE PRECISION. (В стандарте говорится, что результат является приближенным числом с точностью, по крайней мере, такой же как точность приближенного числового операнда: Firebird удовлетворяет этому требованию, всегда используя DOUBLE PRECISION, поскольку этот тип является максимальным приближенным числовым типом, который предоставлен в Firebird.)
3.4. Типы данных для работы с датой и временем

В СУБД Firebird для работы с данными, содержащими дату и время, используются типы данных DATE, TIME и TIMESTAMP. В 3-м диалекте присутствуют все три вышеназванных типа данных, а в 1-м для операций с датой и временем доступен только тип данных DATE, который не тождественен типу данных DATE 3-го диалекта, а является типом данных TIMESTAMP из 3-го диалекта.

В диалекте 1 тип DATE может быть объявлен как TIMESTAMP. Такое объявление является рекомендуемым для новых баз данных в 1-м диалекте.

Доли секунды

В типах TIMESTAMP и TIME Firebird хранит секунды с точностью до десять тысячных долей. Если вам необходима более низкая гранулярность, то точность может быть указана явно в виде тысячных, сотых или десятых долей секунды в базах данных в 3 диалекте и ODS 11 и выше.

Несколько полезных сведений о точности секунд

Временная часть типов TIME или TIMESTAMP представляет собой 4-байтовое целое (WORD) вмещающее значение времени с долями секунды, и хранящаяся как количество десяти тысячных долей секунды прошедших с полуночи. Фактическая точность значений полученных из time(stamp) функций и переменных будет следующей:

- CURRENT_TIME — по умолчанию имеет точность до секунды, точность до миллисекунд может быть указана следующим образом CURRENT_TIME (0 | 1 | 2 | 3)

- CURRENT_TIMESTAMP — по умолчанию имеет точность до миллисекунды, точность от секунд до миллисекунд может быть указана следующим образом CURRENT_TIMESTAMP (0 | 1 | 2 | 3)

- LOCALTIME — по умолчанию имеет точность до секунды, точность до миллисекунд может быть указана следующим образом LOCALTIME (0 | 1 | 2 | 3)

- LOCALTIMESTAMP — по умолчанию имеет точность до миллисекунды, точность от секунд до миллисекунд может быть указана следующим образом LOCALTIMESTAMP (0 | 1 | 2 | 3)

- Литерал 'NOW' имеет точность до миллисекунд;

- Функции DATEADD и DATEDIFF поддерживают точность до десятих долей миллисекунд;

- Функция EXTRACT возвращает значения с точностью до десятих долей миллисекунды для аргументов SECOND и MILLISECOND;
Хранение типов с часовым поясом

Типы данных с поддержкой часовых поясов сохраняются в виде значений в формате UTC (смещение 0) с использованием структуры TIME или TIMESTAMP + два дополнительных байта для информации о часовом поясе (либо смещение в минутах, либо идентификатор именованного часовового пояса).

Хранение в формате UTC позволяет Firebird индексировать и сравнивать два значения в разных часовых поясах.

При хранении в UTC есть некоторые предостережения:

- Когда вы используете именованные зоны и правила часовых поясов для этой зоны меняются, время в формате UTC остается прежним, но местное время в названной зоне может измениться.
- Для типа данных TIME WITH TIME ZONE при вычислении смещения часового пояса для именованной зоны для получения местного времени в зоне применяются правила, действующие на 1 января 2020 года, чтобы гарантировать стабильное значение. Это может привести к неожиданным или сбивающим с толку результатам.

3.4.1. DATE

В 3-м диалекте тип данных DATE, как это и следует предположить из названия, хранит только одну дату без времени. В 1-м диалекте тип DATE эквивалентен типу TIMESTAMP и хранит дату вместе со временем.

Допустимый диапазон хранения от 01.01.0001 н.э. до 31.12.9999 н.э.

В случае необходимости сохранять в 1 диалекте только значения даты, без времени, при записи в таблицу добавляйте время к значению даты в виде литерала ’00:00:00.0000’.

Example 11. Пример использования DATE

```sql
CREATE TABLE DataLog(
    id BIGINT NOT NULL,
    bydate DATE
);
```

```sql
... AS
    DECLARE BYDATE DATE;
BEGIN
    ...
```
3.4.2. TIME

Синтаксис

TIME [{WITH | WITHOUT} TIME ZONE]
EXTENDED TIME WITH TIME ZONE

Этот тип данных доступен только в 3-м диалекте. Позволяет хранить время дня в диапазоне от 00:00:00.0000 до 23:59:59.9999. По умолчанию тип TIME не содержит информацию о часовом поясе. Для того чтобы тип TIME включал информацию о часовом поясе необходимо использовать его с модификатором WITH TIME ZONE.

EXTENDED TIME WITH TIME ZONE предназначен для использования только при общении с клиентами, он решает проблему представления правильного времени на клиентах, у которых отсутствует библиотека ICU. Нельзя использовать расширенные типы данных в таблицах, процедурах и т.д. Единственный способ использовать эти типы данных — это приведение типов данных, включая инструкцию SET BIND (дополнительную информацию смотри в SET BIND OF).

Example 12. Пример использования TIME

```
CREATE TABLE DataLog(
    id BIGINT NOT NULL,
    bytime TIME WITH TIME ZONE
);
...
AS
    DECLARE BYTIME TIME; -- без часового пояса
    DECLARE BYTIME2 TIME WITHOUT TIME ZONE; -- без часового пояса
    DECLARE BYTIME3 TIME WITH TIME ZONE; -- с информацией о часовом поясе
BEGIN
...
```

См. также EXTRACT, AT, LOCALTIME, CURRENT_TIME, Преобразование строк в дату и время.

3.4.3. TIMESTAMP
Синтаксис

TIMESTAMP [{WITH | WITHOUT} TIME ZONE]
EXTENDED TIMESTAMP WITH TIME ZONE

Этот тип данных хранит временную метку (дату вместе со временем) в диапазоне от 01.01.0001 00:00:00.0000 до 31.12.9999 23:59:59.9999. По умолчанию тип TIMESTAMP не содержит информацию о часовом поясе. Для того чтобы тип TIMESTAMP включал информацию о часовом поясе необходимо использовать его с модификатором WITH TIME ZONE.

EXTENDED TIMESTAMP WITH TIME ZONE предназначен для использования только при общении с клиентами, он решает проблему представления правильного времени на клиентах, у которых отсутствует библиотека ICU. Нельзя использовать расширенные типы данных в таблицах, процедурах и т.д.

Единственный способ использовать эти типы данных — это приведение типов данных, включая инструкцию SET BIND (дополнительную информацию смотри в SET BIND OF).

Example 13. Пример использования TIME

```sql
CREATE TABLE DataLog(
    id BIGINT NOT NULL,
    bydate TIMESTAMP WITH TIME ZONE
);

AS
DECLARE BYDATE TIMESTAMP; -- без часового пояса
DECLARE BYDATE2 TIMESTAMP WITHOUT TIME ZONE; -- без часового пояса
DECLARE BYDATE3 TIMESTAMP WITH TIME ZONE; -- с информацией о часовом поясе
BEGIN
...
```

См. также EXTRACT, AT, LOCALTIMESTAMP, CURRENT_TIMESTAMP, Преобразование строк в дату и время.

3.4.4. Часовой пояс сеанса

Часовой пояс сеанса как следует из названия может быть разным для каждого соединения с базой данных. Он может быть установлен с помощью DPB isc_dbp_session_time_zone, а если нет, то он будет считан из параметра DefaultTimeZone конфигурации firebird.conf. Если параметр DefaultTimeZone не установлен, то часовой пояс сеанса будет тем же, что используется операционной системой в которой запущен процесс Firebird.
Часовой пояс сеанса может быть изменён с помощью оператора `SET TIME ZONE` или сброшен в исходное значение с помощью `SET TIME ZONE LOCAL`.

### Получение часового пояса сеанса

Получить текущий часовой пояс сеанса можно с использованием функции `RDB$GET_CONTEXT` с аргументами 'SYSTEM' для пространства имён и 'SESSION_TIMEZONE' в качестве имени переменной.

**Example 14. Получение часового пояса сеанса**

```sql
set timezone '-02:00';
select rdb$get_context('SYSTEM', 'SESSION_TIMEZONE') from rdb$database;
-- returns -02:00

set timezone 'America/Sao_Paulo';
select rdb$get_context('SYSTEM', 'SESSION_TIMEZONE') from rdb$database;
-- returns America/Sao_Paulo
```

#### 3.4.5. Формат часового пояса

Часовой пояс может быть задан строкой с регионом часового пояса (например, America/Sao_Paulo), или в виде смещения “часов:минут” относительно GMT (например, -03:00). Список региональных часовых поясов и их идентификаторов можно посмотреть в таблице `RDB$TIME_ZONES`. Правила преобразования региональных часовых поясов в смещение в минутах можно получить с помощью процедуры `RDB$TIME_ZONE_UTIL.TRANSITIONS`.

{TIME | TIMESTAMP} WITH TIMEZONE считается равным другому {TIME | TIMESTAMP} WITH TIMEZONE, если их преобразование в UTC равно, например `time '10:00 -02' = time '09:00 -03'`, поскольку оба времени эквивалентны `time '12:00 GMT'`. Это также справедливо в контексте ограничения UNIQUE и для сортировки.

### Региональная семантика TIME WITH TIME ZONE

По определению региональные часовые пояса зависят от момента (дата и время — или `timestamp`), чтобы узнать его смещение UTC относительно GMT. Но Firebird также поддерживает региональные часовые пояса в значениях `TIME WITH TIME ZONE`.

При построении значения `TIME WITH TIME ZONE` из литерала или его преобразования, значение UTC должно быть вычислено и не может быть изменено, поэтому текущая дата может не использоваться. В этом случае используется фиксированная дата 2020-01-01. Таким образом, при сравнении `TIME WITH TIME ZONE` с различными часовыми поясами сравнение выполняется аналогично тому, как они представляют собой значения `TIMESTAMP WITH TIME ZONE` на заданную дату.

Однако при преобразовании между типами `TIMESTAMP` в `TIME WITH TIME ZONE` эта фиксированная дата не используется, в противном случае могут наблюдаться некоторые
странные преобразования, когда текущая дата имеет другое смещение (из-за изменений летнего времени), чем в 2020-01-01. В этом случае при преобразовании TIME WITH TIME ZONE в TIMESTAMP WITH TIME ZONE сохраняется часть времени (если это возможно). Например, если текущая дата 2020-05-03, эффективное смещение в часовом поясе America/Los_Angeles равно -420, а его эффективное смещение в 2020-01-01 равно -480, но cast(time '10:00:00 America/Los_Angeles' as timestamp with time zone) даст в результате 2020-05-03 10:00:00.0000 America/Los_Angeles вместо корректировки временной части.

Но в дату, когда начинается летнее время, пропущен час, например, для часового пояса America/Los_Angeles в 2021-03-14 нет времени с 02:00:00 до 02:59:59. В этом случае преобразование выполняется как построение литерала, и час корректируется до следующего допустимого значения. Например, в 2021-03-14 cast(time '02:10:00 America/Los_Angeles' as timestamp with time zone) даст результат 2021-03-14 03:10:00.0000 America/Los_Angeles.

### 3.4.6. Литералы даты и времени

Для записи литералов даты и времени в Firebird используются сокращенные "C-style" выражения. Строковое представление даты и времени должно быть в одном из разрешённых форматов.

#### Синтаксис

| date_literal | ::= DATE <date> |
| time_literal | ::= TIME <time> |
| timestamp_literal | ::= TIMESTAMP <timestamp> |

| date | ::= |
| YYYY<p> | MM<p>DD | |
| MM<p>DD | YYYY | |
| DD<p>MM | YYYY | |
| MM<p>DD | YY | |
| DD<p>MM | YY | |

| time | ::= HH:mm[:SS[.NNNN]] [<time zone>] |
| timestamp | ::= <date> <time> |

| time zone | ::= |
| <time zone region> | |
| [+/-] <hour displacement> [<minute displacement>] |

| p | ::= whitespace | . | : | | , | - | / |

#### Table 8. Описание формата даты и времени
### Аргумент | Описание
--- | ---
datetime | Строковое представление даты-времени.
date | Строковое представление даты.
time | Строковое представление времени.
YYYY | Год из четырёх цифр.
YY | Последние две цифры года (00-99).
MM | Месяц. Может содержать 1 или 2 цифры (1-12 или 01-12). В качестве месяца допустимо также указывать трёх буквенно сокращение или полное наименование месяца на английском языке, регистр не имеет значение.
DD | День. Может содержать 1 или 2 цифры (1-31 или 01-31).
HH | Час. Может содержать 1 или 2 цифры (0-23 или 00-23).
mm | Минуты. Может содержать 1 или 2 цифры (0-59 или 00-59).
SS | Секунды. Может содержать 1 или 2 цифры (0-59 или 00-59).
NNNN | Десятитысячные доли секунды. Может содержать от 1 до 4 цифр (0-9999).
p | Разделитель, любой из разрешённых символов, лидирующие и завершающие пробелы игнорируются.
time zone region | Один из часовых поясов связанных с регионом.
hour displacement | Смещение времени для часов относительно GMT.
minute displacement | Смещение времени для минут относительно GMT.

Правила:

- В формате Год-Месяц-День, год обязательно должен содержать 4 цифры;
- Для дат в формате с завершающим годом, если в качестве разделителя дат используется точка ".", то дата интерпретируется в форме День-Месяц-Год, для остальных разделителей она интерпретируется в форме Месяц-День-Год;
- Если год не указан, то в качестве года берётся текущий год;
- Если указаны только две цифры года, то для получения столетия Firebird использует алгоритм скользящего окна. Задача заключается в интерпретации двух символьного значения года как ближайшего к текущему году в интервале предшествующих и последующих 50 лет;
- Если в строковом представлении времени присутствует часовой пояс или смещение времени, то тип литерала будет WITH TIME ZONE, в противном случае WITHOUT TIME ZONE;
- Если не указан один из элементов времени, то оно принимается равным 0.

⚠️ Настоятельно рекомендуем в литералах дат использовать только формы с полным указанием года в виде 4 цифр во избежание путаницы.
Example 15. Примеры литералов дат и времени

```sql
SELECT
    date '04.12.2014' AS d1, -- DD.MM.YYYY
    date '12-04-2014' AS d2, -- MM-DD-YYYY
    date '12/04/2014' AS d3, -- MM/DD/YYYY
    date '04.12.14' AS d4, -- DD.MM.YY
    -- DD.MM в качестве года берётся текущий
date '04.12' AS d5, -- MM/DD в качестве года берётся текущий
date '2014/12/04' AS d6,
date '2014.12.04' AS d7, -- YYYY.MM.DD
    date '2014-12-04' AS d8, -- YYYY-MM-DD
time '11:37' AS t1, -- HH:mm
    time '11:37:12' AS t2, -- HH:mm:ss
    time '11:31:12.1234' AS t3, -- HH:mm:ss.nnnn
    -- HH:mm:ss.nnnn +hh
    time '11:31:12.1234 +03' AS t4,
    -- HH:mm:ss.nnnn +hh:mm
    time '11:31:12.1234 +03:30' AS t5,
    -- HH:mm:ss.nnnn tz
    time '11:31:12.1234 Europe/Moscow' AS t6,
    -- HH:mm tz
time '11:31 Europe/Moscow' AS t7,
    -- DD.MM.YYYY HH:mm
timestamp '04.12.2014 11:37' AS dt1,
    -- MM/DD/YYYY HH:mm
timestamp '12/04/2014 11:37:12' AS dt2,
    -- DD.MM.YYYY HH:mm:ss
    timestamp '04.12.2014 11:31:12.1234' AS dt3,
    -- YYYY-MM-DD HH:mm:ss.nnnn +hh:mm
    timestamp '2014-12-04 11:31:12.1234 +03:00' AS dt4,
    -- DD.MM.YYYY HH:mm:ss.nnnn tz
    timestamp '04.12.2014 11:31:12.1234 Europe/Moscow' AS dt5
FROM rdb$database
```

Обратите внимание, что эти сокращённые выражения вычисляются сразу же во время синтаксического анализа (подготовки запроса или компиляции процедуры, функции или триггера). До Firebird 4.0 сокращённые выражения позволялись также для специальных строковых литералов 'NOW', 'TODAY', 'TOMORROW', 'YESTERDAY'. Использование таких выражений в компилируемом PSQL приводило к тому, что значение "замораживалось" на момент компиляции, и возвращалось не актуальное значение. Поэтому в Firebird 4.0 сокращённые выражения для таких строковых литералов запрещены, однако вы можете использовать их при приведении типа оператором CAST.
См. также:
Преобразование строк в дату и время.

3.4.7. Операции, использующие значения даты и времени

Благодаря способу хранения даты и времени с этими типами возможны арифметические операции вычитания из более поздней даты (времени) более раннюю. Дата представлена количеством дней с "нулевой даты" – 17 ноября 1858 г. Время представлено количеством секунд (с учётом десятитысячных долей), прошедших с полуночи.

Table 9. Арифметические операции для типов данных даты и времени

<table>
<thead>
<tr>
<th>Операнд 1</th>
<th>Оператор</th>
<th>Операнд 2</th>
<th>Результат</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>+</td>
<td>TIME</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>DATE</td>
<td>+</td>
<td>TIME WITH TIME ZONE</td>
<td>TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>DATE</td>
<td>+</td>
<td>Числовое значение n</td>
<td>DATE, увеличенная на n целых дней (дробная часть игнорируется).</td>
</tr>
<tr>
<td>TIME</td>
<td>+</td>
<td>DATE</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>+</td>
<td>DATE</td>
<td>TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>TIME</td>
<td>+</td>
<td>Числовое значение n</td>
<td>TIME, увеличенное на n секунд (дробная часть учитывается).</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>+</td>
<td>Числовое значение n</td>
<td>TIME WITH TIME ZONE, увеличенное на n секунд (дробная часть учитывается).</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>+</td>
<td>Числовое значение n</td>
<td>TIMESTAMP, где дата будет увеличиваться на количество дней и на часы, представленную числом n - поэтому “+ 2.75” сдвинет дату вперед на 2 дня и 18 часов.</td>
</tr>
</tbody>
</table>
### Операнды и операторы

<table>
<thead>
<tr>
<th>Операнд 1</th>
<th>Оператор</th>
<th>Операнд 2</th>
<th>Результат</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>+</td>
<td>Числовое значение ( n )</td>
<td>TIMESTAMP WITH TIME ZONE, где дата будет увеличиваться на количество дней и на часть дня, представленную числом ( n ) - поэтому “+ 2.75” сдвинет дату вперед на 2 дня и 18 часов.</td>
</tr>
<tr>
<td>DATE</td>
<td>-</td>
<td>DATE</td>
<td>Количество дней в интервале как DECIMAL (9, 0).</td>
</tr>
<tr>
<td>DATE</td>
<td>-</td>
<td>Числовое значение ( n )</td>
<td>DATE, уменьшенная на ( n ) целых дней (дробная часть игнорируется).</td>
</tr>
<tr>
<td>TIME</td>
<td>-</td>
<td>TIME</td>
<td>Количество секунд в интервале как DECIMAL (9, 4).</td>
</tr>
<tr>
<td>TIME</td>
<td>-</td>
<td>( n )</td>
<td>TIME, уменьшенное на ( n ) секунд (дробная часть учитывается).</td>
</tr>
<tr>
<td>TIME</td>
<td>-</td>
<td>TIME WITH TIME ZONE</td>
<td>Значение без часового пояса преобразуется в WITH TIME ZONE в часом пояс течущего сеанса. Возвращается количество секунд в интервале между UTC значениями как DECIMAL(9, 4). То же правило действует при изменении порядка операндов.</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>-</td>
<td>TIME WITH TIME ZONE</td>
<td>Возвращается количество секунд в интервале между UTC значениями как DECIMAL(9, 4).</td>
</tr>
<tr>
<td>Операнд 1</td>
<td>Оператор</td>
<td>Операнд 2</td>
<td>Результат</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>-</td>
<td>TIMESTAMP</td>
<td>Количество дней и части дня в интервале как DECIMAL (18, 9).</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>-</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Значение без часового пояса преобразуется в WITH TIME ZONE в часовом поясе текущего сеанса. Количества дней и части дня в интервале между UTC значениями как DECIMAL (18, 9). То же правило действует при изменении порядка operandов.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>-</td>
<td>n</td>
<td>TIMESTAMP, где дата будет уменьшена на количество дней, и часть дня, представленную числом n - поэтому “-2.25” сдвинет дату назад на 2 дня и 6 часов.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>-</td>
<td>n</td>
<td>TIMESTAMP WITH TIME ZONE, где дата будет уменьшена на количество дней, и часть дня, представленную числом n - поэтому “-2.25” сдвинет дату назад на 2 дня и 6 часов.</td>
</tr>
</tbody>
</table>

Одно значение даты/времени может быть вычтено из другого если:

- Оба значения имеют один и тот же тип даты/времени;
- Первый operand является более поздним, чем второй.

В диалекте 1 тип DATE рассматривается как TIMESTAMP.

См. также:
3.4.8. Дополнительные функции для поддержки часовых поясов

Firebird 4 предоставляет ряд функций для получения информации о часовых поясах.

**Виртуальная таблица RDB$TIME_ZONES**

Виртуальная таблица со списком часовых поясов, поддерживаемых Firebird.

См. также RDB$TIME_ZONES в приложении “Системные таблицы”.

**Пакет RDB$TIME_ZONE_UTIL**

Пакет RDB$TIME_ZONE_UTIL пакет содержит процедуры и функции для работы с часовым поясами.

Подробное описание пакета вы можете найти в секции RDB$TIME_ZONE_UTIL главы “Системные пакеты”.

3.4.9. Обновление базы данных часовых поясов

Часовые пояса меняются часто: конечно, когда это происходит, желательно как можно скорее обновить базу данных часовых поясов.

Firebird хранит значения WITH TIME ZONE, переведённые во время UTC. Предположим, что значение создано с помощью одной базы данных часового пояса, и более позднее обновление этой базы данных изменяет информацию в диапазоне нашего сохраненного значения. Когда это значение будет прочитано, оно будет возвращено как отличное от значения, которое было сохранено изначально.

Firebird использует IANA базу данных часовых поясов через библиотеку ICU. Библиотека ICU, представленная в комплекте Firebird (Windows) или установлена в операционной системе POSIX, иногда может иметь устаревшую базу данных часовых поясов.

Обновленную базу данных можно найти на этой странице в FirebirdSQL GitHub. Имя файла le.zip обозначает прямой порядок байтов и является необходимым файлом для большинства компьютерных архитектур (совместимых с Intel/AMD x86 или x64), в то время как be.zip обозначает архитектуры с прямым порядком байтов и необходим в основном для компьютерных архитектур RISC. Содержимое zip-файла должно быть извлечено в подкаталог /tzdata установки Firebird, перезаписывая существующие файлы *.res.

/tzdata — это каталог по умолчанию, в котором Firebird ищет базу данных часовых поясов. Его можно переопределить с помощью переменной среды ICU_TIMEZONE_FILES_DIR.

3.5. Символьные типы данных

В СУБД Firebird для работы с символьными данными есть типы фиксированной длины CHAR
и переменной длины VARCHAR. Максимальный размер текстовых данных, хранящийся в этих типах данных, составляет 32767 байт для CHAR и 32765 байт для VARCHAR. Максимальное количество символов, которое поместится в этот объём, зависит от используемого набора символов CHARACTER SET. Последовательность сортировки, задаваемая предложением COLLATE, не влияет на этот максимум, хотя может повлиять на максимальный размер любого индекса, который включает столбец.

В случае отсутствия явного указания набора символов при описании текстового объекта базы данных будет использоваться набор символов по умолчанию, заданный при создании базы данных. При отсутствии явного указания набора символов, а также отсутствия набора символов по умолчанию для базы данных, поле получает набор символов CHARACTER SET NONE.

3.5.1. Unicode

В настоящее время все современные средства разработки поддерживают Unicode. При возникновении необходимости использования восточноевропейских текстов в строковых полях базы данных или для более экзотических алфавитов, рекомендуется работать с набором символов UTF8. При этом следует иметь в виду, что на один символ в данное наборе приходится до 4 байт. Следовательно, максимальное количество символов в символьных полях составит 32765/4 = 8191.

При этом следует обратить внимание, что фактически значение параметра “bytes per character” зависит от диапазона, к которому принадлежит символ: английские буквы занимают 1 байт, русские буквы — 2 байта, остальные символы — могут занимать до 4-х байт.

Набор символов UTF8 поддерживает последнюю версию стандарта Unicode, до 4 байт на символ, поэтому для интернациональных баз рекомендуется использовать именно эту реализацию поддержки Unicode в Firebird.

3.5.2. Набор символов клиента

При работе со строками важно помнить о наборе символов клиентского соединения. В случае различия набора символов, при выдаче результата для строковых столбцов происходит автоматическая перекодировка как при передаче данных с клиента на сервер, так и в обратном направлении с сервера на клиента. То есть, совершенно нормальной является ситуация, когда база создана в кодировке WIN1251, а в настройках клиента в параметрах соединения стоит KOI8R или UTF8.

3.5.3. Специальные наборы символов

Набор символов NONE

Набор символов NONE относится к специальным наборам символов. Его можно охарактеризовать тем, что каждый байт является частью строки, но в системе хранится без указаний, к какому фактическому набору символов они относятся. Разбираться с такими данными должно клиентское приложение, на него возлагается ответственность в правильной трактовке символов из таких полей.
Набор символов OCTETS

Также к специальным наборам символов относится OCTETS. В этом случае данные рассматриваются как байты, которые могут в принципе не интерпретироваться как символы. OCTETS позволяет хранить бинарные данные и/или результаты работы некоторых функций Firebird. Правильное отображение данных пользователю, хранящихся в полях с CHARACTER SET OCTETS, также становится заботой клиентской стороны. При работе с подобными данными следует также помнить, что СУБД не контролирует их содержимое и возможно возникновение исключения при работе кода, когда идёт попытка отображения бинарных данных в желаемой кодировке.

3.5.4. Последовательность сортировки

Каждый набор символов имеет последовательность сортировки (сопоставления) по умолчанию (COLLATE), которая определяет порядок сопоставления. Обычно он обеспечивает упорядочивание на основе числового кода символов и базовое сопоставление символов верхнего и нижнего регистра. Если для строк требуется какое-то поведение, которое не обеспечивается последовательностью сортировки по умолчанию, и для этого набора символов поддерживается подходящая альтернативная сортировка, то в определении столбца можно указать предложение COLLATE collation.

Предложение COLLATE collation может применяться в других контекстах помимо определения столбца. Для операций сравнения больше/меньше его можно добавить в предложение WHERE оператора SELECT. Если вывод необходимо отсортировать в специальной алфавитной последовательности или без учета регистра и существует соответствующее сопоставление, то предложение COLLATE может быть использовано в предложении ORDER BY, когда строки сортируются по символьному полю, и в предложении GROUP BY в случае групповых операций.

Независимый от регистра поиск

Для независимого от регистра поиска можно воспользоваться функцией UPPER.

Для поиска без учета регистра вы можете воспользоваться функция UPPER для преобразования как аргумента поиска, так и искомых строк в верхний регистр перед попыткой сопоставления.

```sql
...
WHERE UPPER(name) = UPPER(:flt_name)
...
```

Для строк в наборе символов, для которых доступна сортировка без учета регистра, вы можете просто применить сопоставление, чтобы напрямую сравнить аргумент поиска и искомые строки. Например, при использовании набора символов WIN1251 вы можете использовать для этой цели сортировку PXW_CYRL не чувствительную к регистру символов.

```sql
...
WHERE FIRST_NAME COLLATE PXW_CYRL >= :FLT_NAME
...
```
См. также: CONTAINING.

Последовательности сортировки для UTF-8

Ниже приведена таблица возможных последовательностей сортировки для набора символов UTF8.

Table 10. Последовательности сортировки для UTF8

<table>
<thead>
<tr>
<th>COLLATION</th>
<th>Комментарии</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS_BASIC</td>
<td>Сортировка работает в соответствии с положением символа в таблице (бинарная).</td>
</tr>
<tr>
<td>UNICODE</td>
<td>Сортировка работает в соответствии с алгоритмом UCA (Unicode Collation Algorithm) (алфавитная).</td>
</tr>
<tr>
<td>UTF-8</td>
<td>По умолчанию используется двоичное сопоставление, идентичное UCS_BASIC, которое было добавлено для совместимости с SQL стандартом.</td>
</tr>
<tr>
<td>UNICODE_CI</td>
<td>Сортировка без учета регистра символов.</td>
</tr>
<tr>
<td>UNICODE_CI_AI</td>
<td>Сортировка без учета регистра и без учета диакритических знаков в алфавитном порядке.</td>
</tr>
</tbody>
</table>

Пример сортировки строк для набора символов UTF8 без учёта регистра символов и диакритических знаков.

ORDER BY NAME COLLATE UNICODE_CI_AI

3.5.5. Индексирование символьных типов

При построении индекса по строковым полям необходимо учитывать ограничение на длину ключа индекса. Максимальная используемая длина ключа индекса равна 1/4 размера страницы, то есть от 1024 (для страниц размером 4096) до 8192 байтов (для страницы размером 32768). Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. В таблице приведены данные для максимальной длины индексируемой строки (в символах) в зависимости от размера страницы и набора символов, её можно вычислить по следующей формуле:

\[ \text{max\_char\_length} = \text{FLOOR}(\frac{\text{page\_size}}{4} - 9) / N, \]

где N — число байтов на представление символа.

Table 11. Длина индексируемой строки и набор символов
В кодировках, нечувствительных к регистру ("_CI"), один символ в индексе будет занимать не 4, а 6 байт, поэтому максимальная длина ключа для страницы, например для страницы 4096 байт составит 169 символов.

Последовательность сортировки (COLLATE) тоже может повлиять на максимальную длину индексируемой строки. Полный список доступных наборов символов и нестандартных порядков сортировки доступен в приложении Наборы символов и порядки сортировки.

См. также

CREATE DATABASE, Порядок сортировки, SELECT, WHERE, GROUP BY, ORDER BY

3.5.6. BINARY

BINARY является типом данных с фиксированной длиной для хранения бинарных данных. Если переданное количество байт меньше объявленной длины, то значение будет дополнено нулями. В случае если не указана длина, то считается, что она равна единице.

Синтаксис

BINARY [(<length>)]

Этот тип является псевдонимом типа CHAR [(<length>)] CHARACTER SET OCTETS и обратно совместим с ним.

Данный тип хорошо подходит для хранения уникального идентификатора полученного с помощью функции GEN_UUID().

См. также:

CHAR, CHARACTER SET OCTETS.

3.5.7. CHAR

CHAR является типом данных фиксированной длины. Если введённое количество символом меньше объявленной длины, то поле дополнится концевыми пробелами. В общем случае символ заполнитель может и не являться пробелом, он зависит от набора символов, так например, для набора символов OCTETS — это ноль.
Полное название типа данных CHARACTER, но при работе нет необходимости использовать полные наименования; инструменты по работе с базой прекрасно понимают и короткие имена символьных типов данных.

**Синтаксис**

```
{CHAR | CHARACTER} [(length)]
[CHARACTER SET <charset>] [COLLATE <collate>]
```

В случае если не указана длина, то считается, что она равна единице.

Данный тип символьных данных можно использовать для хранения в справочниках кодов, длина которых стандартна и определённой "ширины". Примером такого может служить почтовый индекс в России – 6 символов.

**3.5.8. VARBINARY**

VARBINARY является типом для хранения бинарных данных переменной длины. Реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля.

Полное название BINARY VARYING.

**Синтаксис**

```
{VARBINARY  | BINARY VARYING} (<length>)
```

Этот тип является псевдонимом типа VARCHAR (<length>) CHARACTER SET OCTETS и обратно совместим с ним.

**Example 16. Использование типов BINARY и VARBINARY в PSQL**

```
DECLARE VARIABLE VAR1 VARBINARY(10);
```

**Example 17. Использование типов BINARY и VARBINARY при определении таблицы**

```
CREATE TABLE INFO (  
  GUID BINARY(16),  
  ENCRYPT_KEY VARBINARY(100),  
  ICON BINARY VARYING(32000));
```

См. также:

VARCHAR, CHARACTER SET OCTETS.
3.5.9. VARCHAR

VARCHAR является базовым строковым типом для хранения текстов переменной длины, поэтому реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля.

Все символы, которые передаются с клиентского приложения в базу данных, считаются как значимые, включая начальные и конечные пробельные символы.

Полное название CHARACTER VARYING. Имеется и сокращённый вариант записи CHAR VARYING.

Синтаксис

```sql
{VARCHAR | {CHAR | CHARACTER} VARYING} (length)
[CHARACTER SET <charset>] [COLLATE <collate>]
```

3.5.10. NCHAR

Представляет собой символьный тип данных фиксированной длины с предопределённым набором символов ISO8859_1.

Синтаксис

```sql
{NCHAR | NATIONAL {CHAR | CHARACTER}} [((length))]
```

Синонимом является написание NATIONAL CHAR.

Аналогичный тип данных доступен для строкового типа переменной длины: NATIONAL CHARACTER VARYING.

3.6. Логический тип данных

В Firebird 3.0 был введён полноценный логический тип данных.

3.6.1. BOOLEAN

SQL-2008 совместимый тип данных BOOLEAN (8 бит) включает различные значения истиности TRUE и FALSE. Если не установлено ограничение NOT NULL, то тип данных BOOLEAN поддерживает также значение истиности UNKNOWN как NULL значение. Спецификация не делает различия между значением NULL этого типа и значением истиности UNKNOWN, которое является результатом SQL предиката, поискового условия или выражения логического типа. Эти значения взаимозаменяемы и обозначают одно и то же.

Как и в других языках программирования, значения типа BOOLEAN могут быть проверены в неявных значениях истиности. Например, field1 OR field2 или NOT field1 являются допустимыми выражениями.
Оператор IS

Предикаты могут использовать оператор Логический IS [NOT] для сопоставления. Например, field1 IS FALSE, или field1 IS NOT TRUE.

- Операторы эквивалентности ("=", "!=", "<>" и др.) допустимы во всех сравнениях.

Примеры BOOLEAN

**INSERT и SELECT**

```sql
CREATE TABLE TBOOL (ID INT, BVAL BOOLEAN);

COMMIT;

INSERT INTO TBOOL VALUES (1, TRUE);
INSERT INTO TBOOL VALUES (2, 2 = 4);
INSERT INTO TBOOL VALUES (3, NULL = 1);
COMMIT;

SELECT * FROM TBOOL

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;true&gt;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&lt;false&gt;</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&lt;null&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```

Проверка TRUE значения

```sql
SELECT * FROM TBOOL WHERE BVAL

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;true&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```

Проверка FALSE значения

```sql
SELECT * FROM TBOOL WHERE BVAL IS FALSE

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>&lt;false&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```
Проверка UNKNOWN значения

```sql
SELECT * FROM TBOOL WHERE BVAL IS UNKNOWN
```

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

Логические выражения в SELECT списке

```sql
SELECT ID, BVAL, BVAL AND ID < 2 FROM TBOOL
```

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;true&gt; &lt;true&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;false&gt; &lt;false&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;null&gt; &lt;false&gt;</td>
</tr>
</tbody>
</table>

PSQL объявления с начальным значением

```sql
DECLARE VARIABLE VAR1 BOOLEAN = TRUE;
```

Сравнения с UNKNOWN

```sql
-- Допустимый синтаксис, но как и сравнение
-- с NULL, никогда не вернёт ни одной записи
SELECT * FROM TBOOL WHERE BVAL = UNKNOWN
SELECT * FROM TBOOL WHERE BVAL <> UNKNOWN
```

Использование Boolean с другими типами данных

Хотя BOOLEAN по своей сути не может быть преобразован в какой-либо другой тип данных, начиная с версии 3.0.1 строки 'true' и 'false' (без учета регистра) будут неявно приводиться к BOOLEAN в выражениях значений, например

```sql
if (true > 'false') then ...
```

'false' преобразуется в BOOLEAN. Любая попытка использовать логические операторы AND, NOT, OR и IS потерпят неудачу. Например, NOT 'False' приведёт к ошибке.

А BOOLEAN может быть явно преобразован в строку и из нее с помощью CAST. Значение UNKNOWN не доступен при преобразовании к строке.
3.7. Бинарные типы данных

3.7.1. BLOB

BLOB (Binary Large Objects, большие двоичные объекты) представляют собой сложные структуры, предназначенные для хранения текстовых и двоичных данных неопределенной длины, зачастую очень большого объёма.

Синтаксис

```
BLOB [SUB_TYPE <subtype>]  
[SEGMENT SIZE <seg_length>]  
[CHARACTER SET <charset>]  
[COLLATE <collation name>]
```

Сокращённый синтаксис:

```
BLOB (<seg_length>)  
BLOB (<seg_length>, <subtype>)  
BLOB (, <subtype>)
```

Размер сегмента:

Указание размера сегмента BLOB является некоторым атавизмом, оно идёт с тех времён, когда приложения для работы с данными BLOB писались на C (Embedded SQL) при помощи GPRE. В настоящий момент размер сегмента при работе с данными BLOB определяется клиентской частью, причём размер сегмента может превышать размер страницы данных.

Подтипы BLOB

Подтип BLOB отражает природу данных, записанную в столбце. Firebird предоставляет два предопределённых подтипа для сохранения пользовательских данных:

Подтип 0 (BINARY)

Если подтип не указан, то данные считаются не типизированными и значение подтипа принимается равным 0. Псевдоним подтипа 0 — BINARY. Этот подтип указывает, что данные имеют форму бинарного файла или потока (изображение, звук, видео, файлы текстового процессора, PDF и т.д.).

Подтип 1 (TEXT)

Подтип 1 имеет псевдоним TEXT, который может быть использован вместо указания номера подтипа. Например, BLOB SUBTYPE TEXT. Это специализированный подтип, который
используется для хранения текстовых данных большого объёма. Для текстового подтипа BLOB может быть указан набор символов и порядок сортировки COLLATE, аналогично символьному полю.

Пользовательские подтипы
Кроме того, существует возможность добавления пользовательских подтипов данных, для них зарезервирован интервал от -1 до -32768. Пользовательские подтипы с положительными числами не поддерживаются, поскольку Firebird использует числа больше 2 для внутренних подтипов метаданных.

Особенности BLOB

Размер
Максимальный размер поля BLOB ограничен 4Гб и не зависит от варианта сервера, 32 битный или 64 битный (во внутренних структурах, связанных с BLOB присутствуют 4-х байтные счётчики). Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб.

Операторы и выражения
Текстовые BLOB любой длины и с любым набором символов (включая multi-byte) могут быть использованы практически с любыми встроенными функциями и операторами.

Полностью поддерживаются следующие операторы:

= (присвоение)
=, <>, <, <=, >, >= (сравнение)
|| (конкатенация)
BETWEEN, IS [NOT] DISTINCT FROM,
IN, ANY | SOME,
ALL

Частично поддерживаются следующие операторы:

• возникает ошибка, в случае если второй аргумент больше или равен 32 Кб

STARTING [WITH], LIKE,
CONTAINING

• Предложения агрегирования работают не с содержимым самого поля, а с идентификатором BLOB ID. Помимо этого, есть некоторые странности:

SELECT DISTINCT
ORDER BY
GROUP BY

объединяет одинаковые строки, если они находятся рядом, но не делает этого, если они располагаются вдали друг от друга
Хранение BLOB

• По умолчанию, для каждого BLOB создаётся обычная запись, хранящаяся на какой-то выделенной для этого странице данных (data page). Если весь BLOB на эту страницу поместится, его называют BLOB уровня 0. Номер этой специальной записи хранится в записи таблицы и занимает 8 байт.

• Если BLOB не помещается на одну страницу данных (data page), то его содержимое размещается на отдельных страницах, целиком выделенных для него (blob page), а в записи о BLOB помещают номера этих страниц. Это BLOB уровня 1.

• Если массив номеров страниц с данными BLOB не помещается на страницу данных (data page), то его (массив) размещают на отдельных страницах (blob page), а в запись о BLOB помещают уже номера этих страниц. Это BLOB уровня 2.

• Уровни выше 2 не поддерживаются.

См. также:
FILTER, DECLARE FILTER.

3.7.2. Массивы

Поддержка массивов в СУБД Firebird является расширением традиционной реляционной модели. Поддержка в СУБД такого инструмента позволяет проще решать некоторые задачи по обработке однотипных данных. Массивы в Firebird реализованы на базе полей типа BLOB. Массивы могут быть одномерными и многомерными.

```sql
CREATE TABLE SAMPLE_ARR ( 
    ID INTEGER NOT NULL PRIMARY KEY, 
    ARR_INT INTEGER [4]);
```

Так будет создана таблица с полем типа массива из четырёх целых. Индексы данного массива от 1 до 4.

Указание явных границ для измерений

По умолчанию размеры начинаются с 1. Для определения верхней и нижней границы значений индекса следует воспользоваться следующим синтаксисом:

```
[<lower>:<upper>]
```

Добавление дополнительных измерений

Добавление новой размерности в синтаксисе идёт через запятую. Пример создания таблицы с массивом размерности два, в котором нижняя граница значений начинается с нуля:

```sql
```
Использование массивов

СУБД не предоставляет большого набора инструментов для работы с содержимым массивов. База данных employee.fdb, которая находится в дистрибутиве Firebird, содержит пример хранимой процедуры, показывающей возможности работы с массивами. Ниже приведён её текст:

```
CREATE OR ALTER PROCEDURE SHOW_LANGS ( 
    CODE VARCHAR(5),
    GRADE SMALLINT,
    CTY VARCHAR(15))
RETURNS ( 
    LANGUAGES VARCHAR(15)) 
AS 
    DECLARE VARIABLE I INTEGER;
BEGIN
    I = 1;
    WHILE (I <= 5) DO
        BEGIN 
            SELECT LANGUAGE_REQ[>:I]
            FROM JOB
            WHERE (JOB_CODE = :CODE)
                AND (JOB_GRADE = :GRADE)
                AND (JOB_COUNTRY = :CTY)
                AND (LANGUAGE_REQ IS NOT NULL))
            INTO :LANGUAGES;

            IF (:LANGUAGES = '') THEN
                /* PRINTS 'NULL' INSTEAD OF BLANKS */
                LANGUAGES = 'NULL';
                I = I +1;
                SUSPEND;
            END
        END
    END
```

Если приведённых выше возможностей достаточно для ваших задач, то вы вполне можете применять массивы для своих проектов. В настоящее время совершенствования механизмов обработки массивов средствами СУБД не производится.

3.8. Специальные типы данных
3.8.1. Тип данных SQL_NULL

Данный тип данных содержит не данные, а только состояние: NULL или NOT NULL. Также, этот тип данных не может быть использован при объявлении полей таблицы, переменных PSQL, используемых в описании параметров. Этот тип данных добавлен для улучшения поддержки нетипизированных параметров в предикате IS NULL. Такая проблема возникает при использовании “отключаемых фильтров” при написании запросов следующего типа:

```sql
WHERE col1 = :param1 OR :param1 IS NULL
```

После обработки, на уровне API запрос будет выглядеть как

```sql
WHERE col1 = ? OR ? IS NULL
```

В данном случае получается ситуация, когда разработчик при написании SQL запроса рассматривает `:param1` как одну переменную, которую использует два раза, а на уровне API запрос содержит два отдельных и независимых параметра. Добавление к этому, сервер не может определить тип второго параметра, поскольку он идёт в паре с IS NULL.

Именно для решения проблемы “? IS NULL” и был добавлен этот специальный тип данных SQL_NULL.

После введения данного специального типа данных при передаче запроса и его параметров на сервер будет работать такая схема: приложение передаёт параметризованные запросы на сервер в виде “?” . Это делает невозможным слияние пары “одинаковых” параметров в один. Так, например, для двух фильтров (двух именованных параметров) необходимо передать четыре позиционных параметра (далее предполагается, что читатель имеет некоторое знакомство с Firebird API):

```sql
SELECT  
    SH.SIZE, SH.COLOUR, SH.PRICE 
FROM SHIRTS SH 
WHERE (SH.SIZE = ? OR ? IS NULL) 
    AND (SH.COLOUR = ? OR ? IS NULL)
```

После выполнения `isc_dsql_describe_bind()` sqltype 2-го и 4-го параметров устанавливается в SQL_NULL. Как уже говорилось выше, сервер Firebird не имеет никакой информации об их связи с 1-м и 3-м параметрами— это полностью прерогатива программиста. Как только значения для 1-го и 3-го параметров были установлены (или заданы как NULL) и запрос подготовлен, каждая пара XSQLVARs должна быть заполнена следующим образом:

**Пользователь задал параметры**

- Первый параметр (сравнение значений): установка *sqldata в переданное значение и *sqlind в 0 (для NOT NULL);
- Второй параметр (проверка на NULL): установка *sqldata в null (указатель null, а не SQL NULL) и *sqlind в 0 (для NOT NULL).
Пользователь оставил поле пустым

- Оба параметра (проверка на NULL): установка *sqldata в null (указатель null, а не SQL NULL) и *sqlind в -1 (индикация NULL).

Другими словами: значение параметра сравнения всегда устанавливается как обычно. SQL_NULL параметр устанавливается также, за исключением случая, когда sqldata передаётся как null.

3.9. Преобразование типов данных

При написании выражения или при задании, например, условий сравнения, нужно стараться использовать совместимые типы данных. В случае необходимости использования смешанных данных различных типов, желательно первоначально выполнить преобразования типов, а уже потом выполнять операции.

При рассмотрении вопроса преобразования типов в Firebird большое внимание стоит уделить тому, в каком диалекте база данных.

3.9.1. Явное преобразование типов данных

В тех случаях, когда требуется выполнить явное преобразование одного типа в другой, используют функцию CAST.

Синтаксис

```
CAST (<expression> | NULL AS <data_type>)
```

<data_type> ::=  
  <datatype>  
  | [TYPE OF] domain  
  | TYPE OF COLUMN relname.colname

<datatype> ::=  
  <scalar_datatype> | <blob_datatype> | <array_datatype>

<scalar_datatype> ::=  
  См. <<fblangref-datatypes-syntax-scalar, Синтаксис скалярных типов данных>>

<blob_datatype> ::=  
  См. <<fblangref-datatypes-syntax-blob, Синтаксис типа данных BLOB>>

(array_datatype) ::=  
  См. <<fblangref-datatypes-syntax-array, Синтаксис массивов>>

Преобразование к домену

При преобразованиях к домену учитываются объявленные для него ограничения, например, NOT NULL или описанные в CHECK и если <expression> не пройдёт проверку, то преобразование не удалится. В случае если дополнительно указывается TYPE OF (преобразование к базовому типу), при преобразованиях игнорируются любые ограничения домена. При использовании TYPE OF с типом [VAR]CHAR набор символов и сортировка сохраняются.
Преобразование к типу столбца

При преобразовании к типу столбца допускается использовать указание столбца таблицы или представления. Используется только сам тип столбца; в случае строковых типов это также включает набор символов, но не сортировку. Ограничения и значения по умолчанию исходного столбца не применяются.

```sql
CREATE TABLE TTT ( 
  S VARCHAR(40) 
  CHARACTER SET UTF8 COLLATE UNICODE_CI_AI); 
COMMIT; 

/* У меня много друзей (шведский)*/
SELECT 
  CAST ('Jag har manga vanner' AS TYPE OF COLUMN TTT.S) 
FROM RDB$DATABASE;
```

Допустимые преобразования для функции `CAST`

<table>
<thead>
<tr>
<th>Из типа</th>
<th>В тип</th>
</tr>
</thead>
<tbody>
<tr>
<td>Числовые типы</td>
<td>Числовые типы, [VAR]CHAR, BLOB</td>
</tr>
<tr>
<td>[VAR]CHAR, BLOB</td>
<td>[VAR]CHAR, BLOB, BOOLEAN, Числовые типы, DATE, TIME, TIMESTAMP</td>
</tr>
<tr>
<td>DATE, TIME</td>
<td>[VAR]CHAR, BLOB, TIMESTAMP</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>[VAR]CHAR, BLOB, TIME, DATE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>[VAR]CHAR, BLOB</td>
</tr>
</tbody>
</table>

Для преобразования строковых типов данных в тип BOOLEAN необходимо чтобы строковый аргумент был одним из предопределённых литералов логического типа ('true' или 'false').

При преобразовании типов следует помнить о возможной частичной потери данных, например, при преобразовании типа данных TIMESTAMP в DATE.

Преобразование строк в дату и время

Для преобразования строковых типов данных в типы DATE, TIME или TIMESTAMP необходимо чтобы строковый аргумент был либо одним из предопределённых литералов даты и времени, либо строковое представление даты в одном из разрешёных форматов.

```
<date_literal> ::= 
  [YYYY<p>]MM<p>DD | 
  MM<p>DD[<p>YYYY] | 
  DD<p>MM[<p>YYYY] |
```

Chapter 3. Типы данных
Даты и время

\[
\text{MM}<\text{DD}[<\text{p}>\text{YY}] | \\
\text{DD}<\text{MM}[<\text{p}>\text{YY}]
\]

\[
<\text{time\_literal}> := \text{HH}:\text:mm[:\text{SS}\.\text{NNNN}]]
\]

\[
<\text{datetime\_literal}> ::= <\text{date\_literal}> <\text{time\_literal}>
\]

\[
<\text{time\_zone}> ::= <\text{time\_zone\_region} | [+/-] <\text{hour\_displacement} > [ : <\text{minute\_displacement}>] \]
\[
<\text{p}> ::= \text{whitespace} | . | , | - | /
\]

**Table 13. Описание формата даты и времени**

<table>
<thead>
<tr>
<th>Аргумент</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>datetime_literal</code></td>
<td>Литерал даты-времени.</td>
</tr>
<tr>
<td><code>date_literal</code></td>
<td>Литерал даты.</td>
</tr>
<tr>
<td><code>time_literal</code></td>
<td>Литерал времени.</td>
</tr>
<tr>
<td><code>YYYY</code></td>
<td>Год из четырёх цифр.</td>
</tr>
<tr>
<td><code>YY</code></td>
<td>Последние две цифры года (00-99).</td>
</tr>
<tr>
<td><code>MM</code></td>
<td>Месяц. Может содержать 1 или 2 цифры (1-12 или 01-12). В качестве месяца допустимо также указывать трёх буквенноное сокращение или полное наименование месяца на английском языке, регистр не имеет значения.</td>
</tr>
<tr>
<td><code>DD</code></td>
<td>День. Может содержать 1 или 2 цифры (1-31 или 01-31).</td>
</tr>
<tr>
<td><code>HH</code></td>
<td>Час. Может содержать 1 или 2 цифры (0-23 или 00-23).</td>
</tr>
<tr>
<td><code>mm</code></td>
<td>Минуты. Может содержать 1 или 2 цифры (0-59 или 00-59).</td>
</tr>
<tr>
<td><code>SS</code></td>
<td>Секунды. Может содержать 1 или 2 цифры (0-59 или 00-59).</td>
</tr>
<tr>
<td><code>NNNN</code></td>
<td>Десятьтичные доли секунды. Может содержать от 1 до 4 цифр (0-9999).</td>
</tr>
<tr>
<td><code>p</code></td>
<td>Разделитель, любой из разрешённых символов, лидирующие и завершающие пробелы игнорируются</td>
</tr>
<tr>
<td><code>time\_zone\_region</code></td>
<td>Один из часовых поясов связанных с регионом</td>
</tr>
<tr>
<td><code>hour\_displacement</code></td>
<td>Смещение времени для часов относительно GMT</td>
</tr>
<tr>
<td><code>minute\_displacement</code></td>
<td>Смещение времени для минут относительно GMT</td>
</tr>
</tbody>
</table>

**Table 14. Литералы с предопределёнными значениями даты и времени**

<table>
<thead>
<tr>
<th>Литерал</th>
<th>Значение</th>
<th>Тип данных для диалекта 1</th>
<th>Тип данных для диалекта 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>'NOW'</td>
<td>Текущая дата и время</td>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Литерал</td>
<td>Значение</td>
<td>Тип данных для диалекта 1</td>
<td>Тип данных для диалекта 3</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>--------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>'TODAY'</td>
<td>Текущая дата</td>
<td>TIMESTAMP (с нулевым временем)</td>
<td>DATE (только дата)</td>
</tr>
<tr>
<td>'TOMORROW'</td>
<td>Завтрашняя дата</td>
<td>TIMESTAMP (с нулевым временем)</td>
<td>DATE (только дата)</td>
</tr>
<tr>
<td>'YESTERDAY'</td>
<td>Вчерашняя дата</td>
<td>TIMESTAMP (с нулевым временем)</td>
<td>DATE (только дата)</td>
</tr>
</tbody>
</table>

Правила:

- В формате Год-Месяц-День, год обязательно должен содержать 4 цифры;
- Для дат в формате с завершающим годом, если в качестве разделителя дат используется точка “.”, то дата интерпретируется в форме День-Месяц-Год, для остальных разделителей она интерпретируется в форме Месяц-День-Год;
- Если год не указан, то в качестве года берётся текущий год;
- Если указаны только две цифры года, то для получения столетия Firebird использует алгоритм скользящего окна. Задача заключается в интерпретации двухсимвольного значения года как ближайшего к текущему году в интервале предшествующих и последующих 50 лет;
- Если не указан один из элементов времени, то оно принимается равным 0.

При использовании преобразования строковых литералов в тип даты/времени с помощью функции CAST() вычисление значения всегда происходит в момент выполнения.

При преобразовании строковых литералов с предопределёнными значениями даты и времени в тип TIMESTAMP точность составляет 3 знака после запятой (миллисекунды).

Настоятельно рекомендуем в литералах дат использовать только формы с полным указанием года в виде 4 цифр во избежание путаницы.

**Example 18. Преобразование строк в дату и время:**

```sql
SELECT
  CAST('04.12.2014' AS DATE) AS d1, -- DD.MM.YYYY
  CAST('12-04-2014' AS DATE) AS d2, -- MM-DD-YYYY
  CAST('12/04/2014' AS DATE) AS d3, -- MM/DD/YYYY
  CAST('04.12.14' AS DATE) AS d4, -- DD.MM.YY
  -- DD.MM в качестве года берётся текущий
  CAST('04.12' AS DATE) AS d5,
  -- MM/DD в качестве года берётся текущий
  CAST('12/4' AS DATE) AS d6,
  CAST('2014/12/04' AS DATE) AS d7, -- YYYY/MM/DD
  CAST('2014.12.04' AS DATE) AS d8, -- YYYY.MM/DD
  CAST('2014-12-04' AS DATE) AS d9, -- YYYY-MM-DD
  CAST('11:37' AS TIME) AS t1, -- HH:mm
```
Поскольку \texttt{CAST('NOW' AS TIMESTAMP)} всегда возвращает актуальные значения даты и времени, то она может использоваться для измерения временных интервалов и скорости выполнения кода в процедурах, триггерах и блоках кода PSQL.

\textit{Example 19. Использование \texttt{CAST('NOW' AS TIMESTAMP)} измерения длительности выполнения кода}

```sql
EXECUTE BLOCK
RETURNS (ms BIGINT)
AS
DECLARE VARIABLE t1 TIME;
DECLARE VARIABLE n BIGINT;
BEGIN
```
t1 = CAST('now' AS TIMESTAMP);

/* Долгая операция */
SELECT COUNT(*) FROM rdb$types, rdb$types, rdb$types INTO n;

/*========*/
ms = DATEDIFF(MILLISECOND FROM t1 TO CAST('now' AS TIMESTAMP));
SUSPEND;
END

См. также:
Литералы даты и времени, CAST().

3.9.2. Неявное преобразование типов данных

В 3-м диалекте невозможно неявное преобразование данных, здесь требуется указывать функцию CAST для явной трансляции одного типа в другой. Однако это не относится к операции конкатенации, при которой все другие типы данных будут неявно преобразованы к символьному.

При использовании 1-го диалекта во многих выражениях выполняется неявное преобразование одних типов в другой без применения функции CAST. Например, в выражении отбора в диалекте 1 можно записать:

```
WHERE DOC_DATE < '31.08.2014'
```

и преобразование строки в дату произойдет неявно.

В 1-м диалекте можно смешивать целые данные и числовые строки, строки неявно преобразуются в целое, если это будет возможно, например:

```
2 + '1'
```

корректно выполнится.

В 3-м диалекте подобное выражение вызовет ошибку, в нем потребуется запись следующего вида:

```
2 + CAST('1' AS SMALLINT)
```

Неявное преобразование типов при конкатенации

При конкатенации множества элементов разных типов, все не строковые данные будут неявно преобразованы к строке, если это возможно.
3.10. Пользовательские типы данных — домены

Домены в СУБД Firebird реализуют широко известный по многим языкам программирования инструмент “типы данных, определённые пользователем”. Когда несколько таблиц в базе данных содержат поля с характеристиками одинаковыми или практически одинаковыми, то есть целесообразность сделать домен, в котором описать набор свойств поля и использовать такой набор свойств, описанный один раз, в нескольких объектах базы данных. Домены могут использоваться помимо описания полей таблиц в представлениях (VIEW) и при объявлении входных и выходных параметров, а также при объявлении переменных в коде PSQL.

3.10.1. Атрибуты домена

Определение домена содержит обязательные и необязательные атрибуты. К обязательному атрибуту относится тип данных. К необязательным относятся:

- значение по умолчанию;
- возможности использования NULL для домена;
- ограничения CHECK для данных домена;
- набор символов (для символьных типов данных и BLOB полей);
- порядок сортировки (для символьных типов данных).

Example 21. Создание домена

```sql
CREATE DOMAIN BOOL3 AS SMALLINT
CHECK (VALUE IS NULL OR VALUE IN (0, 1));
```

См. также:

Явное преобразование типов данных, где описаны отличия работы механизма преобразования данных при указании доменов для опций TYPE OF и TYPE OF COLUMN.
3.10.2. Переопределение свойств доменов

При описании таблиц базы данных некоторые свойства столбцов, базирующихся на доменах, могут быть переопределены. Возможности переопределения атрибутов столбцов на базе доменов приведены в таблице.

**Table 15. Возможности переопределения атрибутов столбцов на базе доменов**

<table>
<thead>
<tr>
<th>Атрибут</th>
<th>Переопределяется?</th>
<th>Примечания</th>
</tr>
</thead>
<tbody>
<tr>
<td>тип данных</td>
<td>нет</td>
<td></td>
</tr>
<tr>
<td>значение по умолчанию</td>
<td>да</td>
<td></td>
</tr>
<tr>
<td>текстовый набор символов</td>
<td>да</td>
<td>также может использоваться, чтобы восстановить для столбца значения по умолчанию для базы данных</td>
</tr>
<tr>
<td>текстовый порядок сортировки</td>
<td>да</td>
<td></td>
</tr>
<tr>
<td>условия проверки CHECK</td>
<td>нет</td>
<td>для добавления в проверку новых условий, можно использовать в операторах CREATE и ALTER на уровне таблицы соответствующие предложения CHECK.</td>
</tr>
<tr>
<td>NOT NULL</td>
<td>нет</td>
<td>во многих случаях лучше оставить при описании домена возможность значения NULL, а контроль его допустимости осуществлять в описании полей на уровне таблицы.</td>
</tr>
</tbody>
</table>

3.10.3. Создание доменов

Создание домена производится оператором CREATE DOMAIN.

**Краткий синтаксис:**

```
CREATE DOMAIN name [AS] <type>
[DEFAULT {<literal> | NULL | <context_var>}] [NOT NULL] [CHECK (<condition>)] [COLLATE collation];
```

*См. также:*  
CREATE DOMAIN.
### 3.10.4. Изменение доменов

Для редактирования свойств домена используют оператор ALTER DOMAIN языка определения данных (DDL).

При редактировании домена можно:

- переименовать домен;
- изменить тип данных;
- удалить текущее значение по умолчанию;
- установить новое значение по умолчанию;
- установить ограничение NOT NULL;
- удалить ограничение NOT NULL;
- удалить текущее ограничение CHECK;
- добавить новое ограничение CHECK.

**Краткий синтаксис:**

```
ALTER DOMAIN name
    [[TO new_name]]
    [[SET DEFAULT {<literal> | NULL | <context_var>} | DROP DEFAULT]]
    [[SET | DROP] NOT NULL]
    [[ADD [CONSTRAINT] CHECK (<dom_condition>) | DROP CONSTRAINT]]
    [[TYPE <datatype>]];
```

**Example 22. Изменение значения по умолчанию для домена**

```
ALTER DOMAIN STORE_GRP SET DEFAULT -1;
```

При изменении доменов следует учитывать и его зависимости: имеются ли столбцы таблиц; находятся ли в коде PSQL объявления переменных, входных и/или выходных параметров с типом этого домена. При поспешном редактировании без внимательной проверки можно сделать данный код неработоспособным!

При смене в домене типа данных не допустимы преобразования, которые могут привести к потере данных. Также, например, при преобразовании VARCHAR в INTEGER проверьте, все ли данные, что используют данных домен, смогут пройти преобразование.

**См. также:**

ALTER DOMAIN.
3.10.5. Удаление доменов

Оператор DROP DOMAIN удаляет из базы данных домена при условии, что он не используется в каком либо из объектов базы данных.

Синтаксис:

```
DROP DOMAIN name;
```

Example 23. Удаление домена

```
DROP DOMAIN Test_Domain;
```

См. также:

DROP DOMAIN.

3.11. Синтаксис объявления типа данных

В этом разделе описывается синтаксис объявления типов данных. Объявление типа данных чаще всего встречается в операторах DDL, но также в CAST и EXECUTE BLOCK.

На приведенный ниже синтаксис есть ссылки из других частей этого руководства.

3.11.1. Синтаксис скалярных типов данных

Скалярные типы данных — это простые типы данных, которые содержат одно значение. Синтаксис типов BLOB рассматривается отдельно в секции Синтаксис типов данных BLOB.

Синтаксис:

```
<domain_or_non_array_type> ::= 
    <scalar_datatype> 
    | <blob_datatype> 
    | [TYPE OF] domain 
    | TYPE OF COLUMN rel.col

<scalar_datatype> ::= 
    {SMALLINT | INT[GER] | BIGINT | INT128} 
    | BOOLEAN 
    | {FLOAT | REAL | DOUBLE PRECISION} 
    | [LONG] FLOAT [{(binary-precision)}] 
    | DECFLOAT[{{16 | 34}}] 
    | DATE 
    | {TIME | TIMESTAMP} [{WITH | WITHOUT} TIME ZONE] 
    | {DECIMAL | NUMERIC} [{(precision [, scale])}]
    | {VARCHAR | {CHAR | CHARACTER} VARYING} (length) 
    | [CHARACTER SET charset]
```
Chapter 3. Типы данных

| {CHAR | CHARACTER} [(length)] [CHARACTER SET charset] |
| {NCHAR | NATIONAL {CHARACTER | CHAR}} VARYING (length) |
| {NCHAR | NATIONAL {CHARACTER | CHAR}} [(length)] |

Table 16. Параметры декларации скалярных типов

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>Имя домена (только не домены типа массив).</td>
</tr>
<tr>
<td>rel</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>col</td>
<td>Имя столбца таблицы или представления (только столбцы не типа массив).</td>
</tr>
<tr>
<td>binary-precision</td>
<td>Двоичная точность. От 1 до 53 бит. По умолчанию 24 бита.</td>
</tr>
<tr>
<td>precision</td>
<td>Десятичная точность. От 1 до 38 десятичных цифр.</td>
</tr>
<tr>
<td>scale</td>
<td>Масштаб или количество знаков после запятой. От 0 до 38. Оно должно быть меньше или равно точности.</td>
</tr>
<tr>
<td>length</td>
<td>Максимальная длина строки в символах.</td>
</tr>
<tr>
<td>charset</td>
<td>Набор символов.</td>
</tr>
<tr>
<td>domain_or_non_array_type</td>
<td>Типы, не являющиеся массивами, которые можно использовать в коде PSQL и операторе CAST.</td>
</tr>
</tbody>
</table>

Использование доменов в объявлениях

Имя домена может быть указано как тип параметра PSQL или локальной переменной. Параметр или переменная наследует все атрибуты домена. Если для параметра или переменной указано значение по умолчанию, оно переопределяет значение по умолчанию, указанное в определении домена.

Если предложение TYPE OF добавлено перед именем домена, то используется только тип данных домена: любые другие атрибуты домена — ограничение NOT NULL, ограничение CHECK, значение по умолчанию — не проверяются и не используются. Однако, если домен имеет текстовый тип, всегда используются его набор символов и последовательность сортировки.

Использование TYPE OF COLUMN в объявлениях

Входные и выходные параметры или локальные переменные также могут быть объявлены с использованием типа данных столбцов в существующих таблицах и представлениях. Для этого используется предложение TYPE OF COLUMN, в котором в качестве аргумента указывается rel.col.

Когда используется TYPE OF COLUMN, параметр или переменная наследует только тип данных и — для строковых типов — набор символов и последовательность сортировки. Ограничения и значение столбца по умолчанию игнорируются.
3.11.2. Синтаксис типов данных BLOB

Типы данных BLOB содержат данные в двоичном, символьном или пользовательском формате неопределенного размера. Для получения дополнительной информации см. BLOB.

**Синтаксис типа BLOB**

```plaintext
<blob_datatype> ::= 
    BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
    | BLOB [(seglen [, subtype_num])]
```

**Table 17. Параметры декларации типа BLOB**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>Набор символов (игнорируется для всех подтипов кроме 1 (TEXT)).</td>
</tr>
<tr>
<td>subtype_num</td>
<td>Номер подтипа BLOB.</td>
</tr>
<tr>
<td>subtype_name</td>
<td>Мнемоническое имя подтипа BLOB; это может быть TEXT, BINARY или одно из (других) стандартных или настраиваемых имен, определенных в RDB$TYPES для RDB$FIELD_NAME = 'RDB$FIELD_SUB_TYPE'.</td>
</tr>
<tr>
<td>seglen</td>
<td>Размер сегмента не может быть больше 65535, по умолчанию — 80, если не указан. Размер сегмента может быть переопределен клиентом и в большинстве случаев не учитывается.</td>
</tr>
</tbody>
</table>

3.11.3. Синтаксис массивов

Тип данных массив содержит несколько скалярных значений в одном или многомерном массиве. Для получения дополнительной информации см. Тип массив.

**Синтаксис массивов**

```plaintext
<array_datatype> ::= 
    {SMALLINT | INT[TEGER] | BIGINT | INT128} <array_dim>
    | BOOLEAN <array_dim>
    | {FLOAT | REAL | DOUBLE PRECISION} <array_dim>
    | [LONG] FLOAT [binary-precision] <array_dim>
    | DECFLOAT[({16 | 34})] <array_dim>
    | DATE <array_dim>
    | {TIME | TIMESTAMP} [{WITH | WITHOUT} TIME ZONE] <array_dim>
    | {DECIMAL | NUMERIC} [{(precision [, scale])} <array_dim>
    | {VARCHAR | {CHAR | CHARACTER} VARYING} (length) <array_dim>
    | {CHAR | CHARACTER} [{(length)} <array_dim> [CHARACTER SET charset]
    | {NCHAR | NATIONAL {CHARACTER | CHAR}} VARYING (length) <array_dim>
    | {NCHAR | NATIONAL {CHARACTER | CHAR}} [{(length)} <array_dim>

<array_dim> ::= '[' '[m:]n [,,[m:]n ... ] ']
```
Table 18. Параметры декларации массивов

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary-precision</td>
<td>Двоичная точность. От 1 до 64 бит.</td>
</tr>
<tr>
<td>precision</td>
<td>Десятичная точность. От 1 до 38 десятичных цифр.</td>
</tr>
<tr>
<td>scale</td>
<td>Масштаб или количество знаков после запятой. От 0 до 38. Оно должно быть меньше или равно точности.</td>
</tr>
<tr>
<td>length</td>
<td>Максимальная длина строки в символах.</td>
</tr>
<tr>
<td>charset</td>
<td>Набор символов.</td>
</tr>
<tr>
<td>m, n</td>
<td>Целые числа, определяющие диапазон индекса измерения массива.</td>
</tr>
</tbody>
</table>
Chapter 4. Общие элементы языка

В этой главе рассматриваются элементы, которые являются общими для всех реализаций языка SQL — выражения, которые используются для извлечения и работают на утверждениях о данных, и предикатов, которые проверяют истинность этих утверждений.

4.1. Выражения

Выражения SQL представляют формальные методы для вычисления, преобразования и сравнения значений. Выражения SQL могут включать в себя столбцы таблиц, переменные, константы, литералы, различные операторы и предикаты, а также другие выражения. Полный список допустимых символов (tokens) в выражениях описан ниже.

Описание элементов языка

Имя столбца

Идентификаторы столбцов из указанных таблиц, используемые в вычислениях, или сравнениях, или в качестве условия поиска. Столбец типа массив не может быть элементом выражения, если только он не проверяется на IS [NOT] NULL.

Элементы массива

В выражении может содержаться ссылка на элемент массива, т.е. <array_name>[s], где s — индекс элемента в массиве.

Арифметические операторы

Символы +, -, *, / используемые для вычисления значений.

Оператор конкатенации

Оператор || ("две вертикальные линии") используется для соединения символьных строк.

Логические операторы

Зарезервированные слова NOT, AND и OR используются при комбинировании простых условий поиска для создания сложных утверждений.

Операторы сравнения

Символы =, !=, <>, <, <=, >, >=, !<, ^<, !>, < and ^>.

Предикаты сравнения

LIKE, STARTING WITH, CONTAINING, SIMILAR TO, BETWEEN, IS [NOT] NULL, IS [NOT] {TRUE | FALSE | UNKNOWN}, и IS [NOT] DISTINCT FROM

Предикаты существования

Предикаты, используемые для проверки существования значений в наборе. Предикат IN может быть использован как с наборами констант, так и со скалярными подзапросами. Предикаты EXISTS, SINGULAR, ALL ANY, SOME могут быть использованы только с подзапросами.
Константы и литералы

Числа, заключённые в апострофы строковые литералы, логические значения TRUE, FALSE и UNKNOWN, псевдозначение NULL.

Литералы дат и времени

Выражения, подобные строковым литералам, заключённые в апострофах, которые могут быть интерпретированы как значения даты, времени или даты-времени. Литералами дат могут быть строки из символов и чисел, такие как TIMESTAMP '25.12.2016 15:30:35', которые могут быть преобразованы в дату, время или дату с временем.

Мнемоники дат и времени

Строковый литерал с описанием желаемого значения даты и/или времени, которое можно привести к типу даты и/или времени. Для примера 'NOW', 'TODAY'.

Контекстные переменные

Встроенные контекстные переменные.

Локальные переменные

Локальные переменные, входные или выходные параметры PSQL модулей (хранимых процедур, триггеров, анонимных блоков SQL).

Позиционные параметры

В DSQL в качестве параметров запроса могут быть использованы только позиционные параметры. Позиционные параметры представляют собой знаки вопроса (?) внутри DSQL оператора. Доступ к таким параметрам осуществляется по его номеру (позиции в запросе относительно предыдущего позиционного параметра) поэтому они называются позиционными. Обычно компоненты доступа позволяют работать с именованными параметрами, которые они сами преобразовывают в позиционные.

Подзапросы

Оператор SELECT заключённый в круглые скобки, который возвращает одно единственное (скалярное) значение или множество значений (при использовании в предикатах существования).

Идентификаторы функций

Идентификаторы встроенных или внешних функций в функциональных выражениях.

Приведения типа

Выражение явного преобразования одного типа данных в другой с использованием CAST как CAST(<value> AS <datatype>).

Условные выражения

Выражение CASE и встроенные функции COALESCE, NULLIF.

Круглые скобки

Пара скобок ( ... ) используются для группировки выражений. Операции внутри скобок выполняются перед операциями вне скобок. При использовании вложенных скобок, сначала вычисляются значения самых внутренних выражений, а затем вычисления...
Предложение COLLATE
Предложение применяется к типам CHAR и VARCHAR, чтобы в указанной кодировке установить параметры сортировки, используемые при сравнении.

NEXT VALUE FOR sequence
Конструкция NEXT VALUE FOR позволяет получить следующее значение последовательности, то же самое делает встроенная функция GEN_ID().

Выражение AT
Выражение для изменения часового пояса даты и времени.

4.1.1. Литералы (константы)
Литерал или константа — это значение, подставляемое непосредственно в SQL оператор, которое не получено из выражения, параметра, ссылки на столбец или переменной.

Строковые литералы (константы)
Строковый литерал это последовательность символов, заключенных между парой апострофов (“одинарных кавычек”). Максимальная длина строковой константы составляет 65535 байт; максимальное количество символов будет определяться количеством байт, используемых для кодирования каждого символа.

- Двойные кавычки не должны (допускаются 1 диалектом) использоваться для квотирования строк. В SQL они предусмотрены для других целей.
- Если литерал апострофа требуется в строковой константе, то он может быть “экранирован” другим предшествующим апострофом. Например, 'Mother O’Reilly's home-made hooch'. Или используйте альтернативные кавычки для литералов: q'{Mother O’Reilly's home-made hooch}'.
- Необходимо быть осторожным с длиной строки, если значение должно быть записано в столбец типа VARCHAR. Максимальная длина строки для типа VARCHAR составляет 32765 байт (32767 для типа CHAR). Если значение должно быть записано в столбец типа BLOB, то максимальная длина строкового литерала составляет 65535 байт.

Предполагается, что набор символов строковой константы совпадает с набором символов столбца предназначенного для её сохранения.

Строковые константы в шестнадцатеричной нотации
Начиная с Firebird 2.5 строковые константы могут быть записаны в шестнадцатеричной нотации, так называемые “двоичные строки”. Каждая пара шестнадцатеричных цифр определяет один байт в строке. Строки введённые таким образом будут иметь кодировку OCTETS по умолчанию, но вводный синтаксис (introducer syntax) может быть использован для принудительной интерпретации строки в другом наборе символов.
Синтаксис:

{x|X}'<hexstring>'

<hexstring> ::= an even number of <hexdigit>

<hexdigit> ::= 0..9 | A..F | a..f

Example 24. Примеры:

```
SELECT x'4E657276656E' FROM rdb$database
-- returns 4E657276656E, a 6-byte 'binary' string

SELECT _ascii x'4E657276656E' FROM rdb$database
-- returns 'Nerven' (same string, now interpreted as ASCII text)

SELECT _iso8859_1 x'53E46765' FROM rdb$database
-- returns 'Säge' (4 chars, 4 bytes)

SELECT _utf8 x'53C3A46765' FROM rdb$database
-- returns 'Säge' (4 chars, 5 bytes)
```

Как будут отображены двоичные строки зависит от интерфейса клиента. Например, утилита isql использует заглавные буквы A-F, в то время как FlameRobin буквы в нижнем регистре. Другие могут использовать другие правила конвертирования, например отображать пробелы между парами байт: '4E 65 72 76 65 6E'.

Шестнадцатеричная нотация позволяет вставить любой байт (включая 00) в любой позиции в строке.

Альтернативы для апострофов в строковых литералах

Вместо двойного (экранного) апострофа вы можете использовать другой символ или пару символов.

Ключевое слово q или Q предшествующее строке в кавычках сообщает парсеру, что некоторые левые и правые пары одинаковых символов являются разделителями для встроенного строкового литерала.

Синтаксис:

```
<alternate string literal> ::= { q | Q } <quote> <alternate start char> [ { <char> }... ] <alternate end char> <quote>
```
Правила использования

Когда <alternate start char> является одним из символов '(', '{', '[' или '<', то <alternate end char> должен быть использован в паре с соответствующим “партнёром”, а именно ')', '}', ']' или '>'. В других случаях <alternate end char> совпадает с <alternate start char>.

Внутри строки, т.е. <char> элементах, одиночные (не экранированные) кавычки могут быть использованы. Каждая кавычка будет частью результирующей строки.

**Example 25. Использование альтернативных апострофов в строковых литералах**

```sql
-- result: abc{def}ghi
SELECT Q'abc{def}ghi' FROM rdb$database;

-- result: That's a string
SELECT Q'!That's a string!' FROM rdb$database;
```

**Example 26. Динамическая сборка запроса использующего строковые литералы.**

```sql
EXECUTE BLOCK
RETURNS ( RDB$TRIGGER_NAME CHAR(64) )
AS
    DECLARE VARIABLE S VARCHAR(8191);
BEGIN
    S = 'SELECT RDB$TRIGGER_NAME FROM RDB$TRIGGERS WHERE RDB$RELATION_NAME IN ';
    S = S || Q'! ('SALES_ORDER', 'SALES_ORDER_LINE')!';
    FOR EXECUTE STATEMENT :S
    INTO :RDB$TRIGGER_NAME
    DO
        SUSPEND;
END
```

Вводный синтаксис для строковых литералов

При необходимости, строковому литералу может предшествовать имя набор символов, который начинается с префикса подчеркивания “_”. Это известно как вводный синтаксис (Introducer syntax). Его цель заключается в информировании Firebird о том, как интерпретировать и хранить входящую строку.
Числовые константы

Числовая константа — это любое правильное число в одной из поддерживаемых нотаций:

- В SQL, для чисел в стандартной десятичной записи, десятичная точка всегда представлена символом точки и тысячи не разделены. Включение запятых, пробелов, и т.д. вызовет ошибки.
- Экспоненциальная запись, например число 0.0000234 может быть записано как 2.34e-5.
- Шестнадцатеричная запись (см. ниже) чисел поддерживается начиная с Firebird 2.5.

Далее показаны форматы числовых литералов и их типы. Где <d> - десятичная цифра, <h> - шестнадцатеричная цифра.

**Table 19. Формат числовых констант**

<table>
<thead>
<tr>
<th>Формат</th>
<th>Тип</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;d&gt;[&lt;d&gt; …]</td>
<td>INTEGER, BIGINT, INT128 или DECFLOAT(34) (зависит от того, подходит ли значение типу). DECFLOAT(34) используется для значений, которые не помещаются в INT128.</td>
</tr>
<tr>
<td>0{x</td>
<td>X} &lt;h&gt;[&lt;h&gt; …]</td>
</tr>
<tr>
<td>&lt;d&gt;[&lt;d&gt; …].[&lt;d&gt; …]</td>
<td>NUMERIC(18, n), NUMERIC(38, n) или DECFLOAT(34) где n зависит от количества цифр после десятичной точки, а точность от общего количества цифр. Для обеспечения обратной совместимости некоторые значения из 19 цифр отображаются на NUMERIC(18, n). DECFLOAT(34) используется, когда немасштабированное значение не помещается в INT128.</td>
</tr>
<tr>
<td>&lt;d&gt;[&lt;d&gt; …][. [&lt;d&gt; …]] E &lt;d&gt;[&lt;d&gt; …]</td>
<td>DOUBLE PRECISION или DECFLOAT(34), где DECFLOAT используется, только если количество цифр 20 или больше, или абсолютный показатель степени 309 или больше.</td>
</tr>
</tbody>
</table>
Шестнадцатеричная нотация чисел

Константы целочисленных типов можно указать в шестнадцатеричном формате. Начиная с Firebird 4.0.1 числа состоящие из 17-32 шестнадцатеричных цифр будут интерпретированы как INT128.

Синтаксис:

```
{x|X}<hexdigits>
<hexdigits> ::= 1-32 of <hexdigit>
<hexdigit> ::= 0..9 | A..F | a..f
```

Table 20. Константы целочисленных типов в шестнадцатеричном формате

<table>
<thead>
<tr>
<th>Количество шестнадцатеричных цифр</th>
<th>Тип данных</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>INTEGER</td>
</tr>
<tr>
<td>9-16</td>
<td>BIGINT</td>
</tr>
<tr>
<td>17-32</td>
<td>INT128</td>
</tr>
</tbody>
</table>

Example 28. Шестнадцатеричные константы

```
SELECT 0x6FAA0D3 FROM rdb$database -- returns 117088467
SELECT 0x4F9 FROM rdb$database -- returns 1273
SELECT 0x6E44F9A8 FROM rdb$database -- returns 1850014120
SELECT 0x9E44F9A8 FROM rdb$database -- returns -1639646808 (an INTEGER)
SELECT 0x09E44F9A8 FROM rdb$database -- returns 2655320488 (a BIGINT)
SELECT 0x28ED678A4C987 FROM rdb$database -- returns 720001751632263
SELECT 0xFFFFFFFFFFFFFFFF FROM rdb$database -- returns -1
```

Диапазон значений шестнадцатеричных чисел

- Шестнадцатеричные числа в диапазоне 0 .. 7FFF FFFF являются положительными INTEGER числа со значениями 0 .. 2147483647. Для того чтобы интерпретировать константу как BIGINT число, необходимо дополнить достаточно количество нулей слева. Это изменит тип, но не значение.

- Числа в диапазоне 8000 0000 .. FFFF FFFF требуют особого внимания:

  - При записи восемью шестнадцатеричный числом, такие как 0x9E44F9A8, интерпретируется как 32-битное целое. Поскольку крайний левый (знаковый) бит установлен, то такие числа будут находиться в отрицательном диапазоне -2147483648 .. -1.

  - Числа предварённые одним или несколькими нулями, такие как 0x09E44F9A8, будут интерпретированы как 64-разрядный BIGINT в диапазоне значений 0000 0000 8000 0000 .. 0000 0000 FFFF FFFF. В этом случае знаковый бит не установлен, поэтому они
отображаются в положительном диапазоне 2147483648 .. 4294967295 десятичных чисел.

Таким образом, только в этом диапазоне числа, предварённые совершенно незначимым нулём, имеют кардинально разные значения. Это необходимо знать.

• Шестнадцатеричные числа в диапазоне 1 0000 0000 .. 7FFF FFFF FFFF FFFF являются положительными BIGINT числами.

• Шестнадцатеричные числа в диапазоне 8000 0000 0000 0000 .. FFFF FFFF FFFF FFFF являются отрицательными BIGINT числами.

• Числа с типом SMALLINT не могут быть записаны в шестнадцатеричном виде, строго говоря, так как даже 0x1 оценивается как INTEGER. Тем не менее, если вы записываете положительное целое число в пределах 16-разрядного диапазона от 0x0000 (десятичный ноль) до 0x7FFF (десятничное 32767), то оно будет преобразовано в SMALLINT прозрачно.

Вы можете записать отрицательное SMALLINT число в шестнадцатеричном виде используя 4-байтное шестнадцатеричное число в диапазоне от 0xFFFF8000 (десятничное -32768) до 0xFFFFFFFF (десятничное -1).

**Логические литералы**

Логический литерал может быть одним из следующих значений: TRUE, FALSE или UNKNOWN.

### 4.1.2. Операторы SQL

SQL операторы включают в себя операторы для сравнения, вычисления, оценки и конкатенации значений.

**Приоритет операторов**

Приоритет определяет порядок, в котором операторы и получаемые с помощью них значения вычисляются в выражении.

Все операторы разбиты на 4 типа. Каждый тип оператора имеет свой приоритет. Чем выше приоритет типа оператора, тем раньше он будет вычислен. Внутри одного типа операторы имеют собственный приоритет, который также определяет порядок их вычисления в выражении. Операторы с одинаковым приоритетом вычисляются слева направо. Для изменения порядка вычислений операции могут быть сгруппированы с помощью круглых скобок.

**Table 21. Приоритеты типов операторов**

<table>
<thead>
<tr>
<th>Тип оператора</th>
<th>Приоритет</th>
<th>Пояснение</th>
</tr>
</thead>
<tbody>
<tr>
<td>Конкатенация</td>
<td>1</td>
<td>Строки объединяются до выполнения любых других операций.</td>
</tr>
<tr>
<td>Арифметический</td>
<td>2</td>
<td>Арифметические операции выполняются после конкатенации строк, но перед выполнением операторов сравнения и логических операций.</td>
</tr>
</tbody>
</table>
### Оператор конкатенации

Оператор конкатенации `||` соединяет две символьные строки и создаёт одну строку. Символьные стоки могут быть константами или значениями, полученными из столбцов или других выражений.

**Example 29. Оператор конкатенации**

```sql
SELECT LAST_NAME || ' ' || FIRST_NAME AS FULL_NAME
FROM EMPLOYEE
```

### Арифметические операторы

**Table 22. Приоритет арифметических операторов**

<table>
<thead>
<tr>
<th>Оператор</th>
<th>Назначение</th>
<th>Приоритет</th>
</tr>
</thead>
<tbody>
<tr>
<td>+signed_number</td>
<td>Унарный плюс</td>
<td>1</td>
</tr>
<tr>
<td>-signed_number</td>
<td>Унарный минус</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>Умножение</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>Деление</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>Сложение</td>
<td>3</td>
</tr>
<tr>
<td>-</td>
<td>Вычитание</td>
<td>3</td>
</tr>
</tbody>
</table>

**Example 30. Арифметические операторы**

```sql
UPDATE T
SET A = 4 + 1/(B-C)*D
```

### Операторы сравнения

**Table 23. Операторы сравнения**

<table>
<thead>
<tr>
<th>Оператор</th>
<th>Приоритет</th>
<th>Пояснение</th>
</tr>
</thead>
<tbody>
<tr>
<td>Сравнение</td>
<td>3</td>
<td>Операции сравнения вычисляются после конкатенации строк и выполнения арифметических операций, но до логических операций.</td>
</tr>
<tr>
<td>Логический</td>
<td>4</td>
<td>Логические операторы выполняются после всех других типов операторов.</td>
</tr>
<tr>
<td>Оператор</td>
<td>Назначение</td>
<td>Приоритет</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>IS</td>
<td>Проверяет, что выражение в левой части является псевдо значением NULL или соответствует логическому значению в правой части.</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>Равно, идентично</td>
<td>2</td>
</tr>
<tr>
<td>&lt;&gt;, !=, ^=</td>
<td>Не равно</td>
<td>2</td>
</tr>
<tr>
<td>&gt;</td>
<td>Больше</td>
<td>2</td>
</tr>
<tr>
<td>&lt;</td>
<td>Меньше</td>
<td>2</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Больше или равно</td>
<td>2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Меньше или равно</td>
<td>2</td>
</tr>
<tr>
<td>!&gt;, &lt;&lt;- ^=</td>
<td>Не больше</td>
<td>2</td>
</tr>
<tr>
<td>!&lt;, &lt;&lt;- ^=</td>
<td>Не меньше</td>
<td>2</td>
</tr>
</tbody>
</table>

В эту же группу входят предикаты сравнения IS DISTINCT FROM, BETWEEN, IN, LIKE, CONTAINING, SIMILAR TO и другие.

*Example 31. Использование оператора сравнения*

```
IF (SALARY > 1400) THEN
...
```

*См. также:*

Другие предикаты сравнения.

**Логические операторы**

*Table 24. Приоритет логических операторов*

<table>
<thead>
<tr>
<th>Оператор</th>
<th>Назначение</th>
<th>Приоритет</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>Отрицание условия поиска.</td>
<td>1</td>
</tr>
<tr>
<td>AND</td>
<td>Объединяет два предиката и более, каждый из которых должен быть истинным, чтобы истинным был и весь предикат.</td>
<td>2</td>
</tr>
<tr>
<td>OR</td>
<td>Объединяет два предиката и более, из которых должен быть истинным хотя бы один предикат, чтобы истинным был и весь предикат.</td>
<td>3</td>
</tr>
</tbody>
</table>

*Example 32. Использование логических операторов*

```
IF (A > B OR (A > C AND A > D) AND NOT (C = D)) THEN
```
4.1.3. AT

Доступно в DSQL, PSQL.

Синтаксис

```sql
<expr> AT {TIME ZONE <time zone string> | LOCAL}
```

```sql
<time zone string> ::= 
    '<time zone>'

<time zone> ::= 
    <time zone region> | 
    [+/-] <hour displacement> [: <minute displacement>]
```

Преобразует время или временную метку в указанный часовой пояс. Если используется ключевое слово LOCAL, то преобразование происходит в часовой пояс сессии.

**Example 33. Использование функции AT**

```sql
select time '12:00 GMT' at time zone '-03'
from rdb$database;

select current_timestamp at time zone 'America/Sao_Paulo'
from rdb$database;

select timestamp '2018-01-01 12:00 GMT' at local
from rdb$database;
```

4.1.4. NEXT VALUE FOR

Доступно в DSQL, PSQL.

Синтаксис

```sql
NEXT VALUE FOR sequence-name
```

Возвращает следующее значение в последовательности (SEQUENCE). SEQUENCE является SQL совместимым термином генератора в InterBase и Firebird. Оператор NEXT VALUE FOR полностью эквивалентен функции GEN_ID (sequence-name, 1) и является рекомендуемым синтаксисом.
NEXT VALUE FOR не поддерживает значение приращения, отличное от того, что было указано при создании последовательности в предложении INCREMENT [BY]. Если требуется другое значение шага, то используйте старую функцию GEN_ID.

Example 34. Использование NEXT VALUE FOR

```
NEW.CUST_ID = NEXT VALUE FOR CUSTSEQ;
```

См. также:
SEQUENCE (GENERATOR), GEN_ID.

4.1.5. Условные выражения

Условное выражение — это выражение, которое возвращает различные значения в зависимости от истинности некоторого условия или условий. В данном разделе описано лишь одно условное выражение CASE. Остальные условные выражения являются производными встроенными функциями и описаны в разделе Условные функции.

CASE

Доступно в
DSQL, ESQL.

Оператор CASE возвращает только одно значение из нескольких возможных. Есть два синтаксических варианта:

- Простой CASE, сравнимый с Pascal case или C switch;
- Поисковый CASE, который работает как серия операторов “if … else if … else if”.

Простой CASE

Синтаксис

```
CASE <test-expr>
    WHEN <expr> THEN <result>
    [WHEN <expr> THEN <result> ...]
    [ELSE <defaultresult>]
END
```

При использовании этого варианта test-expr сравнивается с первым expr, затем вторым expr и так далее, до тех пор, пока не будет найдено совпадение, и тогда возвращается соответствующий результат. Если совпадений не найдено, то возвращается defaultresult из ветви ELSE. Если нет совпадений, и ветвь ELSE отсутствует, то возвращается значение NULL.

Совпадение эквивалентно оператору “=”, то есть если test-expr имеет значение NULL, то он не
соответствует ни одному из $expr$, даже тем, которые имеют значение $NULL$.

Результаты необязательно должны быть литеральными значениями, они также могут быть именами полей, переменными, сложными выражениями или $NULL$.

*Example 35. Использование простого CASE*

```sql
SELECT
    NAME,
    AGE,
    CASE UPPER(SEX)
        WHEN 'M' THEN 'Male'
        WHEN 'F' THEN 'Female'
        ELSE 'Unknown'
    END AS SEXNAME,
    RELIGION
FROM PEOPLE
```

Сокращённый вид простого оператора CASE используется в функции $DECODE$.

*Поисковый CASE*

**Синтаксис**

```sql
CASE
    WHEN <bool_expr> THEN <result>
    [WHEN <bool_expr> THEN <result> …]
    [ELSE <defaultresult>]
END
```

Здесь $<bool_expr>$ выражение, которое даёт тройной логический результат: TRUE, FALSE или NULL. Первое выражение, возвращающее TRUE, определяет результат. Если нет выражений, возвращающих TRUE, то в качестве результата берётся $defaultresult$ из ветви ELSE. Если нет выражений, возвращающих TRUE, и ветвь ELSE отсутствует, результатом будет NULL.

Как и в простом операторе CASE, результаты не обязаны быть литературными значениями: они могут быть полями или именами переменных, сложными выражениями, или NULL.

*Example 36. Использование поискового CASE*

```sql
CANVOTE = CASE
    WHEN AGE >= 18 THEN 'Yes'
    WHEN AGE < 18 THEN 'No'
    ELSE 'Unsure'
END;
```
4.1.6. **NULL в выражениях**

NULL не является значением — это состояние, указывающее, что значение элемента неизвестно или не существует. Это не ноль, не пустота, не "пустая строка", и оно не ведёт себя как какое-то из этих значений.

При использовании NULL в числовых, строковых выражениях или в выражениях, содержащих дату/время, в результате вы всегда получите NULL. При использовании NULL в логических (булевых) выражениях результат будет зависеть от типа операции и других вовлечённых значений. При сравнении значения с NULL результат будет неопределенным (UNKNOWN).

![ retirees ] Неопределённый логический результат UNKNOWN тоже представлен псевдозначением NULL.

**Выражения возвращающие NULL**

Выражения в этом списке всегда возвратят NULL:

```
1 + 2 + 3 + NULL
'Home ' || 'sweet ' || NULL
MyField = NULL
MyField <> NULL
NULL = NULL
not (NULL)
```

Если вам трудно понять, почему, вспомните, что NULL — значит "неизвестно".

**NULL в логических выражениях**

Мы уже рассмотрели, что not (NULL) даёт в результате NULL. Для операторов AND и OR взаимодействие несколько сложнее:

```
NULL or false → NULL
NULL or true → true
NULL or NULL → NULL
NULL and false → false
NULL and true → NULL
NULL and NULL → NULL
```

**Example 37. NULL в логических выражениях**

```
(1 = NULL) or (1 <> 1) -- returns NULL
(1 = NULL) or FALSE  -- returns NULL
(1 = NULL) or (1 = 1) -- returns TRUE
(1 = NULL) or TRUE   -- returns TRUE
(1 = NULL) or (1 = NULL) -- returns NULL
```
4.2. Подзапросы

Подзапрос — это специальный вид выражения, которое фактически является запросом, встроенным в другой запрос. Подзапросы пишутся как обычные SELECT запросы, но должны быть заключены в круглые скобки. Выражения подзапроса используются следующими способами:

• Для задания выходного столбца в списке выбора SELECT;
• Для получения значений или условий для предикатов поиска (предложения WHERE, HAVING);
• Для создания набора данных, из которого включающий запрос может выбирать, как будто это обычная таблица или представление. Подобные подзапросы появляются в предложении FROM (производные таблицы) или в общем табличном выражении (CTE).

4.2.1. Коррелированные подзапросы

Подзапрос может быть коррелированным (соотнесённым). Запрос называется коррелированным, когда подзапрос и основной запрос взаимозависимы. Это означает, что для обработки каждой записи подзапроса, должна быть получена также запись из основного запроса, т.е. подзапрос всецело зависит от основного запроса.

Example 38. Коррелированный подзапрос

SELECT *
FROM Customers C
WHERE EXISTS
  (SELECT *
   FROM Orders O
   WHERE C.cnum = O.cnum
   AND O.adate = DATE '10.03.1990');

При использовании подзапросов для получения значений выходного столбца в списке выбора SELECT, подзапрос должен возвращать скалярный результат.

4.2.2. Подзапросы возвращающие скалярный результат

Подзапросы, используемые в предикатах поиска, кроме предикатов существования и
количественных предикатов, должны возвращать скалярный результат, то есть не более чем один столбец из одной отобранной строки или одно агрегированное значение, в противном случае, произойдет ошибка времени выполнения (“Multiple rows in a singleton select...”).

Несмотря на то, что Firebird сообщает о подлинной ошибке, сообщение может немного вводить в заблуждение. “singleton SELECT” — это запрос, который не должен возвращать более одной строки. Однако “singleton” и “scalar” не являются синонимами: не все одноэлементные SELECTS должны быть скалярными; а выборка по одному столбцу может возвращать несколько строк для предикатов существования и количественных предикатов.

**Example 39. Подзапрос в качестве выходного столбца в списке выбора**

```sql
SELECT
e.first_name,
e.last_name,
(SELECT
  sh.new_salary
FROM
  salary_history sh
WHERE
  sh.emp_no = e.emp_no
ORDER BY sh.change_date DESC ROWS 1) AS last_salary
FROM
employee e
```

**Example 40. Подзапрос в предложении WHERE для получения значения максимальной зарплаты сотрудника и фильтрации по нему**

```sql
SELECT
e.first_name,
e.last_name,
e.salary
FROM
employee e
WHERE
e.salary = (SELECT
  MAX(ie.salary)
FROM
  employee ie)
```
4.3. Предикаты

Предикат — это простое выражение, утверждающее некоторый факт, назовем его \( P \). Если \( P \) разрешается как \( \text{TRUE} \), он успешен. Если он принимает значение \( \text{FALSE} \) или \( \text{NULL} \) (UNKNOWN), он терпит неудачу. Однако здесь кроется ловушка: предположим, что предикат \( P \) возвращает \( \text{FALSE} \). В этом случае \( \text{NOT} \ (P) \) вернет \( \text{TRUE} \). С другой стороны, если \( P \) возвращает \( \text{NULL} \) (неизвестно), то \( \text{NOT} \ (P) \) также возвращает \( \text{NULL} \).

В SQL предикаты проверяют в ограничении CHECK, предложении WHERE, выражении CASE, условии соединения во фразе ON для предложений JOIN, а также в предложении HAVING. В PSQL операторы управления потоком выполнения проверяют предикаты в предложениях IF, WHILE и WHEN. Поскольку начиная с Firebird 3.0 введена поддержка логического типа, то предикат может встречаться в любом правильном выражении.

4.3.1. Утверждения

Проверяемые условия не всегда являются простыми предикатами. Они могут быть группой предикатов, каждый из которых при вычислении делает вклад в вычислении общей истинности. Такие сложные условия называются утверждениями. Утверждения могут состоять из одного или нескольких предикатов, связанных логическими операторами \( \text{AND} \), \( \text{OR} \) и \( \text{NOT} \). Для группировки предикатов и управления порядком вычислений можно использовать скобки.

Каждый из предикатов может содержать вложенные предикаты. Результат вычисления истинности утверждения получается в результате вычисления всех предикатов по направлению от внутренних к внешним. Каждый “уровень” вычисляется в порядке приоритета до тех пор, пока не будет получено значение истинности окончательного утверждения.

4.3.2. Предикаты сравнения

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения:

\[ =, >, <, \geq, \leq, \neq \]

(Полный список операторов сравнения см. Операторы сравнения).

Если в одной из частей (левой или правой) предиката сравнения встречается \( \text{NULL} \), то значение предиката будет неопределённым (UNKNOWN).

Example 41. Предикаты сравнения

Получить информацию о компьютерах, имеющих частоту процессора не менее 500 МГц и цену ниже $800

```sql
SELECT *
```
FROM Pc
WHERE speed >= 500 AND price < 800;

Получить информацию обо всех принтерах, которые являются матричными и стоят меньше $300

SELECT *
FROM Printer
WHERE type = 'matrix' AND price < 300;

Следующий запрос не вернёт ни одной записи, поскольку сравнение происходит с псевдо-значением NULL, даже если существуют принтеры с неуказанным типом.

SELECT *
FROM Printer
WHERE type = NULL AND price < 300;

Замечание о сравнении строк
При сравнении на равенство полей типов CHAR и VARCHAR завершающий пробелы игнорируются во всех случаях.

4.3.3. Другие предикаты сравнения
Другие предикаты сравнения состоят из ключевых слов.

BETWEEN
Доступно в
DSQL, PSQL, ESQL.

Синтаксис

<value> [NOT] BETWEEN <value_1> AND <value_2>

Предикат BETWEEN проверяет, попадает (или не попадает при использовании NOT) ли значение во включающий диапазон значений.

Операнды для предиката BETWEEN — это два аргумента совместимых типов. В отличие от некоторых других СУБД в Firebird предикат BETWEEN не является симметричным. Меньшее значение должно быть первым аргументом, иначе предикат BETWEEN всегда будет ложным. Поиск является включающим. Таким образом, предикат BETWEEN можно переписать следующим образом:

<value> >= <value_1> AND <value> <= <value_2>
При использовании предиката BETWEEN в поисковых условиях DML запросов, оптимизатор Firebird может использовать индекс по искомому столбцу, если таковой доступен.

**Example 42. Использование предиката BETWEEN**

```sql
SELECT * 
FROM EMPLOYEE
WHERE HIRE_DATE BETWEEN date '01.01.1992' AND CURRENT_DATE
```

**LIKE**

**Доступно в**

DSQL, PSQL, ESQIL.

**Синтаксис**

```sql
<match value> [NOT] LIKE <pattern>
[ESCAPE <escape character>]
```

- `<match value>` ::= выражение символьного типа
- `<pattern>` ::= шаблон поиска
- `<escape character>` ::= символ экранирования

Предикат LIKE сравнивает выражение символьного типа с шаблоном, определённым во втором выражении. Чувствительность к регистру или диакритическим знакам при сравнении определяется используемым параметром сортировки (COLLATION).

При использовании оператора LIKE во внимание принимаются все символы строки-шаблона. Это касается так же начальных и конечных пробелов. Если операция сравнения в запросе должна вернуть все строки, содержащие строки LIKE 'абв' (с символом пробела на конце), то строка, содержащая 'абв' (без пробела), не будет возвращена.

**Трафаретные символы**

В шаблоне, разрешается использование двух трафаретных символов:

- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении;
- символ подчёркивания (_), который можно применять вместо любого единичного символа в проверяемом значении.

Если проверяемое значение соответствует образцу с учётом трафаретных символов, то предикат истиен.

**Использование управляющего символа в предложении ESCAPE**

Если искомая строка содержит трафаретный символ, то следует задать управляющий символ в предложении ESCAPE. Этот управляющий символ должен использоваться в образце.
перед трафаретным символом, сообщая о том, что последний следует трактовать как обычный символ.

Примеры использования предиката LIKE

Example 43. Поиск строк начинающихся с заданной подстроки с использованием предиката LIKE

Поиск номеров отделов, названия которых начинаются со слова “Software”

```sql
SELECT DEPT_NO
FROM DEPT
WHERE DEPT_NAME LIKE 'Software%';
```

В данном запросе может быть использован индекс, если он построен на поле DEPT_NAME.

Оптимизация LIKE

В общем случае предикат LIKE не использует индекс. Однако если предикат принимает вид LIKE 'string%', то он будет преобразован в предикат STARTING WITH, который будет использовать индекс. Если вам необходимо выполнить поиск с начала строки, то вместо предиката LIKE рекомендуется использовать предикат STARTING WITH.

Example 44. Использование трафаретного символа “_” в предикате LIKE

Поиск сотрудников, имена которых состоят из 5 букв, начинающихся с букв “Sm” и заканчивающихся на “th”. В данном случае предикат будет истинен для имен “Smith” и “Smyth”.

```sql
SELECT first_name
FROM employee
WHERE first_name LIKE 'Sm_th'
```

Example 45. Поиск внутри строки с использованием предиката LIKE

Поиск всех заказчиков, в адресе которых содержится строка “Ростов”.

```sql
SELECT *
FROM CUSTOMER
WHERE ADDRESS LIKE '%Ростов%'
```
Если вам необходимо выполнить поиск внутри строки, то вместо предиката `LIKE` рекомендуется использовать предикат `CONTAINING`.

**Example 46. Использование управляющего символа в предложении ESCAPE с предикатом `LIKE`**

Поиск таблиц, содержащих в именах знак подчёркивания. В данном случае в качестве управляющего символа задан символ “#”.

```
SELECT RDB$RELATION_NAME
FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME LIKE '%#_%' ESCAPE '#'
```

**См. также:**

`STARTING WITH, CONTAINING, SIMILAR TO.`

**STARTING WITH**

Доступно в

DSQL, PSQL, ESQL.

**Синтаксис**

```
<value> [NOT] STARTING WITH <start-value>
```

Предикат `STARTING WITH` ищет строку, которая начинается с символов в его аргументе `start-value`. Чувствительность к регистру и ударению в `STARTING WITH` зависит от сортировки (COLLATION) первого аргумента `value`.

При использовании предиката `STARTING WITH` в поисковых условиях DML запросов, оптимизатор Firebird может использовать индекс по искомому столбцу, если он определён.

**Example 47. Использование предиката `STARTING WITH`**

Поиск сотрудников, фамилии которых начинаются с “Jo”.

```
SELECT LAST_NAME, FIRST_NAME
FROM EMPLOYEE
WHERE LAST_NAME STARTING WITH 'Jo'
```

**См. также:**

`LIKE`. 
CONTAINING

Доступно в
DSQL, PSQL, ESQL.

Синтаксис

<value> [NOT] CONTAINING <substring>

Оператор CONTAINING ищет строку или тип, подобный строке, отыскивая последовательность символов, которая соответствует его аргументу. Он может быть использован для алфавитно-цифрового (подобного строковому) поиска в числах и датах. Поиск CONTAINING не чувствителен к регистру. Тем не менее, если используется сортировка чувствительная к акцентам, то поиск будет чувствителен к акцентам.

При использовании оператора CONTAINING во внимание принимаются все символы строки. Это касается так же начальных и конечных пробелов. Если операция сравнения в запросе должна вернуть все строки, содержащие строки CONTAINING 'абв ' (с символом пробела на конце), то строка, содержащая 'абв' (без пробела), не будет возвращена.

При использовании предиката CONTAINING в поисковых условиях DML запросов, оптимизатор Firebird не может использовать индекс по искомому столбцу.

**Example 48. Поиск подстроки с использованием предиката CONTAINING**

Поиск проектов в именах, которых присутствует подстрока “Map”:

```sql
SELECT *
FROM PROJECT
WHERE PROJ_NAME CONTAINING 'map';
```

В данном случае будет возвращены две строки с именами “AutoMap” и “MapBrowser port”.

**Example 49. Поиск внутри даты с использованием предиката CONTAINING**

Поиск записей об изменении зарплат с датой содержащей число 84 (в данном случае изменения, которые произошли в 1984 году):

```sql
SELECT *
FROM SALARY_HISTORY
WHERE CHANGE_DATE CONTAINING 84;
```

См. также:

LIKE.
СИМИЛЯР ТО

Доступно в DSQ\L, PSQL.

Синтаксис

\texttt{string-expression \[NOT\] SIMILAR TO <pattern> \[ESCAPE <escape-char>\]}

\texttt{<pattern> ::= регулярное выражение SQL}
\texttt{<escape-char> ::= символ экранирования}

Оператор \texttt{SIMILAR TO} проверяет соответствие строки с шаблоном регулярного выражения SQL. В отличие от некоторых других языков для успешного выполнения шаблон должен соответствовать всей строке—соответствие подстроки недостаточно. Если один из операндов имеет значение \texttt{NULL}, то и результат будет \texttt{NULL}. В противном случае результат является \texttt{TRUE} или \texttt{FALSE}.

Синтаксис регулярных выражений SQL

Следующий синтаксис определяет формат регулярного выражения SQL. Это полное и корректное его определение. Он является весьма формальным и довольно длинным и, вероятно, озадачивает тех, кто не имеет опыта работы с регулярными выражениями. Не стесняйтесь пропустить его и начать читать следующий раздел, Создание регулярных выражений, использующий подход от простого к сложному.

\texttt{<regular expression> ::= <regular term> [\'|' <regular term> ...]}

\texttt{<regular term> ::= <regular factor> ...}

\texttt{<regular factor> ::= <regular primary> [<quantifier>]}

\texttt{<quantifier> ::= ? | * | + | '{' <m> [,<n>] '}'}

\texttt{<m>, <n> ::= целые положительные числа, если присутствуют оба числа, то <m> \leq <n>}

\texttt{<regular primary> ::=}
\texttt{\hspace{1em} <character> | <character class> | %}
\texttt{\hspace{1em} | \((<regular expression>)\)}

\texttt{<character> ::= <escaped character> | <non-escaped character>}

\texttt{<escaped character> ::=}
\texttt{\hspace{1em} <escape-char> <special character> | <escape-char> <escape-char>}

\texttt{<special character> ::= любой из символов [\{\]()\^-\*%_?{}\

\texttt{<non-escaped character> ::=}
\texttt{\hspace{1em} любой символ за исключением <special character>}
\texttt{\hspace{1em} и не эквивалентный <escape-char> (если задан)
Создание регулярных выражений

В этом разделе представлены элементы и правила построения регулярных выражений SQL.

Символы

В регулярных выражениях большинство символов представляет сами себя, за исключением специальных символов (special character):

\[ ] ( ) | ^ - + * % _ \{ \}

... и управляющих символов (escaped character), если они заданы.

Регулярному выражению, не содержащему специальных или управляющих символов, соответствует только полностью идентичные строки (в зависимости от используемой сортировки). То есть это функционирует точно так же, как оператор “=":

'Apple' SIMILAR TO 'Apple' -- TRUE
'Apples' SIMILAR TO 'Apple' -- FALSE
'Apple' SIMILAR TO 'Apples' -- FALSE
'APPLE' SIMILAR TO 'Apple' -- в зависимости от сортировки

Шаблоны

Известным SQL шаблонам ‘_’ и ‘%’ соответствует любой единственный символ и строка любой длины, соответственно:

'Birne' SIMILAR TO 'B_rne' -- TRUE
'Birne' SIMILAR TO 'B_ne' -- FALSE
'Birne' SIMILAR TO 'B%ne' -- TRUE
'Birne' SIMILAR TO 'Bir%ne%' -- TRUE
'Birne' SIMILAR TO 'Birr%ne' -- FALSE
Обратите внимание, что шаблон ‘%’ также соответствует пустой строке.

**Классы символов**

Набор символов, заключённый в квадратные скобки определяют класс символов. Символ в строке соответствует классу в шаблоне, если символ является элементом класса:

```plaintext
'Citroen' SIMILAR TO 'Cit[arju]oen' -- TRUE
'Citroen' SIMILAR TO 'Ci[tr]oen' -- FALSE
'Citroen' SIMILAR TO 'Ci[tr][tr]oen' -- TRUE
```

Как видно из второй строки классу только соответствует единственный символ, а не их последовательность.

Два символа, соединённые дефисом, в определении класса определяют диапазон. Диапазон для активного сопоставления включает в себя эти два конечных символа и все символы, находящиеся между ними. Диапазоны могут быть помещены в любом месте в определении класса без специальных разделителей, чтобы сохранить в классе и другие символы.

```plaintext
'Datte' SIMILAR TO 'Dat[q-u]e' -- TRUE
'Datte' SIMILAR TO 'Dat[abq-uy]e' -- TRUE
'Datte' SIMILAR TO 'Dat[bcg-km-pwz]e' -- FALSE
```

**Предопределённые классы символов**

Следующие предопределенные классы символов также могут использоваться в определении класса:

- `[:ALPHA:]`
  Латинские буквы a...z и A...Z. Этот класс также включает символы с диакритическими знаками при нечувствительных к акцентам сортировках.

- `[:DIGIT:]`
  Десятичные цифры 0...9.

- `[:ALNUM:]`
  Объединение `[:ALPHA:]` и `[:DIGIT:]`.

- `[:UPPER:]`
  Прописные (в верхнем регистре) латинские буквы A...Z. Также включает в себя символы в нижнем регистре при нечувствительных к регистру сортировках и символы с диакритическими знаками при нечувствительных к акцентам сортировках.

- `[:LOWER:]`
  Строчные (в нижнем регистре) латинские буквы a...z. Также включает в себя символы в верхнем регистре при нечувствительных к регистру сортировках и символы с диакритическими знаками при нечувствительных к акцентам сортировках.
Символ пробела (ASCII 32).

Горизонтальная табуляция (ASCII 9), перевод строки (ASCII 10), вертикальная табуляция (ASCII 11), разрыв страницы (ASCII 12), возврат каретки (ASCII 13) и пробел (ASCII 32).

Включение в оператор SIMILAR TO предопределённого класса имеет тот же эффект, как и включение всех его элементов. Использование предопределённых классов допускается только в пределах определения класса. Если вам нужно сопоставление только с предопределённым классом и ничего больше, то поместите дополнительную пару скобок вокруг него.

```
'Erdbeere' SIMILAR TO 'Erd[:ALNUM:][a]eere' -- TRUE
'Erdbeere' SIMILAR TO 'Erd[:DIGIT:][a]eere' -- FALSE
'Erdbeere' SIMILAR TO 'Erd[a[:SPACE:][b]eere' -- TRUE
'Erdbeere' SIMILAR TO '[[:ALPHA:]]' -- FALSE
'E' SIMILAR TO '[[:ALPHA:]]' -- TRUE
```

Если определение класса запускается со знаком вставки (^), то все, что следует за ним, исключается из класса. Все остальные символы проверяются.

```
'Framboise' SIMILAR TO 'Fra[^ck-p]boise' -- FALSE
'Framboise' SIMILAR TO 'Fr[^a][^a]boise' -- FALSE
'Framboise' SIMILAR TO 'Fra[^[:DIGIT:]]boise' -- TRUE
```

Если знак вставки (^) находится не в начале последовательности, то класс включает в себя все символы до него и исключает символы после него.

```
'Grapefruit' SIMILAR TO 'Grap[a-m^[f-i]fruit' -- TRUE
'Grapefruit' SIMILAR TO 'Grap[abc^[xyz]fruit' -- FALSE
'Grapefruit' SIMILAR TO 'Grap[abc^[de]fruit' -- FALSE
'3' SIMILAR TO '[[:DIGIT:]^4-8' -- TRUE
'6' SIMILAR TO '[[:DIGIT:]^4-8' -- FALSE
```

Наконец, уже упомянутый подстановочный знак `_` является собственным классом символов, соответствуя любому единительному символу.

**Кванторы**

Вопросительный знак (`?`) сразу после символа или класса указывает на то, что для соответствия предыдущий элемент должен встретиться 0 или 1 раз:

```
'Hallon' SIMILAR TO 'Hal?on' -- FALSE
'Hallon' SIMILAR TO 'Hal?lon' -- TRUE
```
Звёздочка (**) сразу после символа или класса указывает на то, что для соответствия предыдущий элемент должен встретиться 0 или более раз:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Icaque'</td>
<td>'Ica*que'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Icaque'</td>
<td>'Icar*que'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Icaque'</td>
<td>'I[a-c]*que'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Icaque'</td>
<td>'*'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Icaque'</td>
<td>'[:ALPHA:]*'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Icaque'</td>
<td>'Ica[xyz]*e'</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Знак плюс (+) сразу после символа или класса указывает на то, что для соответствия предыдущий элемент должен встретиться 1 или более раз:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Jujube'</td>
<td>'Ju_'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Jujube'</td>
<td>'Ju+jube'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Jujube'</td>
<td>'Jujuber+'</td>
<td>FALSE</td>
</tr>
<tr>
<td>'Jujube'</td>
<td>'J[jux]+be'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Jujube'</td>
<td>'J[:DIGIT:]+ujube'</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Если символ или класс сопровождаются числом, заключённым в фигурные скобки ('{ и '}), то для соответствия необходимо повторение элемента точно это число раз:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Kiwi'</td>
<td>'Ki{2}wi'</td>
<td>FALSE</td>
</tr>
<tr>
<td>'Kiwi'</td>
<td>'K[ipw]{2}i'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Kiwi'</td>
<td>'K[ipw]{2}'</td>
<td>FALSE</td>
</tr>
<tr>
<td>'Kiwi'</td>
<td>'K[ipw]{3}'</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Если число сопровождается запятой (','), то для соответствия необходимо повторение элемента как минимум это число раз:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Limone'</td>
<td>'Li{2,}mone'</td>
<td>FALSE</td>
</tr>
<tr>
<td>'Limone'</td>
<td>'Li{1,}mone'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Limone'</td>
<td>'Li[nezom]{2,}'</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Если фигурные скобки содержат два числа (m и n), разделённые запятой, и второе число больше первого, то для соответствия элемент должен быть повторен, как минимум, m раз и не больше n раз:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Mandarijn'</td>
<td>'M[a-p]{2,5}rijn'</td>
<td>TRUE</td>
</tr>
<tr>
<td>'Mandarijn'</td>
<td>'M[a-p]{2,3}rijn'</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Кванторы ‘?’, ‘*’ и ‘+’ являются сокращением для \{0,1\}, \{0,\} и \{1,\}, соответственно.

**Термин ИЛИ**

В условиях регулярных выражений можно использовать оператор ИЛИ ‘|’. Соответствие произошло, если строка параметра соответствует, по крайней мере, одному из условий:

\['Mandarijn' SIMILAR TO 'M[a-p]{2,3}arijn' -- TRUE\]

\['Nektarin' SIMILAR TO 'Nektarin' -- FALSE\]
\['Nektarin' SIMILAR TO 'Nektarin|Persika' -- TRUE\]
\['Nektarin' SIMILAR TO 'M_+|N_+|P_+' -- TRUE\]

**Подвыражения**

Одна или более частей регулярного выражения могут быть сгруппированы в подвыражения (также называемые подмасками). Для этого их нужно заключить в круглые скобки (‘(‘ и ‘)’):

\['Orange' SIMILAR TO 'O(ra|ri|ro)nge' -- TRUE\]
\['Orange' SIMILAR TO 'O(r[a-e])+nge' -- TRUE\]
\['Orange' SIMILAR TO 'O(ra){2,4}nge' -- FALSE\]
\['Orange' SIMILAR TO 'O(ra|in|ng|rong)?e' -- TRUE\]

**Экранирование специальных символов**

Для исключения из процесса сопоставления специальных символов (которые часто встречаются в регулярных выражениях) их надо экранировать. Специальных символов экранирования по умолчанию нет — их при необходимости определяет пользователь:

\['Peer (Poire)' SIMILAR TO 'P[^ ]+ \(P[^ ]+\)' ESCAPE '\'' -- TRUE\]
\['Pera [Pear]' SIMILAR TO 'P[^ ]+ ![P[^ ]+!]' ESCAPE '!' -- TRUE\]
\['Paron-Appledryck' SIMILAR TO 'P%$-A%' ESCAPE '$' -- TRUE\]
\['Parondryck' SIMILAR TO 'P%^A%' ESCAPE '-' -- FALSE\]

**IS DISTINCT FROM**

Доступно в
DSQL, PSQL.

**Синтаксис**

\(<\text{operand1}> IS [NOT] DISTINCT FROM <\text{operand2}>\)

Два операнда считают DISTINCT (различными), если они имеют различные значения, или если одно из них — NULL, а другое нет. Они считаются NOT DISTINCT (равными), если имеют одинаковые значения или оба имеют значение NULL.
IS [NOT] DISTINCT FROM всегда возвращает TRUE или FALSE и никогда UNKNOWN (NULL) (неизвестное значение). Операторы ‘=’ и ‘<>’, наоборот, вернут UNKNOWN (NULL), если один или оба операнда имеют значение NULL.

**Table 25. Результаты выполнения различных операторов сравнения**

<table>
<thead>
<tr>
<th>Характеристики операнда</th>
<th>Результаты различных операторов</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=</td>
</tr>
<tr>
<td>Одинаковые значения</td>
<td>TRUE</td>
</tr>
<tr>
<td>Различные значения</td>
<td>FALSE</td>
</tr>
<tr>
<td>Оба NULL</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>Одно NULL</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

**Example 50. Использование предиката IS [NOT] DISTINCT FROM**

```sql
SELECT ID, NAME, TEACHER
FROM COURSES
WHERE START_DAY IS NOT DISTINCT FROM END_DAY

IF (NEW.JOB IS DISTINCT FROM OLD.JOB) THEN
    POST_EVENT 'JOB_CHANGED';
```

**См. также:**

Логический IS [NOT], IS [NOT] NULL.

Логический IS [NOT]

Доступно в DSQL, PSQL.

**Синтаксис**

```
<value> IS [NOT] {TRUE | FALSE | UNKNOWN}
```

Оператор IS проверяет, что выражение в левой части соответствует логическому значению в правой части. Выражение в левой части должно быть логического типа, иначе будет выдана ошибка.

Для логического типа данных предикат IS [NOT] UNKNOWN эквивалентен IS [NOT] NULL.

**Замечание:**

В правой части предиката могут быть использованы только литералы TRUE, FALSE, UNKNOWN, но не выражения.
**Example 51. Использование оператора IS с логическим типом данных**

```
-- Проверка FALSE значения
SELECT * FROM TBOOL WHERE BVAL IS FALSE
```

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>&lt;false&gt;</td>
</tr>
</tbody>
</table>

```
-- Проверка UNKNOWN значения
SELECT * FROM TBOOL WHERE BVAL IS UNKNOWN
```

<table>
<thead>
<tr>
<th>ID</th>
<th>BVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

**IS [NOT] NULL**

Доступно в
DSQL, PSQL.

**Синтаксис**

```
<value> IS [NOT] NULL
```

Поскольку NULL не является значением, эти операторы не являются операторами сравнения. Оператор IS [NOT] NULL проверяет, что выражение слева имеет значение (IS NOT NULL) или не имеет значения (IS NULL)

**Example 52. Использование предиката IS [NOT] NULL**

Поиск записей о продажах, для которых не установлена дата отгрузки:

```
SELECT *
FROM SALES
WHERE SHIP_DATE IS NULL;
```

**4.3.4. Предикаты существования**

В эту группу предикатов включены предикаты, которые используют подзапросы и передают значения для всех видов утверждений в условиях поиска. Предикаты существования называются так потому, что они различными способами проверяют
существование или отсутствие результатов подзапросов.

**EXISTS**

Доступно в
DSQL, PSQL, ESQL.

**Синтаксис**

\[
\text{[NOT] EXISTS (<select_stmt>)}
\]

Предикат EXISTS использует подзапрос в качестве аргумента. Если результат подзапроса будет содержать хотя бы одну запись, то предикат оценивается как истинный (TRUE), в противном случае предикат оценивается как ложный (FALSE).

Результат подзапроса может содержать несколько столбцов, поскольку значения не проверяются, а просто фиксируется факт наличия строк результата. Данный предикат может принимать только два значения: истина (TRUE) и ложь (FALSE).

Предикат NOT EXISTS возвращает FALSE, если результат подзапроса будет содержать хотя бы одну запись, в противном случае предикат вернёт TRUE.

**Example 53. Предикат EXISTS**

Найти тех сотрудников, у которых есть проекты.

```
SELECT *
FROM employee
WHERE EXISTS (SELECT *
               FROM employee_project ep
               WHERE ep.emp_no = employee.emp_no)
```

**Example 54. Предикат NOT EXISTS**

Найти тех сотрудников, у которых нет проектов.

```
SELECT *
FROM employee
WHERE NOT EXISTS (SELECT *
                  FROM employee_project ep
                  WHERE ep.emp_no = employee.emp_no)
```
Доступно в
DSQL, PSQL, ESQL.

Синтаксис

\[
<value> \ [NOT] \ IN \ (<select\_stmt> \ | \ <value\_list>)
\]

\[
<value\_list> ::=<value\_1> \ [\ , \ <value\_2> \ ...]
\]

Предикат IN проверяет, присутствует ли значение выражения слева в указанном справа наборе значений. Набор значений не может превышать 1500 элементов. Предикат IN может быть переписан в следующей эквивалентной форме:

\[
(<value> = <value\_1> \ [OR <value> = <value\_2> \ ...])
\]

При использовании предиката IN в поисковых условиях DML запросов, оптимизатор Firebird может использовать индекс по искомому столбцу, если он определён.

Во второй форме предикат IN проверяет, присутствует (или отсутствует, при использовании NOT IN) ли значение выражения слева в результате выполнения подзапроса справа. Результат подзапроса может содержать только один столбец, иначе будет выдана ошибка “count of column list and variable list do not match”.

Запросы с использованием предиката IN с подзапросом, можно переписать на аналогичный запрос с использованием предиката EXISTS. Например, следующий запрос:

```sql
SELECT
    model, speed, hd
FROM PC
WHERE
    model IN (SELECT model
                FROM product
                WHERE maker = 'A');
```

Можно переписать на аналогичный запрос с использованием предиката EXISTS:

```sql
SELECT
    model, speed, hd
FROM PC
WHERE
    EXISTS (SELECT *
             FROM product
             WHERE maker = 'A'
             AND product.model = PC.model);
```
Однако, запрос с использованием `NOT IN` не всегда даст тот же результат, что запрос `NOT EXISTS`. Причина заключается в том, что предикат `EXISTS` всегда возвращает `TRUE` или `FALSE`, тогда как предикат `IN` может вернуть `NULL` в следующих случаях:

a. Когда проверяемое значение равно `NULL` и список в `IN` не пуст.

b. Когда проверяемое значение не имеет совпадений в списке `IN` и одно из значений является `NULL`.

В этих двух случаях предикат `IN` вернёт `NULL`, в то время как соответствующий предикат `EXISTS` вернёт `FALSE`. В поисковых условиях или операторе `IF` оба результата обозначают "провал" и обрабатываются одинаково.

Однако на тех же данных `NOT IN` вернёт `NULL`, в то время как `EXISTS` вернёт `TRUE`, что приведёт к противоположному результату.

Это можно продемонстрировать следующим примером.

Предположим у вас есть такой запрос:

```sql
-- Ищем людей, которые не родились в тот же день, что
-- известные жители Нью-Йорка
SELECT P1.name AS NAME
FROM Personnel P1
WHERE P1.birthday NOT IN (SELECT C1.birthday
                            FROM Celebrities C1
                            WHERE C1.birthcity = 'New York');
```

Можно предположить, что аналогичный результат даст запрос с использованием предиката `NOT EXISTS`:

```sql
-- Ищем людей, которые не родились в тот же день, что
-- известные жители Нью-Йорка
SELECT P1.name AS NAME
FROM Personnel P1
WHERE NOT EXISTS (SELECT *
                  FROM Celebrities C1
                  WHERE C1.birthcity = 'New York'
                  AND C1.birthday = P1.birthday);
```

Допустим, что в Нью-Йорке всего один известный житель, и его дата рождения неизвестна. При использовании предиката `EXISTS` подзапрос внутри него не выдаст результатов, так как при сравнении дат рождения с `NULL` результатом будет `UNKNOWN`. Это приведёт к тому, что результат предиката `NOT EXISTS` будет истинен для каждой строки основного запроса. В то время как результатом предиката `NOT IN` будет `UNKNOWN` и ни одна строка не будет выведена.

**Example 55. Предикат `IN`**

Найти сотрудников с именами “Pete”, “Ann” и “Roger”:
SELECT *
FROM EMPLOYEE
WHERE FIRST_NAME IN ('Pete', 'Ann', 'Roger');

Example 56. Поисковый предикат IN

Найти все компьютеры, для которых существуют модели с производителем начинающимися на букву “A”:

SELECT
   model, speed, hd
FROM PC
WHERE
   model IN (SELECT model
              FROM product
              WHERE maker STARTING WITH 'A');

См. также:
EXISTS.

SINGULAR

Доступно в
DSQL, PSQL, ESQL.

Синтаксис

[NOT] SINGULAR (<select_stmt>)

Предикат SINGULAR использует подзапрос в качестве аргумента и оценивает его как истиный, если подзапрос возвращает одну и только одну строку результата, в противном случае предикат оценивается как ложный. Результат подзапроса может содержать несколько столбцов, поскольку значения не проверяются. Данный предикат может принимать только два значения: истина (TRUE) и ложь (FALSE).

Example 57. Предикат SINGULAR

Найти тех сотрудников, у которых есть только один проект.

SELECT *
FROM employee
WHERE SINGULAR (SELECT *
FROM employee_project ep
WHERE
4.3.5. Количественные предикаты подзапросов

Квантором называется логический оператор, задающий количество объектов, для которых данное утверждение истинно. Это логическое количество, а не числовое; оно связывает утверждение с полным множеством возможных объектов. Такие предикаты основаны на формальных логических квантификаторах общности и существования, которые распознаются формальной логикой.

В выражениях подзапросов количественные предикаты позволяют сравнивать отдельные значения с результатами подзапросов; их общая форма:

\[
\langle \text{value expression} \rangle \ <\text{comparison operator}> \ <\text{quantifier}> \ <\text{subquery}>
\]

**ALL**

Доступно в DSQL, PSQL.

**Синтаксис**

\[
\langle \text{value} \rangle \ <\text{op}> \ \text{ALL} \ (<\text{select_stmt}>)
\]

При использовании квантора ALL, предикат является истинным, если каждое значение выбранное подзапросом удовлетворяет условию в предикате внешнего запроса. Если подзапрос не возвращает ни одной строки, то предикат автоматически считается верным.

**Example 58. Квантор ALL**

Вывести только тех заказчиков, чьи оценки выше, чем у каждого заказчика в Париже.

```sql
SELECT *
FROM Customers
WHERE rating > ALL
(SELECT rating
FROM Customers
WHERE city = 'Paris')
```

Если подзапрос возвращает пустое множество, то предикат будет истинен для каждого левостороннего значения, независимо от оператора. Это может показаться странным и противоречивым, потому что в этом случае каждое левостороннее значение рассматривается как одновременно больше, меньше, равное и неравное любому значению из правого потока.
Тем не менее это нормально согласуется с формальной логикой: если множество пусто, то предикат верен 0 раз, т.е. для каждой строки в множестве.

**ANY и SOME**

*Доступно в DSQL, PSQL.*

**Синтаксис**

```
<value> <op> {ANY | SOME} (<select_stmt>)
```

Эти два квантора идентичны по поведению. Очевидно, оба представлены в стандарте SQL для взаимозаменяемого использования с целью улучшения читаемости операторов. При использовании квантора ANY или SOME, предикат является истинным, если любое из значений выбранное подзапросом удовлетворяет условию в предикате внешнего запроса. Если подзапрос не возвращает ни одной строки, то предикат автоматически считается ложным.

**Example 59. Квантор ANY**

Вывести только тех заказчиков, чьи оценки выше, чем у какого-либо заказчика в Риме.

```
SELECT *
FROM Customers
WHERE rating > ANY
    (SELECT rating
     FROM Customers
     WHERE city = 'Rome')
```
Chapter 5. Операторы определения данных (DDL)

Data Definition Language (DDL) — язык описания данных. С помощью этого подмножества языка создаются, модифицируются и удаляются объекты базы данных (т.е. Метаданные).

5.1. DATABASE

В данном разделе описываются вопросы создания базы данных, подключения к существующей базе данных, изменения структуры файлов, перевод базы данных в состояние, необходимое для безопасного резервного копирования, и обратно и удаления базы данных.

5.1.1. CREATE DATABASE

Назначение
Создание новой базы данных.

Доступно в
DSQL, ESQL

Синтаксис

```sql
CREATE {DATABASE | SCHEMA} <filespec>
[<db_initial_option> [<db_initial_option> ...]]
[<db_config_option> [<db_config_option> ...]]

<db_initial_option> ::= USER username
| PASSWORD 'password'
| ROLE rolename
| PAGE_SIZE [=] size
| LENGTH [=] num [PAGE[S]]
| SET NAMES 'charset'

<db_config_option> ::= DEFAULT CHARACTER SET default_charset
| COLLATION collation
| <sec_file>
| DIFFERENCE FILE 'diff_file'

<filespec> ::= '[<server_spec>]<filepath | db_alias>'

<server_spec> ::= host[/ {port | service}]:
| \host\
| <protocol>://[host::{port | service}]'/
```
Каждый `db_initial_option` и `db_config_option` может встречаться не более одного раза, за исключением `sec_file`, который может встречаться ноль или более раз.

### Table 26. Параметры оператора CREATE DATABASE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>filespec</td>
<td>Спецификация первичного файла базы данных.</td>
</tr>
<tr>
<td>server_spec</td>
<td>Спецификация удалённого сервера. Включает в себя имя сервера и протокол. Необходима, если база данных создаётся на удалённом сервере.</td>
</tr>
<tr>
<td>filepath</td>
<td>Полный путь и имя файла, включая расширение. Имя файла должно быть задано в соответствии со спецификой используемой платформы.</td>
</tr>
<tr>
<td>db_alias</td>
<td>Псевдоним (alias) базы данных, присутствующий в файле <code>databases.conf</code>.</td>
</tr>
<tr>
<td>host</td>
<td>Имя сервера или IP адрес, на котором создаётся база данных.</td>
</tr>
<tr>
<td>port</td>
<td>Номер порта, который слушает удалённый сервер (параметр <code>RemoteServicePort</code> файла <code>firebird.conf</code>).</td>
</tr>
<tr>
<td>service</td>
<td>Имя сервиса. Должно совпадать со значением параметра <code>RemoteServiceName</code> файла <code>firebird.conf</code>.</td>
</tr>
<tr>
<td>protocol</td>
<td>Наименование протокола.</td>
</tr>
<tr>
<td>username</td>
<td>Имя пользователя-владельца базы данных. Может быть заключено в одинарные или двойные кавычки. Если имя пользователя заключено в двойные кавычки, то оно чувствительно к регистру.</td>
</tr>
<tr>
<td>password</td>
<td>Пароль пользователя-владельца базы данных. Чувствительно к регистру.</td>
</tr>
<tr>
<td>role</td>
<td>Имя роли, права которой могут учитываться при создании базы данных. Может быть заключено в одинарные или двойные кавычки. Если имя роли заключено в двойные кавычки, то оно чувствительно к регистру.</td>
</tr>
<tr>
<td>size</td>
<td>Размер страницы для базы данных. Допустимые значения 4096, 8192, 16384, 32768. Размер страницы по умолчанию 8192.</td>
</tr>
<tr>
<td>num</td>
<td>Максимальный размер первичного или вторичного файла в страницах.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>charset</td>
<td>Задаёт набор символов подключения, доступного после успешного создания базы данных.</td>
</tr>
<tr>
<td>default_charset</td>
<td>Задаёт набор символов по умолчанию для строковых типов данных.</td>
</tr>
<tr>
<td>collation</td>
<td>Сортировка для набора символов по умолчанию.</td>
</tr>
<tr>
<td>sec_file</td>
<td>Спецификация вторичного файла.</td>
</tr>
<tr>
<td>pagenum</td>
<td>Номер страницы, с которой начинается вторичный файл базы данных.</td>
</tr>
<tr>
<td>diff_file</td>
<td>Путь и имя дельта файла.</td>
</tr>
</tbody>
</table>

Оператор CREATE DATABASE создаёт новую базу данных. Вы можете использовать CREATE DATABASE или CREATE SCHEMA. Это синонимы.

База данных может состоять из одного или нескольких файлов. Первый, основной, файл называется первичным, остальные файлы — вторичными.

В настоящее время многофайловые базы данных являются атавизмом. Многофайловые базы данных имеют смысл использовать на старых файловых системах, в которых существует ограничение на размер любого файла. Например, в FAT32 нельзя создать файл больше 4-х гигабайт.

Спецификация первичного файла — имя файла базы данных и его расширение с указанием к нему полного пути в соответствии с правилами используемой операционной системы. При создании базы данных файл базы данных должен отсутствовать. В противном случае будет выдано сообщение об ошибке и база данных не будет создана. Если полный путь к базе данных не указан, то база данных будет создана в одном из системных каталогов. В каком именно зависит от операционной системы.

**Использование псевдонимов БД**

Вместо полного пути к первичному файлу базы можно использовать псевдонимы (aliases). Псевдонимы описываются в файле databases.conf в формате:

```
alias = filepath
```

Помимо указания псевдонимов для базы данных в этом файле можно задать параметры уровня базы данных для каждой описываемой базы данных. Эти параметры задаются в фигурных скобках сразу после объявления псевдонима.

**Создание БД на удалённом сервере**

При создании базы данных на удалённом сервере необходимо указать спецификацию удалённого сервера. Спецификация удалённого сервера зависит от используемого протокола.
Если вы при создании базы данных используете протокол TCP/IP, то спецификация первичного файла должна выглядеть следующим образом:

```
host[/{port | service}]:{filepath | db_alias}
```

Если вы при создании базы данных используете протокол под названием именованные каналы (Name Pipes), то спецификация первичного файла должна выглядеть следующим образом.

```
\host\{filepath | db_alias}
```

Существует также унифицированный URL-подобный синтаксис спецификации удалённого сервера. В этом синтаксисе первым параметром указывается наименование протокола, далее указывается имя сервера или IP адрес, номер порта и путь к первичному файлу базы данных или псевдоним. В качестве протокола можно указать следующие значения:

**INET**
TCP/IP (сначала пробует подключиться по протоколу TCP/IP v6, если не получилось, то TCP/IP v4);

**INET4**
TCP/IP v4;

**INET6**
TCP/IP v6;

**WNET**
протокол именованных каналов (Named Pipes);

**XNET**
локальный протокол.

```
<protocol>://[host[:{port | service}]][{filepath | db_alias}
```

**Необязательные параметры CREATE DATABASE**

**USER** и **PASSWORD**

Необязательные предложения USER и PASSWORD задают, соответственно, имя и пароль пользователя присутствующего в базе данных безопасности (security4.fdb или той, что указана в параметре SecurityDatabase). Пользователя и пароль можно не указывать, если установлены переменные окружения ISC_USER и ISC_PASSWORD. Пользователь, указанный при создании базы данных, будет её владельцем.

**ROLE**

Необязательное предложение ROLE задаёт имя роли (обычно это RDB$ADMIN), права которой
будут учитываться при создании базы данных. Роль должна быть назначена пользователю в соответствующей базе данных безопасности.

**PAGE_SIZE**

Необязательное предложение `PAGE_SIZE` задаёт размер страницы базы данных. Этот размер будет установлен для первичного файла и всех вторичных файлов базы данных. При вводе размера страницы БД меньшего, чем 4096, он будет автоматически изменён на 4096. Другие числа (не равные 4096, 8192, 16384 или 32768) будут изменены на ближайшее меньшее из поддерживаемых значений. Если размер страницы базы данных не указан, то по умолчанию принимается значение 8192.

Больше не значит лучше

Большие размеры страницы могут вместить больше записей на одной странице, иметь более широкие индексы и больше индексов, но они также будут тратить больше места для BLOB (сравните потраченное впустую пространство BLOB размером 3 КБ на странице размером 4096 и такого же BLOB на 32768: +/- 1 КБ против +/- 29 КБ). Кроме того, при большом размере страницы увеличивается конкуренция за одну и ту же страницу данных, поскольку на неё вмещается больше записей, который могли бы располагаться на разных страницах.

**LENGTH**

Необязательное предложение `LENGTH` задаёт максимальный размер первичного или вторичного файла базы данных в страницах. При создании базы данных её первичный или вторичный файл будут занимать минимально необходимое количество страниц для хранения системных данных, не зависимо от величины, установленной в предложении `LENGTH`. Для единственного или последнего (в многофайловой базе данных) файла значение `LENGTH` никак не влияет на его размер. Файл будет автоматически увеличивать свой размер по мере необходимости.

**SET NAMES**

Необязательное предложение `SET NAMES` задаёт набор символов подключения, доступного после успешного создания базы данных. По умолчанию используется набор символов `NONE`.

**DEFAULT CHARACTER SET**

Необязательное предложение `DEFAULT CHARACTER SET` задаёт набор символов по умолчанию для строковых типов данных. Наборы символов применяются для типов `CHAR`, `VARCHAR` и `BLOB`. По умолчанию используется набор символов `NONE`. Для набора символов по умолчанию можно также указать сортировку по умолчанию (COLLATION). В этом случае сортировка станет умалчиваемой для набора символов по умолчанию (т.е. для всей БД за исключением случаев использования других наборов символов).

**STARTING AT**

Предложение `STARTING AT` задаёт номер страницы базы данных, с которой должен начинаться следующий файл базы данных. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы, система начнёт...
помещать вновь добавляемые данные в следующий файл базы данных.

**DIFFERENCE FILE**

Необязательное предложение DIFFERENCE FILE задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в БД после перевода её в режим “безопасного копирования” (“copy-safe”) путём выполнения команды ALTER DATABASE BEGIN BACKUP. Полное описание данного параметра см. в ALTER DATABASE.

**Диалект базы данных**

По умолчанию база данных создаётся в 3 диалекте. Для того чтобы база данных была создана в нужном вам диалекте SQL, следует перед выполнением оператора создания базы данных задать нужный диалект, выполнив оператор SET SQL DIALECT.

**Кто может создать базу данных?**

Выполнить оператор CREATE DATABASE могут:

- **Администраторы**;
- Пользователи с привилегией CREATE DATABASE.

**Примеры**

*Example 60. Создание базы данных в операционной системе Windows*

Создание базы данных в операционной системе Windows расположенной на диске D с размером страницы 8192. Владельцем базы данных будет пользователь wizard. База данных будет в 1 диалекте, и использовать набор символов по умолчанию WIN1251.

```sql
SET SQL DIALECT 1;
CREATE DATABASE 'D:\test.fdb'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET WIN1251;
```

*Example 61. Создание базы данных в операционной системе Linux*

Создание базы данных в операционной системе Linux с размером страницы 4096. Владельцем базы данных будет пользователь wizard. База данных будет в 3 диалекте, и использовать набор символов по умолчанию UTF8 с умолчаемой сортировкой UNICODE_CI_AI.

```sql
CREATE DATABASE '/home/firebird/test.fdb'
USER "wizard" PASSWORD 'player' ROLE 'RDB$ADMIN'
PAGE_SIZE = 4096
DEFAULT CHARACTER SET UTF8 COLLATION UNICODE_CI_AI;
```
В данном случае при создании базы данных будет учитываться регистр символов для имени пользователя, потому что оно указано в двойных кавычках.

Example 62. Создание базы данных на удалённом сервере

Создание базы данных на удалённом сервере baseserver расположенном по пути, на который ссылается псевдоним test, описанный в файле databases.conf. Используется протокол TCP. Владельцем базы данных будет пользователь wizard.

```
CREATE DATABASE 'baseserver:test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

То же самое с использованием унифицированного URL-подобного синтаксиса задания спецификации удалённого сервера.

```
CREATE DATABASE 'inet://baseserver:3050/test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

или

```
CREATE DATABASE 'inet://baseserver:gds_db/test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

При использовании доменных имён может быть полезно указать какой именно из протоколов IP v4 или IP v6 вы хотите использовать.

```
CREATE DATABASE 'inet4://baseserver/test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

или

```
CREATE DATABASE 'inet6://baseserver/test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

Создание базы данных с указанием IP адреса (IPv4) вместо указания имени сервера.

```
CREATE DATABASE '127:0:0:1:test'
```
Создание базы данных с указанием IP адреса (IPv6) вместо указания имени сервера.

```
CREATE DATABASE '[::1]:test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

**Example 63. Создание многофайлововой базы данных**

Создание базы данных в 3 диалекте с набором символов по умолчанию UTF8. Первичный файл будет содержать 10000 страниц с размером страницы 8192. Как только в процессе работы с базой данных первичный файл будет заполнен, СУБД будет помещать новые данные во вторичный файл `test.fdb2`. Аналогичные действия будут происходить и со вторым вторичным файлом. Размер последнего файла будет увеличиваться до тех пор, пока это позволяет используемая операционная система или пока не будет исчерпана память на внешнем носителе.

```
SET SQL DIALECT 3;
CREATE DATABASE 'baseserver:D:\test.fdb'
USER wizard PASSWORD 'player' ROLE 'RDB$ADMIN'
PAGE_SIZE = 8192
DEFAULT CHARACTER SET UTF8
FILE 'D:\test.fdb2'
STARTING AT PAGE 10001
FILE 'D:\test.fdb3'
STARTING AT PAGE 20001;
```

**Example 64. Создание многофайлововой базы данных 2**

Создание базы данных в 3 диалекте с набором символов по умолчанию UTF8. Первичный файл будет содержать 10000 страниц с размером страницы 8192. Как только в процессе работы с базой данных первичный файл будет заполнен, СУБД будет помещать новые данные во вторичный файл `test.fdb2`. Аналогичные действия будут происходить и со вторым вторичным файлом.

```
SET SQL DIALECT 3;
CREATE DATABASE 'baseserver:D:\test.fdb'
USER wizard PASSWORD 'player' ROLE 'RDB$ADMIN'
PAGE_SIZE = 8192
LENGTH 10000 PAGES
DEFAULT CHARACTER SET UTF8
FILE 'D:\test.fdb2'
FILE 'D:\test.fdb3'
```
См. также:
ALTER DATABASE, DROP DATABASE.

5.1.2. ALTER DATABASE

Назначение
Изменение структуры файлов базы данных, переключение её в состояние “безопасное для копирования” или изменение некоторых свойств базы данных.

Доступно в
DSQL, ESQL

Синтаксис

ALTER {DATABASE | SCHEMA}
    {<add_sec_clause> [...]} | {ADD DIFFERENCE FILE 'diff_file' | DROP DIFFERENCE FILE}
    | {{BEGIN | END} BACKUP}
    | {SET DEFAULT CHARACTER SET charset}
    | {SET DEFAULT SQL SECURITY {DEFINER | INVOKER}}
    | {SET LINGER TO linger_duration | DROP LINGER}
    | {ENCRYPT WITH plugin_name [KEY key_name] | DECRYPT}
    | {ENABLE | DISABLE} PUBLICATION
    | INCLUDE {TABLE <table_list> | ALL} TO PUBLICATION
    | EXCLUDE {TABLE <table_list> | ALL} FROM PUBLICATION

<add_sec_clause> ::= ADD <sec_file> [<sec_file> ...]

<sec_file> ::=
    FILE 'filepath'
    [STARTING [AT [PAGE]] pagenum]
    [LENGTH [=} num [PAGE[S]]

<table_list> ::= tablename [, tablename ...]

Table 27. Параметры оператора ALTER DATABASE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_sec_clause</td>
<td>Инструкция для добавления вторичного файла базы данных.</td>
</tr>
<tr>
<td>sec_file</td>
<td>Спецификация вторичного файла.</td>
</tr>
<tr>
<td>filepath</td>
<td>Полный путь и имя дельта файла или вторичного файла базы данных.</td>
</tr>
<tr>
<td>pagenum</td>
<td>Номер страницы, с которой начинается вторичный файл базы данных.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>num</td>
<td>Максимальный размер вторичного файла в страницах.</td>
</tr>
<tr>
<td>diff_file</td>
<td>Путь и имя дельта файла.</td>
</tr>
<tr>
<td>charset</td>
<td>Новый набор символов по умолчанию для базы данных.</td>
</tr>
<tr>
<td>linger_duration</td>
<td>Задержка в секундах.</td>
</tr>
<tr>
<td>plugin_name</td>
<td>Имя плагина шифрования.</td>
</tr>
<tr>
<td>key_name</td>
<td>Имя ключа шифрования.</td>
</tr>
<tr>
<td>table_list</td>
<td>Список таблиц, которые необходим разрешить или запретить для публикации (репликации).</td>
</tr>
<tr>
<td>tablename</td>
<td>Имя таблицы.</td>
</tr>
</tbody>
</table>

Оператор `ALTER DATABASE` изменяет структуру файлов базы данных или переключает её в состояние “безопасное для копирования”.

**Добавление вторичного файла**

Предложение `ADD FILE` добавляет к базе данных вторичный файл. Для вторичного файла необходимо указывать полный путь к файлу и имя вторичного файла. Описание вторичного файла аналогично тому, что описано в операторе `CREATE DATABASE`.

*Example 65. Добавление вторичного файла в базу данных*

Как только в предыдущем первичном или вторичных файлах будет заполнено 30000 страниц, СУБД будет помещать данные во вторичный файл `test4.fdb`.

```sql
ALTER DATABASE
ADD FILE 'D:test.fdb4'
STARTING PAGE 30001;
```

**Изменение пути и имени дельта файла**

Предложение `ADD DIFFERENCE FILE` задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в базу данных после перевода её в режим “безопасного копирования” (“copy-safe”). Этот оператор в действительности не добавляет файла. Он просто переопределяет умолчаемые имя и путь файла дельты. Для изменения существующих установок необходимо сначала удалить ранее указанное описание файла дельты с помощью оператора `DROP DIFFERENCE FILE`, а затем задать новое описание файла дельты. Если не переопределять путь и имя файла дельты, то он будет иметь тот же путь и имя, что и БД, но с расширением .delta.

При задании относительного пути или только имени файла файла дельты он будет создаваться в текущем каталоге сервера. Для операционных систем Windows это системный каталог.
Предложение DROP DIFFERENCE FILE удаляет описание (путь и имя) файла дельты, заданное ранее командой ADD DIFFERENCE FILE. На самом деле при выполнении этого оператора файл не удаляется. Он удаляет путь и имя файла дельты и при последующем переводе БД в режим “безопасного копирования” будут использованы значения по умолчанию (т.е. тот же путь и имя, что и у файла БД, но с расширением .delta).

Example 66. Установка пути и имени файла дельты

```
ALTER DATABASE ADD DIFFERENCE FILE 'D:\test.diff';
```

Example 67. Удаление описание файла дельты

```
ALTER DATABASE DROP DIFFERENCE FILE;
```

Перевод базы данных в режим “безопасного копирования”

Предложение BEGIN BACKUP предназначено для перевода базы данных в режим “безопасного копирования” (“copy-safe”). Этот оператор “замораживает” основной файл базы данных, что позволяет безопасно делать резервную копию средствами файловой системы, даже если пользователи подключены и выполняют операции с данными. При этом все изменения, вносимые пользователями в базу данных, будут записаны в отдельный файл, так называемый дельта файл (delta file).

Оператор BEGIN BACKUP, несмотря на синтаксис, не начинает резервное копирование базы данных, а лишь создаёт условия для его осуществления.

Предложение END BACKUP предназначено для перевода базы данных из режима “безопасного копирования” (“copy-safe”) в режим нормального функционирования. Этот оператор объединяет файл дельты с основным файлом базы данных и восстанавливает нормальное состояние работы, таким образом, закрывающая возможность создания безопасных резервных копий средствами файловой системы. (При этом безопасное резервное копирование с помощью утилиты gbak остаётся доступным).

Example 68. Перевод базы данных в режим “безопасного копирования”

```
ALTER DATABASE BEGIN BACKUP;
```
**Example 69. Возвращение базы данных в режим нормального функционирования из режима “безопасного копирования”**

```sql
ALTER DATABASE END BACKUP;
```

**Изменение набора символов по умолчанию**

Предложение `SET DEFAULT CHARACTER SET` изменяет набор символов по умолчанию для базы данных. Это изменение не затрагивает существующие данные. Новый набор символов по умолчанию будет использоваться только в последующих DDL командах, кроме того для них будет использоваться сортировка по умолчанию для нового набора символов.

**Example 70. Изменение набора символов по умолчанию для базы данных**

```sql
ALTER DATABASE SET DEFAULT CHARACTER SET WIN1251;
```

**Изменение привилегий выполнения по умолчанию**

Начиная с Firebird 4.0 появилась возможность указывать объектам метаданных с какими привилегиями они будут выполняться: вызывающего или определяющего пользователя. Для этого используется предложение `SQL SECURITY`, которое можно указать для таблиц, тритера, процедуры, функции или пакета. Если выбрана опция `INVOKER`, то объект метаданных будет выполняться с привилегиями вызывающего пользователя. Если выбрана опция `DEFINER`, то объект метаданных будет выполняться с привилегиями определяющего пользователя (владельца). Если при создании PSQL модуля или таблицы предложение `SQL SECURITY` не задано, то по умолчанию используется опция `INVOKER`.

Предложение `SET DEFAULT SQL SECURITY` изменяет привилегии выполнения с которым по умолчанию выполняются PSQL модули (хранимые процедуры, функции и пакеты).

**Example 71. Изменение привилегий выполнения по умолчанию**

После выполнения данного оператора PSQL модули по умолчанию будут выполняться с опцией `SQL SECURITY DEFINER`

```sql
ALTER DATABASE SET DEFAULT SQL SECURITY DEFINER;
```

**LINGER**

Предложение `SET LINGER` позволяет установить задержку закрытия базы данных. Этот механизм позволяет Firebird в режиме SuperServer, сохранять базу данных в открытом состоянии в течение некоторого времени после того как последние соединение закрыто, т.е. иметь механизм задержки закрытия базы данных. Это может помочь улучшить...
производительность и уменьшить издержки в случаях, когда база данных часто открывается и закрывается, сохраняя при этом ресурсы “разогретыми” до следующего открытия.

Такой режим может быть полезен для Web приложений, в которых коннект к базе обычно “живёт” очень короткое время.

Предложение DROP LINGER удаляет задержку и возвращает базу данных к нормальному состоянию (без задержки). Эта команда эквивалентна установке задержки в 0.

Удаление LINGER не самое лучшее решение для временной необходимости его отключения для некоторых разовых действий, требующих принудительного завершения работы сервера. Утилита gfix теперь имеет переключатель -NoLinger, который сразу закроет указанную базу данных, после того как последнего соединения не стало, независимо от установок LINGER в базе данных. Установка LINGER будет сохранена и нормально отработает в следующий раз.

Кроме того, одноразовое переопределение доступно также через сервисы API, с использованием тега isc_spb_prp_nolinger, например (в такой строке):

```
fbsvcmgr host:service_mgr user sysdba password xxx
action_properties dbname employee prp_nolinger
```

*Example 72. Установка задержки в 30 секунд*

```sql
ALTER DATABASE SET LINGER 30;
```

*Example 73. Удаление задержки*

```sql
ALTER DATABASE DROP LINGER;
```

или

```sql
ALTER DATABASE SET LINGER 0;
```

**Шифрование базы данных**

Оператор ALTER DATABASE с предложением ENCRYPT WITH шифрует базу данных с помощью указанного плагина шифрования. Шифрование начинается сразу после этого оператора и будет выполняться в фоновом режиме. Нормальная работа с базами данных не нарушается во время шифрования.
Процесс шифрования может быть проконтролирован с помощью поля MON$CRYPT_PAGE в псевдо-таблице MON$DATABASE или просмотрен на странице заголовка базы данных с помощью gstat -e.

gstat -h также будет предоставлять ограниченную информацию о состоянии шифрования.

Например, следующий запрос

```
select MON$CRYPT_PAGE * 100 / MON$PAGES from MON$DATABASE
```

будет отображать процент завершения процесса шифрования.

Необязательное предложение KEY позволяет передать имя ключа для плагина шифрования. Что делать с этим именем ключа решает плагин.

Оператор ALTER DATABASE с предложением DECRYPT дешифрует базу данных.

**Example 74. Шифрование базы данных**

```
ALTER DATABASE ENCRYPT WITH DbCrypt;
```

**Example 75. Дешифрование базы данных**

```
ALTER DATABASE DECRIPT;
```

**Управление репликацией**

Оператор ALTER DATABASE с предложением ENABLE PUBLICATION включает репликацию базы данных.

```
ALTER DATABASE ENABLE PUBLICATION
```

Для отключения репликации базы данных выполните оператор

```
ALTER DATABASE DISABLE PUBLICATION
```

Изменения будут применены сразу после подтверждения транзакции.

При настройке репликации должен быть определен набор репликации (он же публикация). Он включает в себя таблицы, которые должны быть реплицированы. Это также делается с помощью команды DDL:
При использовании ключевого слова `ALL` в набор репликации будут включены все таблицы, включая те что будут созданы позднее. Команда будет выглядеть следующим образом:

```
ALTER DATABASE INCLUDE ALL TO PUBLICATION
```

Вы можете задать конкретный набор таблиц для репликации. Для этого после ключевого слова `TABLE` необходимо указать список таблиц через запятую. В следующем примере мы разрешаем репликацию для таблиц `t1` и `t2`:

```
ALTER DATABASE INCLUDE TABLE t1, t2 TO PUBLICATION
```

Для исключения таблиц из набора репликации (публикации) используется следующий оператор:

```
ALTER DATABASE EXCLUDE {TABLE <table_list> | ALL} FROM PUBLICATION
```

При использовании ключевого слова `ALL` из набора репликации будут исключены все таблицы. Если ранее в публикацию были добавлены все таблицы с использованием ключевого слова `ALL`, то данный оператор отключит автоматическую публикацию для вновь создаваемых таблиц. Команда будет выглядеть следующим образом:

```
ALTER DATABASE EXCLUDE ALL FROM PUBLICATION
```

Вы можете задать конкретный набор таблиц для исключения из репликации. Для этого после ключевого слова `TABLE` необходимо указать список таблиц через запятую. В следующем примере мы исключаем таблицы `t1` и `t2` из набора репликации:

```
ALTER DATABASE EXCLUDE TABLE t1, t2 FROM PUBLICATION
```

Таблицы, включенные для репликации, могут быть дополнительно отфильтрованы с использованием двух параметров в файле конфигурации `replication.conf`: `include_filter` и `exclude_filter`. Это регулярные выражения, которые применяются к именам таблиц и определяют правила для включения таблиц в набор репликации или исключения их из набора репликации.
Кто может выполнить ALTER DATABASE?

Выполнить оператор ALTER DATABASE могут:

• Администраторы;
• Владелец базы данных;
• Пользователи с привилегией ALTER DATABASE.

См. также:
CREATE DATABASE, DROP DATABASE.

5.1.3. DROP DATABASE

Назначение
Удаление текущей базы данных.

Доступно в
DSQL, ESQL

Синтаксис

DROP DATABASE

Оператор DROP DATABASE удаляет текущую базу данных. Перед удалением базы данных, к ней необходимо присоединиться. Оператор удаляет первичный, все вторичные файлы и все файлы теневых копий.

Кто может удалить базу данных?

Выполнить оператор DROP DATABASE могут:

• Администраторы;
• Владелец базы данных;
• Пользователи с привилегией DROP DATABASE.

Примеры

Example 76. Удаление базы данных

Удаление базы данных, к которой подключен клиент.

DROP DATABASE;

См. также:
CREATE DATABASE, ALTER DATABASE.
Теневая копия (shadow — дословно тень) является точной страничной копией базы данных. После создания теневой копии все изменения, сделанные в базе данных, сразу же отражаются и в теневой копии. Если по каким либо причинам первичный файл базы данных станет недоступным, то СУБД переключится на теневую копию.

В данном разделе рассматриваются вопросы создания и удаления теневых копий.

5.2.1. CREATE SHADOW

Назначение
Создание теневой копии.

Синтаксис

```
CREATE SHADOW sh_num [AUTO | MANUAL] [CONDITIONAL]
'filepath' [LENGTH [=] num [PAGE[S]]]
[<secondary_file>];
```

Таблица 28. Параметры оператора CREATE SHADOW

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh_num</td>
<td>Номер теневой копии — положительное число, идентифицирующее набор файлов теневой копии.</td>
</tr>
<tr>
<td>filepath</td>
<td>Имя файла и путь к нему в соответствии с требованиями ОС.</td>
</tr>
<tr>
<td>num</td>
<td>Максимальный размер теневой копии в страницах.</td>
</tr>
<tr>
<td>secondary_file</td>
<td>Спецификация вторичного файла.</td>
</tr>
<tr>
<td>page_num</td>
<td>Номер страницы, с которой должен начинаться вторичный файл копии.</td>
</tr>
</tbody>
</table>

Оператор CREATE SHADOW создаёт новую теневую копию. Теневая копия начинает дублировать базу данных сразу в момент создания этой копии.

Теневые копии, как и база данных, могут состоять из нескольких файлов. Количество и размер файлов теневых копий не связано с количеством и размером файлов базы данных.
Для файлов теневой копии размер страницы устанавливается равным размеру страницы базы данных и не может быть изменён.

Если по каким либо причинам файл базы данных становится недоступным, то система преобразует тень в копию базы данных и переключается на неё. Теневая копия становится недоступной. Что будет дальше зависит от выбранного режима.

Режимы AUTO и MANUAL

Когда теневая копия преобразуется в базу данных она становится недоступной. Теневая копия может также стать недоступной если будет удалён её файл, или закончится место на диске, где она расположена, или если этот диск повреждён.

• Если выбран режим AUTO (значение по умолчанию), то в случае, когда теневая копия становится недоступной, автоматически прекращается использование этой копии и из базы данных удаляются все ссылки на нее. Работа с базой данных продолжается обычным образом без осуществления копирования в данную теневую копию.

Если указано ключевое слово CONDITIONAL, то система будет пытаться создать новую теневую копию, чтобы заменить потерянную. Это не всегда возможно, тогда вам потребуется создать новую тень вручную.

• Если выбран режим MANUAL, то в случае, когда теневая копия становится недоступной, все попытки соединения с базой данных и обращения к ней будут вызывать сообщение об ошибке до тех пор, пока теневая копия не станет доступной или пока не будет удалена администратором БД с помощью оператора DROP SHADOW.

Необязательные параметры CREATE SHADOW

LENGTH

Необязательное предложение LENGTH задаёт максимальный размер первичного или вторичного файла теневой копии в страницах. Для единственного или последнего файла теневой копии значение LENGTH никак не влияет на его размер. Файл будет автоматически увеличивать свой размер по мере необходимости.

STARTING AT

Предложение STARTING AT задаёт номер страницы теневой копии, с которой должен начинаться следующий файл теневой копии. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы, система начнёт помещать вновь добавляемые данные в следующий файл теневой копии.

Кто может создать теневую копию?

Выполнить оператор CREATE SHADOW могут:

• Администраторы;
• Владелец базы данных;
• Пользователи с привилегией ALTER DATABASE.
Примеры

Example 77. Создание теневую копию базы данных с номером 1

```
CREATE SHADOW 1 'g:\data\test.shd';
```

Example 78. Создание многофайловую теневую копии

```
CREATE SHADOW 2 'g:\data\test.sh1'
LENGTH 8000 PAGES
FILE 'g:\data\test.sh2';
```

См. также:
CREATE DATABASE, ALTER DATABASE, DROP SHADOW.

5.2.2. DROP SHADOW

Назначение
Удаление теневой копии.

Доступно в
DSQL, ESQL

Синтаксис

```
DROP SHADOW sh_num
[{{PRESERVE | DELETE} FILE}]
```

Table 29. Параметры оператора DROP SHADOW

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh_num</td>
<td>Номер теневой копии — положительное число, идентифицирующее набор файлов теневой копии.</td>
</tr>
</tbody>
</table>

Оператор DROP SHADOW удаляет указанную теневую копию из базы данных, с которой установлено текущее соединение. При удалении теневой копии прекращается процесс дублирования данных в эту копию. Если указана опция DELETE FILE, то будут также удалены и все связанные файлы с этой теневой копией. Если указана опция PRESERVE FILE, то файлы останутся не тронутыми. Это может быть полезно, если вы делаете резервную копию с теневого файла. По умолчанию используется опция DELETE FILE.

Кто может удалить теневую копию?

Выполнить оператор DROP SHADOW могут:
• Администраторы;
• Владелец базы данных;
• Пользователи с привилегией ALTER DATABASE.

Примеры

Example 79. Удаление теневой копии с номером 1

```
DROP SHADOW 1;
```

См. также:
CREATE SHADOW.

5.3. DOMAIN

Домен (Domain) — один из объектов реляционной базы данных, при создании которого можно задать некоторые характеристики, а затем использовать ссылку на домен при определении столбцов таблиц, объявлении локальных переменных, входных и выходных аргументов в модулях PSQL.

В данном разделе рассматриваются синтаксис операторов создания, модификации и удаления доменов. Подробное описание доменов и их использования можно прочесть в главе Пользовательские типы данных — домены.

5.3.1. CREATE DOMAIN

Назначение
Создание нового домена.

Доступно в
DSQL, ESQL.

Синтаксис

```
CREATE DOMAIN name [AS] <datatype>
  [DEFAULT {<literal> | NULL | <context_var}>]
  [NOT NULL] [CHECK (<dom_condition>)]
  [COLLATE collation_name];
```

<datatype> ::=  
  <scalar_datatype> | <blob_datatype> | <array_datatype>

<scalar_datatype> ::=  См. Синтаксис скалярных типов данных

<blob_datatype> ::=  См. Синтаксис типа данных BLOB
<array_datatype> ::= См. Синтаксис массивов

<dom_condition> ::= 
  <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT <val>
  | <val> IS [NOT] {TRUE | FALSE | UNKNOWN}
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<dom_condition>)
  | NOT <dom_condition>
  | <dom_condition> OR <dom_condition>
  | <dom_condition> AND <dom_condition>

<operator> ::= 
  <> | != | ^= | ~= | = | < | > | <= | >=
  | !< | ^< | «< | !> | ^> | »>

<val> ::= {
  VALUE
  | <literal>
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <cast_type>)
  | (<select_one>)
  | func(<val> [, <val> ...])
}

<cast_type> ::= 
  <datatype>
  | [TYPE OF] domain
  | TYPE OF COLUMN rel.col

Table 30. Параметры оператора CREATE DOMAIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Имя домена. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
</tbody>
</table>
Оператор `CREATE DOMAIN` создаёт новый домен.

В качестве базового типа домена можно указать любой тип данных SQL.

**Детали для конкретного типа**

**Массивы**

- Если домен должен быть массивом, базовым типом может быть любой тип данных SQL, кроме BLOB и массива.
- Размеры массива указаны в квадратных скобках. (В синтаксисе эти скобки заключены в кавычки, чтобы отличать их от квадратных скобок, обозначающих необязательные элементы синтаксиса).
- Для каждого измерения массива указывается одно или два целых числа, которые определяют нижнюю и верхнюю границы диапазона индекса:
  - По умолчанию массивы начинаются с 1. Нижняя граница является неявной, и необходимо указать только верхнюю границу. Когда указано только одно число меньше 1, то это определяет диапазон `num .. 1`, а число больше 1 определяет диапазон `1 .. num`.
  - Когда указано два числа, разделенных двоеточием (`:`) и необязательный пробел, то если второе большее, чем первое, это явно определяет диапазон индексов. Одна или обе границы могут быть меньше нуля, если верхняя граница больше нижней.
- Если массив имеет несколько измерений, определения диапазонов для каждого измерения должны быть разделены запятыми и необязательными пробелями.
- Индексы проверяются только, если значение массива действительно существует.
• Это означает, что сообщения об ошибках относительно недопустимых индексов не будут возвращаться, если выбор конкретного элемента массива ничего не вернет или если поле массива имеет значение NULL.

Строковые типы

Для типов CHAR, VARCHAR и BLOB с подтипом text можно указать набор символов в предложении CHARACTER SET. Если набор символов не указан, то по умолчанию принимается тот набор символов, который был указан при создании базы данных.

Если же при создании базы данных не был указан набор символов, то при создании домена по умолчанию принимается набор символов NONE. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец, основанный на таком домене, можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация не выполняется между исходными и конечными кодировками, что может приводить к ошибкам.

Предложение DEFAULT

Необязательное предложение DEFAULT позволяет указать значение по умолчанию для домена. Это значение будет помещено в столбец таблицы, который ссылает на данный домен, при выполнении оператора INSERT, если значение не будет указано для этого столбца. Локальные переменные и аргументы PSQL модулей, которые ссылаются на этот домен, будут инициализированы значением по умолчанию. В качестве значения по умолчанию может быть литерал совместимый по типу, неизвестное значение NULL и контекстная переменная, тип которой совместим с типом домена.

Ограничение NOT NULL

Предложение NOT NULL запрещает столбцам и переменным, основанным на домене, присваивать значение NULL.

Ограничение CHECK

Необязательное предложение CHECK задаёт ограничение домена. Ограничение домена задаёт условия, которому должны удовлетворять значения столбцов таблицы или переменных, которые ссылаются на данный домен. Условие должно быть помещено в круглые скобки. Условие — это логическое выражение, называемое также предикат, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN (эквивалент NULL). Если предикат возвращает FALSE, то значение не будет принято.

Ключевое слово VALUE

Ключевое слово VALUE в ограничении домена является заменителем столбца таблицы, который основан на данном домене, или переменной PSQL модуля. Оно содержит значение, присваиваемое переменной или столбцу таблицы. Ключевое слово VALUE может быть использовано в любом месте ограничения CHECK, но обычно его используют в левой части условия.
COLLATE
Необязательное предложение COLLATE позволяет задать порядок сортировки, если домен основан на одном из строковых типов данных (за исключением BLOB). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки умолчаемый для указанного набора сортировки при создании домена.

Кто может создать домен?
Выполнить оператор CREATE DOMAIN могут:

• Администраторы
• Пользователи с привилегией CREATE DOMAIN.

Пользователь, создавший домен, становится его владельцем.

Примеры

Example 80. Создание домена, который может принимать значения больше 1000.

```
CREATE DOMAIN CUSTNO AS INTEGER DEFAULT 10000 CHECK (VALUE > 1000);
```

Example 81. Создание домена, который может принимать значения 'Да' и 'Нет'.

```
CREATE DOMAIN D_BOOLEAN AS CHAR(3) CHECK (VALUE IN ('Да', 'Нет'));
```

Example 82. Создание домена с набором символов UTF8 и порядком сортировки UNICODE_CI_AI.

```
CREATE DOMAIN FIRSTNAME AS VARCHAR(30) CHARACTER SET UTF8 COLLATE UNICODE_CI_AI;
```

Example 83. Создание домена со значением по умолчанию.

```
CREATE DOMAIN D_DATE AS DATE DEFAULT CURRENT_DATE NOT NULL;
```
**Example 84. Создание домена, определённого как массив из 2 элементов.**

Создание домена, определённого как массив из 2 элементов типа NUMERIC(18, 3), нумерация элементов начинается с 1.

```
CREATE DOMAIN D_POINT AS NUMERIC(18, 3) [2];
```

Вы можете использовать домены определённые как массив только для определения столбцов таблиц. Вы не можете использовать такие домены для определения локальных переменных и аргументов PSQL модулей.

См. также:
ALTER DOMAIN, DROP DOMAIN.

### 5.3.2. ALTER DOMAIN

**Назначение**

Изменение текущих характеристик домена или его переименование.

**Доступно в**

DSQL, ESQL.

**Синтаксис**

```
ALTER DOMAIN domain_name
    [TO new_name]
    [TYPE <datatype>]
    [{SET DEFAULT {<literal> | NULL | <context_var>} | DROP DEFAULT]
    [{SET | DROP} NOT NULL]
    [{ADD [CONSTRAINT] CHECK (<dom_condition>) | DROP CONSTRAINT]
```

<datatype> ::= <scalar_datatype> | <blob_datatype> | <array_datatype>

<scalar_datatype> ::= См. Синтаксис скалярных типов данных

<blob_datatype> ::= См. Синтаксис типа данных BLOB

<array_datatype> ::= См. Синтаксис массивов

<dom_condition> ::= 
    <val> <operator> <val>
    | <val> [NOT] BETWEEN <val> AND <val>
    | <val> [NOT] IN (\<val> [, \<val> ...] | \<select_list>)
    | <val> IS [NOT] NULL
    | <val> IS [NOT] DISTINCT <val>
| <val> IS [NOT] {TRUE | FALSE | UNKNOWN} |
| <val> [NOT] CONTAINING <val> |
| <val> [NOT] STARTING [WITH] <val> |
| <val> [NOT] LIKE <val> [ESCAPE <val>] |
| <val> [NOT] SIMILAR TO <val> [ESCAPE <val>] |
| <val> <operator> {ALL | SOME | ANY} (<select_list>) |
| [NOT] EXISTS (<select_expr>) |
| [NOT] SINGULAR (<select_expr>) |
| (<dom_condition>) |
| NOT <dom_condition> |
| <dom_condition> OR <dom_condition> |
| <dom_condition> AND <dom_condition> |

<operator> ::=<br> <> | != | ^= | ^= | = | > | < | <= | >= |
| !< | ^< | ^< | ^> | ^> | >|

<val> ::=<br> VALUE |
| <literal> |
| <context_var> |
| <expression> |
| NULL |
| NEXT VALUE FOR genname |
| GEN_ID(genname, <val>) |
| CAST(<val> AS <cast_type>) |
| (<select_one>) |
| func(<val> [, <val> ...]) |

<cast_type> ::=<br> <datatype> |
| [TYPE OF] domain |
| TYPE OF COLUMN rel.col |

Table 31. Параметры оператора ALTER DOMAIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
<tr>
<td>new_name</td>
<td>Новое имя домена. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом данных домена.</td>
</tr>
<tr>
<td>dom_condition</td>
<td>Условие домена.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>select_one</td>
<td>Оператор SELECT выбирающий один столбец и возвращающий только одну строку.</td>
</tr>
<tr>
<td>select_list</td>
<td>Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.</td>
</tr>
<tr>
<td>select_expr</td>
<td>Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.</td>
</tr>
<tr>
<td>expression</td>
<td>Выражение.</td>
</tr>
<tr>
<td>genname</td>
<td>Имя последовательности (генератора).</td>
</tr>
<tr>
<td>func</td>
<td>Скалярная функция.</td>
</tr>
</tbody>
</table>

Оператор ALTER DOMAIN изменяет текущие характеристики домена, в том числе и его имя. В одном операторе ALTER DOMAIN можно выполнить любое количество изменений домена.

**TO name**

Предложение TO позволяет переименовать домен. Имя домена можно изменить, если не существует зависимостей от этого домена, т.е. столбцов таблиц, локальных переменных и аргументов процедур, ссылающихся на данный домен.

**SET DEFAULT**

Предложение SET DEFAULT позволяет установить новое значение по умолчанию. Если домен уже содержал значение по умолчанию, то установка нового значения по умолчанию не требует предварительного удаления старого.

**DROP DEFAULT**

Предложение DROP DEFAULT удаляет ранее установленное для домена значение по умолчанию. В этом случае значением по умолчанию становится значение NULL.

**ADD CONSTRAINT CHECK**

Предложение ADD [CONSTRAINT] CHECK добавляет условие ограничения домена. Если домен уже содержал ограничение CHECK, то его предварительно необходимо удалить с помощью предложения DROP CONSTRAINT.

**TYPE**

Предложение TYPE позволяет изменить тип домена на другой допустимый тип. Не допустимы любые изменения типа, которые могут привести к потере данных. Например, количество символов в новом типе для домена не может быть меньше, чем было установлено ранее.

Изменение типа не поддерживается для типа BLOB и массивов.

**SET NOT NULL**

Предложение SET NOT NULL устанавливает ограничение NOT NULL для домена. В этом случае для переменных и столбцах базирующихся на домене значение NULL не допускается.
Успешная установка ограничения NOT NULL для домена происходит только после полной проверки данных таблиц, столбцы которых базируются на домене. Это может занять довольно длительное время.

При изменении описания домена, существующий PSQL код, может стать некорректным. Информация о том, как это обнаружить, находится в приложении Полос RDB$VALID_BLR.

**DROP NOT NULL**

Предложение DROP NOT NULL удаляет ограничение NOT NULL для домена.

**Что не может изменить ALTER DOMAIN**

- Если домен был объявлен как массив, то изменить ни его тип, ни размерность нельзя. Также нет возможности изменить любой другой тип на тип массив.
- Не существует способа изменить сортировку по умолчанию. В этом случае необходимо удалить домен и пересоздать его с новыми атрибутами.

**Кто может изменить домен?**

Выполнить оператор ALTER DOMAIN могут:

- Администраторы
- Владелец домена;
- Пользователи с привилегией ALTER ANY DOMAIN.

**Примеры**

*Example 85. Изменение значения по умолчанию для домена.*

```
ALTER DOMAIN CUSTNO
INTEGER DEFAULT 2000;
```

*Example 86. Переименование домена.*

```
ALTER DOMAIN D_BOOLEAN TO D_BOOL;
```

*Example 87. Удаление значения по умолчанию и добавления ограничения для домена.*

```
ALTER DOMAIN D_DATE
DROP DEFAULT
ADD CONSTRAINT CHECK (VALUE >= date '01.01.2000');
```
Example 88. Изменение ограничения домена.

```sql
ALTER DOMAIN D_DATE
DROP CONSTRAINT;

ALTER DOMAIN D_DATE
ADD CONSTRAINT CHECK
(VALUE BETWEEN date '01.01.1900' AND date '31.12.2100');
```

Example 89. Изменение типа домена.

```sql
ALTER DOMAIN FIRSTNAME
TYPE VARCHAR(50) CHARACTER SET UTF8;
```

Example 90. Добавление ограничения NOT NULL для домена.

```sql
ALTER DOMAIN FIRSTNAME SET NOT NULL;
```

См. также:
CREATE DOMAIN, DROP DOMAIN.

5.3.3. DROP DOMAIN

Назначение
Удаление существующего домена.

Доступно в
DSQL, ESQL.

Синтаксис

```sql
DROP DOMAIN domain_name
```

Table 32. Параметры оператора DROP DOMAIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
</tbody>
</table>

Оператор DROP DOMAIN удаляет домен, существующий в базе данных. Невозможно удалить домен, на который ссылаются столбцы таблиц базы данных или если он был задействован в одном из PSQL модулей. Чтобы удалить такой домен, необходимо удалить из таблиц все столбцы, ссылающиеся на домен и удалить все ссылки на домен из PSQL модулей.
Кто может удалить домен?

Выполнить оператор DROP DOMAIN могут:

- Администраторы
- Владелец домена;
- Пользователи с привилегией DROP ANY DOMAIN.

Примеры

Example 91. Удаление домена

```sql
DROP DOMAIN COUNTRYNAME;
```

См. также:

CREATE DOMAIN, ALTER DOMAIN.

5.4. TABLE


В данном разделе рассматриваются вопросы создания, модификации и удаления таблиц базы данных.

5.4.1. CREATE TABLE

Назначение

Создание новой таблицы.

Доступно в

DSQL, ESQL

Синтаксис

```sql
CREATE [GLOBAL TEMPORARY] TABLE tablename
[EXTERNAL [FILE] 'filespec']
(<col_def> [, <col_def> | <tconstraint> ...])
[ON COMMIT {DELETE | PRESERVE} ROWS]
[SQL SECURITY {DEFINER | INVOKER}]
[{{ENABLE | DISABLE} PUBLICATION]}

<col_def> ::=
```
<regular_col_def> ::= 
| <computed_col_def> 
| <identity_col_def>

<regular_col_def> ::= 
   colname { <datatype> | domain_name } 
   [DEFAULT {<literal> | NULL | <context_var>}] 
   [NOT NULL] 
   [<col_constraint>] 
   [COLLATE collation_name]

<computed_col_def> ::= 
   colname [{ <datatype> | domain_name }] 
   {COMPUTED [BY] | GENERATED ALWAYS AS} (<expression>)

<identity_col_def> ::= 
   colname [ <datatype>] 
   GENERATED {ALWAYS | BY DEFAULT} AS IDENTITY [(<identity column options>)] 
   [<col_constraint>]

<identity column options> ::= 
   <identity column option> [<identity column option>]

<identity column option> ::= 
   START WITH startvalue 
   | INCREMENT [BY] incrementvalue

<datatype> ::= 
   <scalar_datatype> | <blob_datatype> | <array_datatype>

<scalar_datatype> ::= См. Синтаксис скалярных типов данных

<blob_datatype> ::= См. Синтаксис типа данных BLOB

<array_datatype> ::= См. Синтаксис массивов

<col_constraint> ::= [CONSTRAINT constr_name] 
   { UNIQUE [<using_index>] 
     | PRIMARY KEY [<using_index>] 
     | REFERENCES other_table [(other_col)] 
       [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}] 
       [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}] 
       [<using_index>] 
     | CHECK (<check_condition>) }

<tconstraint> ::= [CONSTRAINT constr_name] 
   { UNIQUE (<col_list>) [<using_index>] 
     | PRIMARY KEY (<col_list>) [<using_index>] 
     | ...}
| FOREIGN KEY (<col_list>)
| REFERENCES other_table [(<col_list>)]
| [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
| [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
| [using_index]
| CHECK (<check_condition>)
|
<col_list> ::= colname [, colname ...]

<using_index> ::= USING [ASCENDING | DESCENDING] INDEX indexname


<operator> ::=<br>   <> | != | ^= | ~= | = | < | > | <= | >=<br>   | !< | ^< | ~< | !> | ^> | ~>

<val> ::=<br>   colname ['[<array_idx> [, <array_idx> ...]]']
   | <literal>
   | <context_var>
   | <expression>
   | NULL
   | NEXT VALUE FOR genname
   | GEN_ID(genname, <val>)
   | CAST(<val> AS <cast_type>)
   | (<select_one>)
   | func(<val> [, <val> ...])

<cast_type> ::=<br>   <datatype>
   | [TYPE OF] domain_name
| TYPE OF COLUMN | rel.colname |

### Table 33. Параметры опператора CREATE TABLE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Имя таблицы, может содержать до 63 символов.</td>
</tr>
<tr>
<td>filespec</td>
<td>Спецификация файла (только для внешних таблиц).</td>
</tr>
<tr>
<td>colname</td>
<td>Имя столбца таблицы, может содержать до 63 символов.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
<tr>
<td>startvalue</td>
<td>Начальное значение столбца идентификации.</td>
</tr>
<tr>
<td>identityvalue</td>
<td>Приращение столбца идентификации. Не может быть равно 0.</td>
</tr>
<tr>
<td>col_constraint</td>
<td>Ограничение столбца.</td>
</tr>
<tr>
<td>tconstraint</td>
<td>Ограничение таблицы.</td>
</tr>
<tr>
<td>constr_name</td>
<td>Имя ограничения, может содержать до 63 символов.</td>
</tr>
<tr>
<td>other_table</td>
<td>Имя таблицы, на которую ссылается внешний ключ.</td>
</tr>
<tr>
<td>other_col</td>
<td>Столбец таблицы, на которую ссылается внешний ключ.</td>
</tr>
<tr>
<td>using_index</td>
<td>Позволяет задать имя автоматически создаваемого индекса для ограничения, и опционально определить, какой это будет индекс — по возрастанию (по умолчанию) или по убыванию.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом данных столбца.</td>
</tr>
<tr>
<td>check_condition</td>
<td>Условие проверки ограничения. Выполняется, если оценивается как TRUE или NULL/UNKNOWN.</td>
</tr>
<tr>
<td>collation_name</td>
<td>Порядок сортировки. Нужно указывать если вы хотите чтобы порядок сортировки для столбца отличался от порядка сортировки для набора символов по умолчанию этого столбца.</td>
</tr>
<tr>
<td>select_one</td>
<td>Оператор SELECT выбирающий один столбец и возвращающий только одну строку.</td>
</tr>
<tr>
<td>select_list</td>
<td>Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.</td>
</tr>
<tr>
<td>select_expr</td>
<td>Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.</td>
</tr>
<tr>
<td>expresssion</td>
<td>Выражение.</td>
</tr>
<tr>
<td>genname</td>
<td>Имя последовательности (генератора).</td>
</tr>
<tr>
<td>func</td>
<td>Скалярная функция.</td>
</tr>
</tbody>
</table>

Оператор CREATE TABLE создаёт новую таблицу. Имя таблицы должно быть уникальным.
среди имён всех таблиц, представлений (VIEWs) и хранимых процедур базы данных.

Таблица может содержать, по меньшей мере, один столбец и произвольное количество ограничений таблицы.

Имя столбца должно быть уникальным для создаваемой таблицы. Для столбца обязательно должен быть указан либо тип данных, либо имя домена, характеристики которого будут скопированы для столбца, либо должно быть указано, что столбец является вычисляемым.

В качестве типа столбца можно использовать любой тип данных SQL.

**Символьные столбцы**

Для типов CHAR, VARCHAR и BLOB с подтипа TEXT можно указать набор символов в предложении CHARACTER SET. Если набор символов не указан, то по умолчанию принимается тот набор символов, что был указан при создании базы данных. Если же при создании базы данных не был указан набор символов, то по умолчанию принимается набор символов NONE. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация между исходными и конечными кодировками не выполняется, что может приводить к ошибкам.

Необязательное предложение COLLATE позволяет задать порядок сортировки для строковых типов данных (за исключением BLOB). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки по умолчанию для указанного набора сортировки.

**Ограничение NOT NULL**

По умолчанию столбец может принимать значение NULL.

Необязательное предложение NOT NULL указывает, что столбцу не может быть присвоено значение NULL.

**Значение по умолчанию**

Необязательное предложение DEFAULT позволяет указать значение по умолчанию для столбца таблицы. Это значение будет помещено в столбец таблицы при выполнении оператора INSERT, если значение не будет указано для этого столбца. В качестве значения по умолчанию может быть литерал совместимый по типу, неизвестное значение NULL или контекстная переменная, тип которой совместим с типом столбца. Если значение по умолчанию явно не устанавливается, то подразумевается пустое значение, NULL. Использование выражений в значении по умолчанию недопустимо.

**Столбцы основанные на домене**

Для определения столбца, можно воспользоваться ранее описанным доменом. Если определение столбца основано на домене, оно может включать новое значение по умолчанию, дополнительные ограничения CHECK, предложение COLLATE, которые перекрывают значения указанные при определении домена. Определение такого столбца может включать дополнительные ограничения столбца, например NOT NULL, если домен
его ещё не содержит.

Следует обратить внимание на то, что если в определении домена было указано NOT NULL, на уровне столбца невозможно определить допустимость использования в нем значения NULL. Если вы хотите чтобы на основе домена можно было определять столбцы допускающие псевдозначение NULL и не допускающее его, то хорошей практикой является создание домена допускающего NULL и указание ограничения NOT NULL у столбцов таблицы там где это необходимо.

Столбцы идентификации (автоинкремент)

Столбец идентификации представляет собой столбец, связанный с внутренним генератором последовательностей. Столбцы идентификации могут быть определены либо с помощью предложения GENERATED BY DEFAULT AS IDENTITY, либо предложения GENERATED ALWAYS AS IDENTITY.

Если столбец идентификации задан как GENERATED BY DEFAULT, то его значение будет увеличиваться и использовано как значение по умолчанию при каждой вставке, только в том случае, если значение этого столбца не задано явно.

Чтобы использовать сгенерированное по умолчанию значение, необходимо либо указать ключевое слово DEFAULT при вставке в столбец идентификации, или просто не упоминать столбец идентификации в списке столбцов для вставки. В противном случае будет использовано указанное вами значение.

**Example 92. Столбец определённый как GENERATED BY DEFAULT AS IDENTITY**

```sql
CREATE TABLE greetings (
    id INT GENERATED BY DEFAULT AS IDENTITY,
    name CHAR(50));

-- specify value "1":
INSERT INTO greetings VALUES (1, 'hi');

-- use generated default
INSERT INTO greetings VALUES (DEFAULT, 'salut');

-- use generated default
INSERT INTO greetings(ch) VALUES ('bonjour');
```

Это поведение может быть изменено в операторе INSERT если указана директива OVERRIDING USER VALUE. Подробнее см. Директива OVERRIDING.

Если столбец идентификации задан как GENERATED ALWAYS, то его значение будет увеличиваться при каждой вставке. При попытке явно присвоить значение столбца идентификации в операторе INSERT, будет выдано сообщение об ошибке. В операторе INSERT
вы можете указать ключевое слово DEFAULT вместо значения для столбца идентификации.

```
create table greetings (
    id INT GENERATED ALWAYS AS IDENTITY,
    name CHAR(50));

INSERT INTO greetings VALUES (DEFAULT, 'hello');

INSERT INTO greetings(ch) VALUES ('bonjour');
```

Это поведение может быть изменено в операторе INSERT если указана директива OVERRIDING SYSTEM VALUE. Подробнее см. Директива OVERRIDING.

Необязательное предложение START WITH позволяет указать начальное значение отличное от нуля. Предложение INCREMENT [BY] устанавливает значение приращения. Значение приращения должно быть отлично от 0. По умолчанию значение приращения равно 1.

Правила

• Тип данных столбца идентификации должен быть целым числом с нулевым масштабом. Допустимыми типами являются SMALLINT, INTEGER, BIGINT, NUMERIC(x, 0) и DECIMAL(x, 0);

• Идентификационный столбец не может иметь DEFAULT и COMPUTED значений.

• Идентификационный столбец может быть изменён, чтобы стать обычным столбцом. Обычный столбец не может быть изменён, чтобы стать идентификационным.

• Идентификационные столбцы неявно являются NOT NULL столбцами.

• Уникальность не обеспечивается автоматически. Ограничения UNIQUE или PRIMARY KEY требуются для гарантии уникальности.

См. также:
Директива OVERRIDING.

Вычисляемые поля

Вычисляемые поля могут быть определены с помощью предложения COMPUTED [BY] или GENERATED ALWAYS AS (согласно стандарту SQL-2003). Они эквивалентны по смыслу. Для вычисляемых полей не требуется описывать тип данных (но допустимо), СУБД вычисляет подходящий тип в результате анализа выражения. В выражении требуется указать корректную операцию для типов данных столбцов, входящих в его состав. При явном указании типа столбца для вычисляемого поля результат вычисления приводится к указанному типу, то есть, например, результат числового выражения можно вывести как строку. Вычисление выражения происходит для каждой строки выбранных данных, если в операторе выборки данных SELECT, присутствует такой столбец.
Вместо использования вычисляемого столбца в ряде случаев имеет смысл использовать обычный столбец, значение которого рассчитывается в триггерах на добавление и обновление данных. Это может снизить производительность вставки/модификации записей, но повысит производительность выборки данных.

Столбцы типа массив

Для любого типа данных кроме BLOB можно указать размерность массива, если столбец должен быть массивом. Размерность массива указывается в квадратных скобках. Чтобы не перепутать их с символами, означающими необязательные элементы, они выделены жирным шрифтом. При указании размерности массива указываются два числа через двоеточие. Первое число означает начальный номер элемента массива, второе — конечный. Если указано только одно число, то оно означает последний номер в элементе массива, а первым номером считается 1. Для многомерного массива размерности массива перечисляются через запятую.

Ограничения

Существуют четыре вида ограничений:

- первичный ключ (PRIMARY KEY);
- уникальный ключ (UNIQUE);
- внешний ключ (REFERENCES или FOREIGN KEY);
- проверочное ограничение (CHECK).

Ограничения могут быть указаны на уровне столбца ("ограничения столбцов") или на уровне таблицы ("табличные ограничения"). Ограничения уровня таблицы необходимы, когда ключи (ограничение уникальности, первичный ключ или внешний ключ) должны быть сформированы по нескольким столбцам, или, когда ограничение CHECK включает несколько столбцов, т.е. действует на уровне записи. Синтаксис для некоторых типов ограничений может незначительно отличаться в зависимости от того определяется ограничение на уровне столбца или на уровне таблицы.

- Ограничение на уровне столбца указывается после определения других характеристик столбца. Оно может включать только столбец указанный в этом определении.
- Ограничения на уровне таблицы указываются после определений всех столбцов. Ограничения таблицы являются более универсальным способом записи ограничений, поскольку позволяют ограничение более чем для одного столбца таблицы.
- Вы можете смешивать ограничения столбцов и ограничения таблиц в одном операторе CREATE TABLE.

Системой автоматически создаётся индекс для первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE KEY) и внешнего ключа (REFERENCES для ограничения уровня столбца, и FOREIGN KEY REFERENCES для ограничения уровня таблицы).
Имена для ограничений и их индексов

Если имя ограничения не задано, то оно автоматически будет сгенерировано системой.

Ограничения уровня столбца и их индексы автоматически именуются следующим образом:

• Имена ограничений имеют следующий вид INTEG_<n>, где n представлено одним или несколькими числами;

• Имена индексов имеют вид RDB$PRIMARY<n> (для индекса первичного ключа), RDB$FOREIGN<n> (для индекса внешнего ключа) или RDB$<n> (для индекса уникального ключа), где n представлено одним или несколькими числами;

Схемы автоматического формирования имен для ограничений уровня таблицы и их индексов одинаковы.

Именованные ограничения

Имя ограничения можно задать явно, если указать его в необязательном предложении CONSTRAINT. По умолчанию имя индекса ограничения будет тем же самым, что и самого ограничения. Если для индекса необходимо задать другое имя, то его можно указать в предложении USING.

Предложение USING

Предложение USING позволяет задать определённое пользователем имя автоматически создаваемого индекса для ограничения, и опционально определить, какой это будет индекс — по возрастанию (по умолчанию) или по убыванию.

Первичный ключ (PRIMARY KEY)

Ограничение первичного ключа PRIMARY KEY строится на поле с заданным ограничением NOT NULL и требует уникальности значений столбца. Таблица может иметь только один первичный ключ.

• Первичный ключ по единственному столбцу может быть определён как на уровне столбца, так и на уровне таблицы.

• Первичный ключ по нескольким столбцам может быть определён только на уровне таблицы.

Ограничение уникальности (UNIQUE)

Ограничение уникального ключа UNIQUE задаёт для значений столбца требование уникальности содержимого. Таблица может содержать любое количество уникальных ключей.

Как и первичный ключ, ограничение уникальности может быть определено на нескольких столбцах. В этом случае вы должны определять его как ограничение уровня таблицы.
**NULL в уникальных ключах**

Согласно стандарту SQL-99 Firebird допускает одно или более значений NULL в столбце на который наложено ограничение UNIQUE. Это позволяет определить ограничение UNIQUE на столбцах, которые не имеют ограничения NOT NULL.

Для уникальных ключей, содержащих несколько столбцов, логика немного сложнее:

- Разрешено множество записей со значением NULL во всех столбцах ключа;
- Разрешено множество записей с различными комбинациями null и not-null значений в ключах;
- Разрешено множество записей, в которых в одном из столбцов уникального ключа содержится значение NULL, а остальные столбцы заполнены значениями и эти значения различны хотя бы в одном из них;
- Разрешено множество записей, в которых в одном из столбцов уникального ключа содержится значение NULL, а остальные столбцы заполнены значениями, и эти значения имеют совпадения хотя бы в одном из них.

Это можно резюмировать следующим примером:

```
RECREATE TABLE t( x int, y int, z int, unique(x,y,z));
INSERT INTO t values( NULL, 1, 1 );
INSERT INTO t values( NULL, NULL, 1 );
INSERT INTO t values( NULL, NULL, NULL );
INSERT INTO t values( NULL, NULL, NULL ); -- Разрешено
```

**Внешний ключ (FOREIGN KEY)**

Ограничение внешнего ключа гарантирует, что столбец (столбцы) участник может содержать только те значения, которые существуют в указанном столбце (столбцах) главной таблицы. Эти ссылочные столбцы часто называют столбцами назначения. Они должны быть первичным ключом или уникальным ключом в целевой таблице. Они могут не иметь ограничения NOT NULL, если они входят в ограничение уникального ключа.

Столбцы внешнего ключа не требуют ограничения NOT NULL.

На уровне столбца ограничение внешнего ключа определяется с использованием ключевого слова REFERENCES.

```
...,
ARTIFACT_ID INTEGER REFERENCES COLLECTION (ARTIFACT_ID),
```

В этом примере столбец ARTIFACT_ID ссылается на столбец с тем же именем в таблице COLLECTION.

На уровне таблицы могут быть определены внешний ключ над одним или несколькими
столбцами. Внешние ключи над несколькими столбцами можно определить только на уровне таблицы.

Синтаксис определения внешнего ключа на уровне таблицы несколько отличается. После определения всех столбцов, с их ограничения уровня столбца, вы можете определить именованное ограничение внешнего ключа уровня таблицы, используя ключевые слова `FOREIGN KEY` и имён столбцов для которых оно применяется:

```
... ,
CONSTRAINT FK_ARTSOURCE FOREIGN KEY(DEALER_ID, COUNTRY)
REFERENCES DEALER (DEALER_ID, COUNTRY),
```

Обратите внимание на то, что имена столбцов в целевой (master) таблице могут отличаться от тех что указаны во внешнем ключе.

Если целевые столбцы не указаны, то внешний ключ автоматически ссылается на столбцы первичного ключа целевой таблицы.

**Действия внешнего ключа**

Для обеспечения дополнительной целостности данных можно указать необязательные опции, которые обеспечат согласованность данных между родительскими и дочерними таблицами по заданным правилам:

- Предложение `ON UPDATE` определяет, что произойдёт с записями подчинённой таблицы при изменении значения первичного/уникального ключа в строке главной таблицы.
- Предложение `ON DELETE` определяет, что произойдёт с записями подчинённой таблицы при удалении соответствующей строки главной таблицы.

Для обеспечения ссылочной целостности внешнего ключа, когда изменяется или удаляется значение связанного первичного или уникального ключа, могут быть выполнены следующие действия:

- `NO ACTION` (по умолчанию) — не будет выполнено никаких действий;
- `CASCADE` — при изменении или удалении значения первичного ключа над значением внешнего ключа будут произведены те же действия. При выполнении удаления строки в главной таблице в подчинённой таблице должны быть удалены все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) ключа удалённой строки главной таблицы. При выполнении обновления записи главной таблицы в подчинённой таблице должны быть изменены все значения внешнего ключа, имеющие те же значения, что и значение первичного (уникального) ключа изменяемой строки главной таблицы;
- `SET DEFAULT` — значения внешнего ключа всех соответствующих строк в подчинённой таблице устанавливаются в значение по умолчанию, заданное в предложении `DEFAULT` для этого столбца;
- `SET NULL` — значения внешнего ключа всех соответствующих строк в подчинённой
Ограничение **CHECK**

Ограничение **CHECK** задаёт условие, которому должны удовлетворять значения, помещаемые в данный столбец. Условие — это логическое выражение, называемое также предикат, которое может возвращать значения **TRUE** (и́стина), **FALSE** (ложь) и **UNKNOWN** (неизвестно). Условие считается выполненным, если предикат возвращает значение **TRUE** или **UNKNOWN** (эквивалент **NULL**). Если предикат возвращает **FALSE**, то значение не будет принято. Это условие используется при добавлении в таблицу новой строки (оператор **INSERT**) и при изменении существующего значения столбца таблицы (оператор **UPDATE**), а также операторов, в которых может произойти одно из этих действий (**UPDATE OR INSERT, MERGE**).

При использовании предложения **CHECK** для столбца, базирующегося на **домене**, следует помнить, что выражение в **CHECK** лишь дополняет условие проверки, которое может уже быть определено в домене.

На уровне столбца или таблицы выражение в предложении **CHECK** ссылается на входящее значение с помощью с помощью идентификаторов столбцов, в отличие от доменов, где в ограничении **CHECK** для этих целей используется ключевое слово **VALUE**.

**Example 94. CHECK ограничения уровня столбца и уровня таблицы**

```sql
CREATE TABLE PLACES (  
  ...  
  LAT DECIMAL(9, 6) CHECK (ABS(LAT) <= 90),  
  LON DECIMAL(9, 6) CHECK (ABS(LON) <= 180),  
  ...  
  CONSTRAINT CHK_POLES CHECK (ABS(LAT) < 90 OR LON = 0)  
);```

**Привилегии выполнения**

Необязательное предложение **SQL SECURITY** в спецификации таблицы позволяет задать с какими привилегиями вычисляются вычисляемые столбцы. Если выбрана опция **INVOKER**, то вычисляемые столбцы вычисляются с привилегиями вызывающего пользователя. Если выбрана опция **DEFINER**, то вычисляемые столбцы вычисляются с привилегиями определяющего пользователя (владельца). По умолчанию вычисляемые столбцы вычисляются с привилегиями вызывающего пользователя. Кроме триггеров наследуют
Привилегии выполнения таблицы, если они не переопределены у самих триггеров.

Привилегии выполнения по умолчанию для вновь создаваемых объектов метаданных можно изменить с помощью оператора

```
ALTER DATABASE SET DEFAULT SQL SECURITY {DEFINER | INVOKER}
```

**Управление репликацией**

Необязательное предложение `ENABLE PUBLICATION` включает таблицу в набор репликации (публикацию). Если ранее был выполнен оператор `ALTER DATABASE ADD ALL TO PUBLICATION`, то таблица будет включена в публикацию даже если предложение `ENABLE PUBLICATION` не указано.

Необязательное предложение `DISABLE PUBLICATION` исключает таблицу из набора репликации (публикации). Это предложение имеет смысл указывать только если ранее был выполнен оператор `ALTER DATABASE ADD ALL TO PUBLICATION`, который автоматически добавляет вновь созданные таблицы в публикацию.

**Кто может создать таблицу?**

Выполнить оператор `CREATE TABLE` могут:

- Администраторы
- Пользователи с привилегией `CREATE TABLE`.

Пользователь, создавший таблицу, становится её владельцем.

**Примеры**

*Example 95. Создание таблицы*

```
CREATE TABLE COUNTRY (
    COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
    CURRENCY VARCHAR(10) NOT NULL);
```

*Example 96. Создание таблицы с заданием именованного первичного и уникального ключей*

```
CREATE TABLE STOCK (
    MODEL SMALLINT NOT NULL CONSTRAINT PK_STOCK PRIMARY KEY,
    MODELNAME CHAR(10) NOT NULL,
    ITEMID INTEGER NOT NULL,
    CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID));
```
Example 97. Создание таблицы с добавлением её в набор репликации

```sql
CREATE TABLE STOCK (  
    MODEL SMALLINT NOT NULL CONSTRAINT PK_STOCK PRIMARY KEY,
    MODELNAME CHAR(10) NOT NULL,
    ITEMID INTEGER NOT NULL,
    CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID))  
ENABLE PUBLICATION;
```

Example 98. Таблица с полем массивом

```sql
CREATE TABLE JOB (  
    JOB_CODE JOBCODE NOT NULL,
    JOB_GRADE JOBGRADE NOT NULL,
    JOB_COUNTRY COUNTRYNAME,
    JOB_TITLE VARCHAR(25) NOT NULL,
    MIN_SALARY NUMERIC(18, 2) DEFAULT 0 NOT NULL,
    MAX_SALARY NUMERIC(18, 2) NOT NULL,
    JOB_REQUIREMENT BLOB SUB_TYPE 1,
    LANGUAGE_REQ VARCHAR(15) [1:5],
    PRIMARY KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY),
    FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
    CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY)
);
```

Example 99. Создание таблицы с ограничением первичного, внешнего и уникального ключа для которых заданы пользовательские имена индексов

```sql
CREATE TABLE PROJECT (  
    PROJ_ID PROJNO NOT NULL,
    PROJ_NAME VARCHAR(20) NOT NULL UNIQUE
    USING DESC INDEX IDX_PROJNAME,
    PROJ_DESC BLOB SUB_TYPE 1,
    TEAM_LEADER EMPNO,
    PRODUCT PRODTYPE,
    CONSTRAINT PK_PROJECT PRIMARY KEY (PROJ_ID)
    USING INDEX IDX_PROJ_ID,
    FOREIGN KEY (TEAM_LEADER) REFERENCES EMPLOYEE (EMP_NO)
    USING INDEX IDX_LEADER
);
```
Example 100. Создание таблицы со столбцом идентификации BY DEFAULT

```sql
CREATE TABLE objects (  
id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name VARCHAR(15)
);

INSERT INTO objects (name) VALUES ('Table');
INSERT INTO objects (name) VALUES ('Book');
INSERT INTO objects (id, name) VALUES (10, 'Computer');

SELECT * FROM objects;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Table</td>
</tr>
<tr>
<td>2</td>
<td>Book</td>
</tr>
<tr>
<td>10</td>
<td>Computer</td>
</tr>
</tbody>
</table>

Example 101. Создание таблицы со столбцом идентификации ALWAYS

```sql
CREATE TABLE objects (  
id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(15)
);

INSERT INTO objects (name) VALUES ('Table');
INSERT INTO objects (name) VALUES ('Book');
INSERT INTO objects (id, name) VALUES (DEFAULT, 'Computer');

SELECT * FROM objects;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Table</td>
</tr>
<tr>
<td>2</td>
<td>Book</td>
</tr>
<tr>
<td>3</td>
<td>Computer</td>
</tr>
</tbody>
</table>

Example 102. Создание таблицы со столбцом идентификации с начальным значением равным 10 и приращением равным 2

```sql
CREATE TABLE objects (  
id INTEGER GENERATED BY DEFAULT AS IDENTITY (STER WITH 10 INCREMENT BY 2)
```
Example 103. Создание таблицы с вычисляемыми полями

```
CREATE TABLE SALARY_HISTORY (
    EMP_NO EMPNO NOT NULL,
    CHANGE_DATE TIMESTAMP DEFAULT 'NOW' NOT NULL,
    UPDATER_ID VARCHAR(20) NOT NULL,
    OLD_SALARY SALARY NOT NULL,
    PERCENT_CHANGE DOUBLE PRECISION DEFAULT 0 NOT NULL,
    SALARY_CHANGE GENERATED ALWAYS AS
    (OLD_SALARY * PERCENT_CHANGE / 100),
    NEW_SALARY COMPUTED BY
    (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100)
);
```

Поле SALARY_CHANGE объявлено согласно стандарту SQL::2003, поле NEW_SALARY в классическом стиле объявления вычисляемых полей в Firebird.

```
CREATE TABLE SALARY_HISTORY
(
    EMP_NO EMPNO NOT NULL,
    CHANGE_DATE TIMESTAMP DEFAULT 'NOW' NOT NULL,
    UPDATER_ID VARCHAR(20) NOT NULL,
    OLD_SALARY SALARY NOT NULL,
    PERCENT_CHANGE DOUBLE PRECISION DEFAULT 0 NOT NULL,
    SALARY_CHANGE GENERATED ALWAYS AS
    (OLD_SALARY * PERCENT_CHANGE / 100),
    NEW_SALARY COMPUTED BY
    (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100)
)
SQL SECURITY DEFINER;
```

То же самое, но вычисляемые столбцы вычисляются с правами определяющего пользователя (владельца таблицы). Кроме триггеры наследуют привилегии
выполнения таблицы, если они не переопределены у самих триггеров.

**Глобальные временные таблицы (GTT)**

Глобальные временные таблицы (в дальнейшем сокращённо “GTT”) так же, как и обычные таблицы, являются постоянными метаданными, но данные в них ограничены по времени существования транзакций (значение по умолчанию) или соединения с БД. Каждая транзакция или соединение имеет свой собственный экземпляр GTT с данными, изолированный от всех остальных. Экземпляры создаются только при условии обращения к GTT, и данные в ней удаляются при завершении транзакции или отключении от БД. Метаданные GTT могут быть изменены или удалены с помощью инструкций ALTER TABLE и DROP TABLE.

**Синтаксис**

```
CREATE GLOBAL TEMPORARY TABLE name
  (<column_def> [, {<column_def> | <table_constraint>} ...])
[ON COMMIT {DELETE | PRESERVE} ROWS]
[SQL SECURITY {DEFINER | INVOKER}]
```

Если в операторе создания глобальной временной таблицы указано необязательное предложение ON COMMIT DELETE ROWS, то будет создана GTT транзакционного уровня (по умолчанию). При указании предложения ON COMMIT PRESERVE ROWS — будет создана GTT уровня соединения с базой данных.

Предложение EXTERNAL [FILE] нельзя использовать для глобальной временной таблицы.

Операторы COMMIT RETAINING и ROLLBACK RETAINING сохраняют данные в глобальных временных таблицах объявленных как ON COMMIT DELETE ROWS. В Firebird 2.x была ошибка: COMMIT RETAINING и ROLLBACK RETAINING делали записи не видимыми для текущей транзакции. Для возврата поведения 2.x установить параметр ClearGTTAtRetaining равным 1 в firebird.conf. Этот параметр может быть удалён в Firebird 5.0.

**Ограничения GTT**

GTT обладают всеми атрибутами обычных таблиц (ключи, внешние ключи, индексы и триггеры), но имеют ряд ограничений:

- GTT и обычные таблицы не могут ссылаться друг на друга;
- GTT уровня соединения (“PRESERVE ROWS”) GTT не могут ссылаться на GTT транзакционного уровня (“DELETE ROWS”);
- Ограничения домена не могут ссылаться на любую GTT;
- Уничтожения экземпляра GTT в конце своего жизненного цикла не вызывает срабатывания триггеров до/после удаления.

В существующей базе данных не всегда легко отличить обычную таблицу от
GTT, или GTT транзакционного уровня от GTT уровня соединения.
Используйте следующий запрос для определения типа таблицы:

```
SELECT t.rdb$type_name
FROM rdb$relations r
JOIN rdb$types t ON r.rdb$relation_type = t.rdb$type
WHERE t.rdb$field_name = 'RDB$RELATION_TYPE'
    AND r.rdb$relation_name = 'TABLENAME'
```

Для просмотра информации о типах всех таблиц используйте запрос:

```
SELECT r.rdb$relation_name, t.rdb$type_name
FROM rdb$relations r
JOIN rdb$types t ON r.rdb$relation_type = t.rdb$type
WHERE t.rdb$field_name = 'RDB$RELATION_TYPE'
    AND coalesce (r.rdb$system_flag, 0) = 0
```

Поле RDB$TYPE_NAME будет отображать PERSISTENT для обычной таблицы, VIEW для представления, GLOBAL_TEMPORARY_PREERVE для GTT уровня соединения, и GLOBAL_TEMPORARY_DELETE для GTT уровня транзакции.

Примеры

**Example 104. Создание глобальной временной таблицы уровня соединения**

```
CREATE GLOBAL TEMPORARY TABLE MYCONNGTT (
    ID INTEGER NOT NULL PRIMARY KEY,
    TXT VARCHAR(32),
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ON COMMIT PRESERVE ROWS;
```

**Example 105. Создание глобальной временной таблицы уровня транзакции ссылающейся внешним ключом на глобальную временную таблицу уровня соединения.**

```
CREATE GLOBAL TEMPORARY TABLE MYTXGTT (
    ID INTEGER NOT NULL PRIMARY KEY,
    PARENT_ID INTEGER NOT NULL REFERENCES MYCONNGTT(ID),
    TXT VARCHAR(32),
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP)
```

**Внешние таблицы**

Необязательное предложение EXTERNAL [FILE] указывает, что таблица хранится вне базы данных во внешнем текстовом файле. Столбцы таблицы, хранящейся во внешнем файле,
могут быть любого типа за исключением BLOB и массивов с любым типом данных.

Над таблицей, хранящейся во внешнем файле, допустимы только операции добавления новых строк (INSERT) и выборки (SELECT) данных. Операции же изменения существующих данных (UPDATE) или удаления строк такой таблицы (DELETE) не могут быть выполнены.

Внешняя таблица не может содержать ограничений первичного, внешнего и уникального ключа. Для полей такой таблицы невозможно создать индексы.

Файл с внешней таблицей должен располагаться на устройстве хранения, физически расположенном на сервере, на котором расположена СУБД. Если параметр ExternalFileAccess в файле конфигурации firebird.conf содержит Restrict, то файл внешней таблицы должен находится в одном из каталогов, указанных в качестве аргумента Restrict. Если при обращении к таблице Firebird не находит файла, то он создаёт его при первом обращении.

Возможность использования для таблиц внешних файлов зависит от установки значения параметра ExternalFileAccess в файле конфигурации firebird.conf:

• Если он установлен в значение None, то запрещён любой доступ к внешнему файлу.
• Значение Restrict рекомендуется для ограничения доступа к внешним файлам только каталогами, созданными специально для этой цели администратором сервера. Например:
  ○ ExternalFileAccess = Restrict externalfiles ограничит доступ директорией externalfiles корневого каталога Firebird.
  ○ ExternalFileAccess = Restrict d:\databases\outfiles; e:\infiles ограничит доступ только двумя директориями Windows. Обратите внимание, что любые пути являющиеся отображением сетевых путей не будут работать. Также не будут работать пути заключённые в одинарные или двойные кавычки.
• Значение Full позволяет доступ к внешним файлам в любом месте файловой системы хоста. Это создаёт уязвимость и не рекомендуется к использованию.

**Формат внешних файлов**

Внешняя таблица имеет формат “строк” с фиксированной длинной. Нет никаких разделителей полей: границы полей и строк определяются максимальными размерами в байтах в определении каждого поля. Это необходимо помнить и при определении структуры внешней таблицы, и при проектировании входного файла для внешней таблицы, в которую должны импортироваться данные из другого приложения. Например, широко распространённый формат “.csv”, не может быть использован в качестве входного файла, и не может быть получен непосредственно как внешний файл.

Самым полезным типом данных для столбцов внешних таблиц является тип CHAR с фиксированной длинной, длина должна подходить под данные с которыми необходимо
работать. Числовые типы и даты легко преобразуются в них, строки получаются как есть, в то время как, если данные не читаются другой базой данных Firebird, то родные типы могут быть нераспознаваемыми для внешних приложений и являться для них “абракадаброй”.

Конечно, существуют способы манипулирования типами данных так, чтобы создавать выходные файлы из Firebird, которые могут быть непосредственно прочитаны как входные файлы в других приложениях, используя хранимые процедуры с использованием внешних таблиц или без них. Описания этих методов выходит за рамки данного руководства. Здесь мы приведём лишь некоторые рекомендации и советы для создания и работы с простыми текстовыми файлами, поскольку внешняя таблица часто используется как простой способ для создания или чтения транзакционно-независимого журнала. Эти файлы могут быть прочитаны в офлайн режиме текстовым редактором или приложением аудита.

**Разделитель строк**

Как правило, внешние файлы более удобны если строки разделены разделителем, в виде последовательности "новой строки", которая может быть распознана приложением на предназначенной платформе. Для Windows — это двухбайтная 'CRLF' последовательность, возврат каретки (ASCII код 13) и перевод строки (ASCII код 10). Для POSIX — LF обычно самодостаточен, в некоторых MacOS X приложениях она может быть LFCR. Существуют различные способы для автоматического заполнения столбца разделителя. В нашем примере это сделано с помощью BEFORE INSERT триггера и встроенной функции ASCII_CHAR.

**Примеры использования внешних таблиц**

В нашем примере мы будем определять внешнюю таблицу журнала, которая может быть использована в обработчике исключений внутри хранимой процедуры или триггера. Внешняя таблица выбрана потому, что сообщения из любых обрабатываемых исключений будут сохранены в журнале, даже если транзакция, в которой был запущен процесс, будет откана из-за другого необработанного исключения. В целях демонстрации наша таблица содержит всего два столбца: метку времени и текстовое сообщение. Третий столбец хранит разделитель строки:

```sql
CREATE TABLE ext_log
EXTERNAL FILE 'd:\externals\log_me.txt' (  
    stamp CHAR(24),  
    message CHAR(100),  
    crlf CHAR(2) -- Для Windows  
);

COMMIT;
```

Теперь создадим триггер, для автоматического сохранения метки времени и разделителя строки, каждый раз когда сообщение записывается в таблицу:

```sql
SET TERM ^;
CREATE TRIGGER bi_ext_log FOR ext_log
ACTIVE BEFORE INSERT
```

**Chapter 5. Операторы определения данных (DDL)**

174
AS
BEGIN
    IF (NEW.stamp IS NULL) THEN
        NEW.stamp = CAST (CURRENT_TIMESTAMP AS CHAR(24));
        NEW.crlf = ASCII_CHAR(13) || ASCII_CHAR(10);
    END ^
COMMIT ^
SET TERM ;^

Вставка некоторых записей (это может быть сделано в обработчике исключения):

```
INSERT INTO ext_log (message)
VALUES ('Shall I compare thee to a summer''s day?');
INSERT INTO ext_log (message)
VALUES ('Thou art more lovely and more temperate');
```

Содержимое внешнего файла:

```
2015-10-07 15:19:03.4110Shall I compare thee to a summer's day?
2015-10-07 15:19:58.7600Thou art more lovely and more temperate
```

См. также:
ALTER TABLE, DROP TABLE, CREATE DOMAIN.

5.4.2. ALTER TABLE

Назначение
Изменение структуры таблицы.

Доступно в
DSQL, ESQL.

Синтаксис

```
ALTER TABLE tablename
    <operation> [, <operation>];

<operation> ::= 
    ADD <col_def>
    | ADD <tconstraint>
    | DROP colname
    | DROP CONSTRAINT constr_name
    | DROP SQL SECURITY
    | ALTER [COLUMN] colname <col_mod>
    | ALTER SQL SECURITY {DEFINER | INVOKER}
    | {ENABLE | DISABLE} PUBLICATION
```
<col_def> ::=  
  <regular_col_def>
  | <computed_col_def>
  | <identity_col_def>

<regular_col_def> ::=  
  colname { <datatype> | domainname }  
  [DEFAULT {literal | NULL | <context_var}>]
  [NOT NULL]
  [<col_constraint>]
  [COLLATE collation_name]

<computed_col_def> ::=  
  colname [<datatype>]
  {COMPUTED [BY] | GENERATED ALWAYS AS} (<expression>)

<identity_col_def> ::=  
  colname [<datatype>] {ALWAYS | GENERATED BY} DEFAULT AS IDENTITY
  [(START WITH startvalue)] [<col_constraint>]

<col_mod> ::=  
  TO newname
  | POSITION newpos
  | <regular_col_mod>
  | <computed_col_mod>
  | <identity_col_mod>

<regular_col_mod> ::=  
  | TYPE { <datatype> | domain_name }  
  | SET DEFAULT {literal | NULL | <context_var>}
  | DROP DEFAULT
  | SET NOT NULL
  | DROP NOT NULL

<computed_col_mod> ::=  
  [TYPE <datatype>] {GENERATED ALWAYS AS | COMPUTED [BY]} (<expression>)

<identity_col_mod> ::=  
  <alter identity column option> ...
  | SET GENERATED { ALWAYS | BY DEFAULT } [<alter identity column option> ...]
  | DROP INDENTITY

<alter identity column option> ::=  
  RESTART [ WITH startvalue ]
  | SET INCREMENT [BY] incrementvalue

<datatype> ::=  
  <scalar_datatype> | <blob_datatype> | <array_datatype>

<scalar_datatype> ::=  См. Синтаксис скалярных типов данных
См. Синтаксис типа данных BLOB

См. Синтаксис массивов

[CONSTRAINT constr_name]
{  UNIQUE [<using_index>]
   | PRIMARY KEY [<using_index>]
   | REFERENCES other_table [(other_col)]
   [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
   [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
   [<using_index>]
   | CHECK (<check_condition>)
}

[CONSTRAINT constr_name]
{  UNIQUE (<col_list>) [<using_index>]
   | PRIMARY KEY (<col_list>) [<using_index>]
   | FOREIGN KEY (<col_list>)
   REFERENCES other_table [(<col_list>)]
   [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
   [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
   [<using_index>]
   | CHECK (<check_condition>)
}

<col_list> ::= colname [, colname ...]

<using_index> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX indexname

<check_condition> ::= <val> <operator> <val>
| <val> [NOT] BETWEEN <val> AND <val>
| <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
| <val> IS [NOT] NULL
| <val> IS [NOT] DISTINCT <val>
| <val> IS [NOT] {TRUE | FALSE | UNKNOWN}
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| <val> [NOT] LIKE <val> [ESCAPE <val>]
| <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
| <val> [operator] {ALL | SOME | ANY} (<select_list>)
| [NOT] EXISTS (<select_expr>)
| [NOT] SINGULAR (<select_expr>)
| (<check_condition>)
| NOT <check_condition>
| <check_condition> OR <check_condition>
| <check_condition> AND <check_condition>
Параметры оператора `ALTER TABLE`

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablename</code></td>
<td>Имя таблицы.</td>
</tr>
<tr>
<td><code>operation</code></td>
<td>Одна из допустимых операций по изменению структуры таблицы.</td>
</tr>
<tr>
<td><code>colname</code></td>
<td>Имя столбца таблицы, может содержать до 63 символов. Должно быть уникальным внутри таблицы.</td>
</tr>
<tr>
<td><code>newname</code></td>
<td>Новое имя столбца таблицы, может содержать до 63 символов. Должно быть уникальным внутри таблицы.</td>
</tr>
<tr>
<td><code>gencolname</code></td>
<td>Имя вычисляемого столбца таблицы.</td>
</tr>
<tr>
<td><code>idencolname</code></td>
<td>Имя столбца идентификации.</td>
</tr>
<tr>
<td><code>newpos</code></td>
<td>Новая позиция столбца в таблице. Целое число в диапазоне от 1 до количества столбцов таблицы.</td>
</tr>
<tr>
<td><code>datatype</code></td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td><code>domain_name</code></td>
<td>Имя домена.</td>
</tr>
<tr>
<td><code>startvalue</code></td>
<td>Начальное значение столбца идентификации.</td>
</tr>
<tr>
<td><code>incrementvalue</code></td>
<td>Значение приращения для столбца идентификации. Должно быть отлично от 0.</td>
</tr>
<tr>
<td><code>col_constraint</code></td>
<td>Ограничение столбца.</td>
</tr>
<tr>
<td><code>tconstraint</code></td>
<td>Ограничение таблицы.</td>
</tr>
<tr>
<td><code>constr_name</code></td>
<td>Имя ограничения, может содержать до 63 символов.</td>
</tr>
</tbody>
</table>
### Параметр
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>other_table</td>
<td>Имя таблицы, на которую ссылается внешний ключ.</td>
</tr>
<tr>
<td>other_col</td>
<td>Столбец таблицы, на которую ссылается внешний ключ.</td>
</tr>
<tr>
<td>using_index</td>
<td>Позволяет задать имя автоматически создаваемого индекса для ограничения, и опционально определить, какой это будет индекс — по возрастанию (по умолчанию) или по убыванию.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом данных столбца.</td>
</tr>
<tr>
<td>check_condition</td>
<td>Условие проверки ограничения. Выполняется, если оценивается как TRUE или NULL/UNKNOWN.</td>
</tr>
<tr>
<td>collation_name</td>
<td>Имя порядка сортировки. Необходимо указывать если вы хотите чтобы порядок сортировки столбца отличался от порядка сортировки для набора символов по умолчанию этого столбца.</td>
</tr>
<tr>
<td>select_one</td>
<td>Оператор SELECT выбирающий один столбец и возвращающий только одну строку.</td>
</tr>
<tr>
<td>select_list</td>
<td>Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.</td>
</tr>
<tr>
<td>select_expr</td>
<td>Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.</td>
</tr>
<tr>
<td>expression</td>
<td>Выражение.</td>
</tr>
<tr>
<td>genname</td>
<td>Имя последовательности (генератора).</td>
</tr>
<tr>
<td>func</td>
<td>Скалярная функция.</td>
</tr>
</tbody>
</table>

Оператор ALTER TABLE изменяет структуру существующей таблицы. Одиночный оператор ALTER TABLE позволяет производить множество операций добавления/удаления столбцов и ограничений, а также модификаций столбцов. Список операций выполняемых при модификации таблицы разделяется запятой.

#### Счётчик форматов

Некоторые изменения структуры таблицы увеличивают счётчик форматов, закреплённый за каждой таблицей. Количество форматов для каждой таблицы ограничено значением 255. После того как счётчик форматов достигнет этого значения, вы не сможете больше менять структуру таблицы.

### Сброс счётчика форматов

Для сброса счётчика форматов необходимо сделать резервное копирование и восстановление базы данных (утилитой gbak).
Предложение **ADD**

Предложение **ADD** позволяет добавить новый столбец или новое ограничение таблицы. Синтаксис определения столбца и синтаксис описания ограничения таблицы полностью совпадают с синтаксисом, описанным в операторе **CREATE TABLE**.

**Воздействие на счётчик форматов:**

- При каждом добавлении нового столбца номер формата увеличивается на единицу.
- Добавление нового ограничения таблицы не влечёт за собой увеличение номера формата.

**Example 106. Добавление столбца в таблицу**

```sql
ALTER TABLE COUNTRY
ADD CAPITAL VARCHAR(25);
```

**Example 107. Добавление столбца с ограничением NOT NULL**

```sql
ALTER TABLE OBJECTS
ADD QUANTITY INT DEFAULT 1 NOT NULL;
```

Обратите внимание на предложение **DEFAULT**, которое обязательно при добавлении ограничения NOT NULL, если в таблице есть данные. Дело в том, что в этом случае также происходит проверка данных на допустимость. А поскольку при добавлении нового столбца, он для всех строк таблицы содержит значение NULL, будет сгенерировано исключение.

**Example 108. Добавление столбца с ограничением уникальности и удаление другого столбца**

```sql
ALTER TABLE COUNTRY
ADD CAPITAL VARCHAR(25) UNIQUE,
DROP CURRENCY;
```

Для добавления ограничений уровня таблицы необходимо использовать предложение **ADD [CONSTRAINT]**.

**Example 109. Добавление проверочного ограничения и внешнего ключа**

```sql
ALTER TABLE JOB
ADD CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY),
ADD FOREIGN KEY (JOB_COUNTRY)
REFERENCES COUNTRY (COUNTRY);
```
Будьте осторожны, при добавлении нового ограничения CHECK не осуществляется проверка соответствия ему ранее внесённых данных. Поэтому перед добавлением такого ограничения рекомендуем производить предварительную проверку данных в таблице.

Предложение DROP

Предложение DROP удаляет указанный столбец таблицы. Столбец таблицы не может быть удалён, если от него существуют зависимости. Другими словами для успешного удаления столбца на него должны отсутствовать ссылки. Ссылки на столбец могут содержаться:

- в ограничениях столбцов или таблицы;
- в индексах;
- в хранимых процедурах и триггерах;
- в представлениях.

При каждом удалении столбца номер формата увеличивается на единицу.

Предложение DROP CONSTRAINT

Предложение DROP CONSTRAINT удаляет указанное ограничение столбца или таблицы. Ограничение первичного ключа или уникального ключа не могут быть удалены, если они используются в ограничении внешнего ключа другой таблицы. В этом случае необходимо удалить ограничение FOREIGN KEY до удаления PRIMARY KEY или UNIQUE ключа, на которые оно ссылается.

Удаление ограничения столбца или ограничения таблицы не влечёт за собой увеличение номера формата.

Предложение DROP SQL SECURITY

Предложение DROP SQL SECURITY удаляет привилегии выполнения для таблицы. После удаления привилегий выполнения вычисляемые столбцы таблицы будут вычисляться с привилегиями вызывающего пользователя. Триггеры также будут выполняться с привилегиями вызывающего пользователя, если их привилегии выполнения не переопределены в триггере явно.

Предложение ALTER [COLUMN]

Предложение ALTER [COLUMN] позволяет изменить следующие характеристики существующих столбцов:

- изменение имени (не изменяет номер формата);
- изменение типа данных (увеличивает номер формата на единицу);
- изменение позиции столбца в списке столбцов таблицы (не изменяет номер формата);
- удаление значения по умолчанию столбца (не изменяет номер формата);
- добавление значения по умолчанию столбца (не изменяет номер формата);
• изменение типа и выражения для вычисляемого столбца (не изменяет номер формата);
• добавление ограничения NOT NULL (не изменяет номера формата);
• удаление ограничения NOT NULL (не изменяет номера формата).

Приеменование столбца

Ключевое слово TO переименовывает существующий столбец. Новое имя столбца не должно присутствовать в таблице.

Невозможно изменение имени столбца, если этот столбец включён в какое-либо ограничение — первичный или уникальный ключ, внешний ключ, ограничение столбца или проверочное ограничение таблицы CHECK. Имя столбца также нельзя изменить, если этот столбец таблицы используется в каком-либо триггере, в хранимой процедуре или представлении.

*Example 110. Переименование столбца таблицы*

```sql
ALTER TABLE STOCK
ALTER COLUMN MODELNAME TO NAME;
```

Изменение типа столбца

Ключевое слово TYPE изменяет тип существующего столбца на другой допустимый тип. Не допустимы любые изменения типа, которые могут привести к потере данных. Например, количество символов в новом типе для столбца не может быть меньше, чем было установлено ранее.

Если столбец был объявлен как массив, то изменить ни его тип, ни размерность нельзя.

Нельзя изменить тип данных у столбца, который принимает участие в связке внешний ключ/первичный (unikальный) ключ.

*Example 111. Изменение типа столбца таблицы*

```sql
ALTER TABLE STOCK
ALTER COLUMN ITEMID TYPE BIGINT;
```

Изменение позиции столбца

Ключевое слово POSITION изменяет позицию существующего столбца. Позиции столбцов нумеруются с единицы.

• Если будет задан номер позиции меньше 1, то будет выдано соответствующее сообщение об ошибке.
• Если будет задан номер позиции, превышающий количество столбцов в таблице, то
изменения не будут выполнены, но ни ошибки, ни предупреждения не последуют.

*Example 112. Изменение позиции столбца таблицы*

```
ALTER TABLE STOCK
ALTER COLUMN ITEMID POSITION 5;
```

**Установка и удаление значения по умолчанию**

Предложение DROP DEFAULT удаляет значение по умолчанию для столбца таблицы.

- Если столбец основан на домене со значением по умолчанию — доменное значение перекроет это удаление.
- Если удаление значения по умолчанию производится над столбцом, у которого нет значения по умолчанию, или чьё значение по умолчанию основано на домене, то это приведёт к ошибке выполнения данного оператора.

*Example 113. Удаление значения по умолчанию для столбца*

```
ALTER TABLE STOCK
ALTER COLUMN MODEL DROP DEFAULT;
```

Предложение SET DEFAULT устанавливает значение по умолчанию для столбца таблицы. Если столбец уже имел значение по умолчанию, то оно будет заменено новым. Значение по умолчанию для столбца всегда перекрывает доменное значение по умолчанию.

*Example 114. Установка значения по умолчанию для столбца*

```
ALTER TABLE STOCK
ALTER COLUMN MODEL SET DEFAULT 1;
```

**Установка и удаление ограничения NOT NULL**

Предложение SET NOT NULL добавляет ограничение NOT NULL для столбца таблицы.

Успешное добавление ограничения NOT NULL происходит, только после полной проверки данных таблицы, для того чтобы убедиться, что столбец не содержит значений NULL.

Явное ограничение NOT NULL на столбце, базирующегося на домене, преобладает над установками домена. В этом случае изменение домена для допустимости значения NULL, не распространяется на столбец таблицы.
**Example 115. Добавление ограничения NOT NULL**

```sql
ALTER TABLE STOCK
ALTER COLUMN PROPID SET NOT NULL;
```

Предложение DROP NOT NULL удаляет ограничение NOT NULL для столбца таблицы. Если столбец основан на домене с ограничением NOT NULL, то ограничение домена перекроет это удаление.

**Example 116. Удаление ограничения NOT NULL**

```sql
ALTER TABLE STOCK
ALTER COLUMN ITEMID DROP NOT NULL;
```

**Изменение столбцов идентификации**

Для столбцов идентификации позволено изменять способ генерации, начальное значение и значение приращения.

Предложение SET GENERATED позволяет изменить способ генерации столбца идентификации. Существует два способа генерации столбца идентификации:

- **BY DEFAULT** столбцы позволяют переписать сгенерированное системой значение в операторах INSERT, UPDATE OR INSERT, MERGE просто указав значение этого столбца в списке значений.
- **ALWAYS** столбцы не позволяют переписать сгенерированное системой значение, при попытке переписать значение такого столбца идентификации будет выдана ошибка. Переписать значение этого столбца в операторе INSERT можно только при указании директивы OVERRIDING SYSTEM VALUE.

**Example 117. Изменение способа генерации столбца идентификации**

```sql
ALTER TABLE objects
ALTER ID SET GENERATED ALWAYS;
```

Если указано только предложение **RESTART**, то происходит сброс значения генератора в ноль. Необходимое предложение **WITH** позволяет указать для нового значения внутреннего генератора отличное от нуля значение.

**Example 118. Изменение текущего значения генератора для столбца идентификации**

```sql
ALTER TABLE objects
```
Предложение `SET INCREMENT [BY]` позволяет изменить значение приращения столбца идентификации. Значение приращения должно быть отлично от 0.

**Example 119. Изменение приращения столбца идентификации**
```
ALTER TABLE objects
ALTER ID SET INCREMENT BY 2;
```

В одном операторе можно изменить сразу несколько свойств столбца идентификации, например:

**Example 120. Изменение нескольких свойств столбца идентификации**
```
ALTER TABLE objects
ALTER ID SET GENERATED ALWAYS RESTART SET INCREMENT BY 2;
```

Предложение `DROP IDENTITY` удаляет связанный со столбцом идентификации системную последовательность и преобразует его в обычный столбец.

**Example 121. Превращение столбца идентификации в обычный столбец**
```
ALTER TABLE objects
ALTER ID DROP INDENTITY;
```

**Изменение вычисляемых столбцов**

Для вычисляемых столбцов (`GENERATED ALWAYS AS` или `COMPUTED BY`) позволяет изменить тип и выражение вычисляемого столбца. Невозможно изменить обычный столбец на вычисляемый и наоборот.

**Example 122. Изменение вычисляемых столбцов**
```
ALTER TABLE SALARY_HISTORY
ALTER NEW_SALARY GENERATED ALWAYS
AS (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100),
ALTER SALARY_CHANGE COMPUTED
BY (OLD_SALARY * PERCENT_CHANGE / 100);
```
Не изменяемые атрибуты

На данный момент не существует возможности изменить сортировку по умолчанию.

Предложение ALTER SQL SECURITY

Предложение ALTER SQL SECURITY позволяет изменить привилегии с которыми вычисляются вычисляемые столбцы. Если выбрана опция INVOKER, то вычисляемые столбцы вычисляются с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то вычисляемые столбцы вычисляются с привилегиями определяющего пользователя (владельца). По умолчанию вычисляемые столбцы вычисляются с привилегиями вызывающего пользователя. Кроме того триггеры наследуют привилегии выполнения у таблицы, если они не переопределены у самих триггеров.

```
ALTER TABLE COUNTRY
ALTER SQL SECURITY DEFINER;
```

Управление репликацией

Предложение ENABLE PUBLICATION включает таблицу в набор репликации (публикацию). Соответственно предложение DISABLE PUBLICATION исключает таблицу из набора репликации.

Example 123. Добавление таблицы в набор репликации

```
ALTER TABLE COUNTRY
ENABLE PUBLICATION;
```

Кто может изменить таблицу?

Выполнить оператор ALTER TABLE могут:

- Администраторы
- Владелец таблицы;
- Пользователи с привилегией ALTER ANY TABLE.

См. также:
CREATE TABLE, RECREATE TABLE.

5.4.3. DROP TABLE

Назначение
Удаление существующей таблицы.

Доступно в
**Table 35. Параметры оператора DROP TABLE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Имя таблицы.</td>
</tr>
</tbody>
</table>

Оператор `DROP TABLE` удаляет существующую таблицу. Если таблица имеет зависимости, то удаление не будет произведено. При удалении таблицы будут также удалены все триггеры на её события и индексы, построенные для её полей.

**Example 124. Удаление таблицы**

```
DROP TABLE COUNTRY;
```

**Кто может удалить таблицу?**

Выполнить оператор `DROP TABLE` могут:

- Администраторы
- Владелец таблицы;
- Пользователи с привилегией `DROP ANY TABLE`.

*См. также:* 
CREATE TABLE, RECREATE TABLE.

### 5.4.4. RECREATE TABLE

**Назначение**

Создание новой таблицы или пересоздание существующей.

**Доступно в**

DSQL.

**Синтаксис**

```
RECREATE [GLOBAL TEMPORARY] TABLE tablename
    [EXTERNAL [FILE] 'filespec']
    (<col_def> [, <col_def> | <tconstraint> ...])
    [ON COMMIT {DELETE | PRESERVE} ROWS]
    [SQL SECURITY {DEFINER | INVOKER}]
```
Полное описание определений столбцов и ограничений таблицы смотрите в разделе CREATE TABLE.

Создаёт или пересоздаёт таблицу. Если таблица с таким именем уже существует, то оператор RECREATE TABLE попытается удалить её и создать новую. Оператор RECREATE TABLE не выполнится, если существующая таблица имеет зависимости.

Примеры

**Example 125. Создание или пересоздание таблицы**

```sql
RECREATE TABLE COUNTRY (
    COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
    CURRENCY VARCHAR(10) NOT NULL);
```

*См. также:* CREATE TABLE, DROP TABLE.

## 5.5. INDEX

Индекс — это объект базы данных, используемый для более быстрого извлечения данных из таблицы или для ускорения сортировки в запросе. Кроме того, индексы используются для обеспечения ограничений целостности — PRIMARY KEY, FOREIGN KEY, UNIQUE.

В данном разделе описываются вопросы создания индексов, перевода их в активное/неактивное состояние, удаление индексов и сбор статистики (пересчёт селективности) для индексов.

### 5.5.1. CREATE INDEX

**Назначение**

Создание индекса для таблицы.

**Доступно в**

DSQL, ESQL.

```sql
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX indexname ON tablename
{{(col [, col ...]) | COMPUTED BY (<expression>)}};
```

*Table 36. Параметры оператора CREATE INDEX*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexname</td>
<td>Имя индекса. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>tablename</td>
<td>Имя таблицы, для которой строится индекс.</td>
</tr>
</tbody>
</table>
Оператор CREATE INDEX создаёт индекс для таблицы, который может быть использован для ускорения поиска, сортировки и/или группирования. Кроме того, индекс может быть использован при определении ограничений, таких как первичный ключ, внешний ключ или ограничениях уникальности. Индекс может быть построен на столбцах любого типа кроме BLOB и массивов. Имя индекса должно быть уникальным среди всех имен индексов.

### Индексы в ключах
При добавлении ограничений первичного ключа, внешнего ключа или ограничения уникальности будет неявно создан одноименный индекс. Так, например, при выполнении следующего оператора будет неявно создан индекс PK_COUNTRY.

```
ALTER TABLE COUNTRY
ADD CONSTRAINT PK_COUNTRY PRIMARY KEY (ID);
```

### Уникальные индексы
Если при создании индекса указано ключевое слово UNIQUE, то индекс гарантирует уникальность значений ключей. Такой индекс называется уникальным. Уникальный индекс не является ограничением уникальности.

Уникальные индексы не могут содержать дубликаты значений ключей (или дубликаты комбинаций значений ключей в случае составного, многоколоночного или многосегментного индекса). Дубликаты значения NULL допускаются в соответствии со стандартом SQL-99, в том числе и в многосегментном индексе.

### Направление индекса
Все индексы в Firebird являются однонаправленными. Индекс может быть построен в восходящем и нисходящем порядке. Ключевые слова ASC[ENDING] (сокращённо ASC) и DESC[ENDING] используются для указания направленности индекса. По умолчанию создаётся восходящий ASC[ENDING] индекс. Допускается одновременное определение восходящего и нисходящего индекса на одном и том же столбце или наборе ключей.

Убывающий (DESC[ENDING]) индекс может быть полезен при поиске наивысших значений (максимум, последнее и т.д.)

### Вычисляемые индексы или индексы по выражению
При создании индекса вместо одного или нескольких столбцов вы также можете указать одно выражение, используя предложение COMPUTED BY. Такой индекс называется
вычисляемым или индексом по выражению. Вычисляемые индексы используются в запросах, в которых условие в предложениях WHERE, ORDER BY или GROUP BY в точности совпадает с выражением в определении индекса. Выражение в вычисляемом индексе может использовать несколько столбцов таблицы.

Ограничения на индексы

Максимальная длина ключа индекса ограничена 1/4 размера страницы.

Ограничения на длину индексируемой строки

Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. Максимальная длина индексируемой строки зависит от размера страницы и набора символов.

Table 37. Длина индексируемой строки и набор символов

<table>
<thead>
<tr>
<th>Размер страницы</th>
<th>Максимальная длина индексируемой строки для набора символов, байт/символ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4096</td>
<td>1015</td>
</tr>
<tr>
<td>8192</td>
<td>2039</td>
</tr>
<tr>
<td>16384</td>
<td>4087</td>
</tr>
<tr>
<td>32768</td>
<td>8183</td>
</tr>
</tbody>
</table>

Максимальное количество индексов на таблицу

Для каждой таблицы максимально возможное количество индексов ограничено и зависит от размера страницы и количества столбцов в индексе.

Table 38. Число индексов и количество столбцов

<table>
<thead>
<tr>
<th>Размер страницы</th>
<th>Число индексов в зависимости от количества столбцов в индексе</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4096</td>
<td>203</td>
</tr>
<tr>
<td>8192</td>
<td>408</td>
</tr>
<tr>
<td>16384</td>
<td>818</td>
</tr>
<tr>
<td>32768</td>
<td>1637</td>
</tr>
</tbody>
</table>

Кто может создать индекс?

Выполнить оператор CREATE INDEX могут:

- **Администраторы**
- Владелец таблицы, для которой создаётся индекс;
- Пользователи с привилегией ALTER ANY TABLE.
Примеры

Example 126. Создание индекса

CREATE INDEX IDX_UPDATER ON SALARY_HISTORY (UPDATER_ID);

Example 127. Создание индекса с сортировкой ключей по убыванию

CREATE DESCENDING INDEX IDX_CHANGE ON SALARY_HISTORY (CHANGE_DATE);

Example 128. Создание многосегментного индекса

CREATE INDEX IDX_SALESTAT ON SALES (ORDER_STATUS, PAID);

Example 129. Создание индекса, не допускающего дубликаты значений

CREATE UNIQUE INDEX UNQ_COUNTRY_NAME ON COUNTRY (NAME);

Example 130. Создание вычисляемого индекса

CREATE INDEX IDX_NAME_UPPER ON PERSONS
    COMPUTED BY (UPPER (NAME));

Такой индекс может быть использован для не чувствительного к регистру поиска.

SELECT *
FROM PERSONS
WHERE UPPER(NAME) STARTING WITH UPPER('Iv');

См. также:
ALTER INDEX, DROP INDEX.

5.5.2. ALTER INDEX

Назначение
Перевод индекса в активное/неактивное состояние, перестройка индекса.
Доступно в DSQL, ESQL.

Синтаксис

```
ALTER INDEX indexname {ACTIVE | INACTIVE};
```

Table 39. Параметры оператора ALTER INDEX

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexname</td>
<td>Имя индекса.</td>
</tr>
</tbody>
</table>

Оператор ALTER INDEX переводит индекс в активное/неактивное состояние. Возможность изменения структуры и порядка сортировки ключей этот оператор не предусматривает.

**INACTIVE**

При выборе опции INACTIVE, индекс переводится из активного в неактивное состояние. Перевод индекса в неактивное состояние по своему действию похоже на команду DROP INDEX за исключением того, что определение индекса сохраняется в базе данных. Невозможно перевести в неактивное состояние индекс участвующий в ограничении.

Активный индекс может быть отключен, только если отсутствуют запросы использующие этот индекс, иначе будет возвращена ошибка “object in use”.

Активация неактивного индекса также безопасна. Тем не менее, если есть активные транзакции, модифицирующие таблицу, то транзакция, содержащая оператор ALTER INDEX потерпит неудачу, если она имеет атрибут NO WAIT. Если транзакция находится в режиме WAIT, то она будет ждать завершения параллельных транзакций.

С другой стороны, если наш оператор ALTER INDEX начинает перестраивать индекс на COMMIT, то другие транзакции, изменяющие эту таблицу, потеряют неудачу или будут ожидать в соответствии с их WAIT/NO WAIT атрибутами. Та же самая ситуация будет и при выполнении CREATE INDEX.

Перевод индекса в неактивное состояние может быть полезен при массовой вставке, модификации или удалении записей из таблицы, для которой этот индекс построен.

**ACTIVE**

При выборе альтернативы ACTIVE индекс переводится из неактивного состояния в активное. При переводе индекса из неактивного состояния в активное—индекс перестраивается.

Даже если индекс находится в активном состоянии оператор ALTER INDEX ...

... ACTIVE всѐ равно перестраивает индекс. Таким образом, эту команду можно использовать как часть обслуживания БД для перестройки индексов, автоматически созданных для ограничений PRIMARY KEY, FOREIGN KEY, UNIQUE, для которых выполнение оператора ALTER INDEX ...

... INACTIVE
Использование ALTER INDEX для индексов ограничений

Принудительный перевод индексов созданных для ограничений PRIMARY KEY, FOREIGN KEY и UNIQUE не допускается. Тем не менее выполнение оператора ALTER INDEX ... INACTIVE работает так же хорошо для индексов ограничений, как и другие инструменты для других индексов.

Кто может выполнить ALTER INDEX?

Выполнить оператор ALTER INDEX могут:

- Администраторы
- Владелец таблицы, для которой построен индекс;
- Пользователи с привилегией ALTER ANY TABLE.

Примеры

Example 131. Перевод индекса в неактивное состояние

```sql
ALTER INDEX IDX_UPDATER INACTIVE;
```

Example 132. Возврат индекса в активное состояние

```sql
ALTER INDEX IDX_UPDATER ACTIVE;
```

См. также:

CREATE INDEX, DROP INDEX.

5.5.3. DROP INDEX

Назначение

Удаление индекса из базы данных.

Доступно в

DSQL, ESQL.

Синтаксис

```sql
DROP INDEX indexname
```

Table 40. Параметры оператора DROP INDEX
Оператор DROP INDEX удаляет существующий индекс из базы данных. При наличии зависимостей для существующего индекса (если он используется в ограничении) удаление не будет выполнено.

**Кто может удалить индекс?**

Выполнить оператор DROP INDEX могут:

- Администраторы
- Владелец таблицы, для которой построен индекс;
- Пользователи с привилегией ALTER ANY TABLE.

**Примеры**

*Example 133. Удаление индекса*

```sql
DROP INDEX IDX_UPDATER;
```

См. также: CREATE INDEX, ALTER INDEX.

### 5.5.4. SET STATISTICS

**Назначение**

Пересчёт селективности индекса.

**Доступно в**

DSQL, ESQL.

**Синтаксис**

```sql
SET STATISTICS INDEX indexname
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexname</td>
<td>Имя индекса.</td>
</tr>
</tbody>
</table>

Оператор SET STATISTICS пересчитывает значение селективности для указанного индекса.

**Селективность индекса**

Селективность (избирательность) индекса — это оценочное количество строк, которые
могут быть выбраны при поиске по каждому значению индекса. Уникальный индекс имеет максимальную селективность, поскольку при его использовании невозможно выбрать более одной строки для каждого значения ключа индекса. Актуальность селективности индекса важна для выбора наиболее оптимального плана выполнения запросов оптимизатором.

Пересчёт селективности индекса может потребоваться после массовой вставки, модификации или удалении большого количества записей из таблицы, поскольку она становится неактуальной.

Отметим, что в Firebird статистика индексов автоматически не пересчитывается ни после массовых изменений данных, ни при каких либо других условиях. При создании (CREATE) или его активации (ALTER INDEX ACTIVE) статистика индекса полностью соответствует его содержимому.

Пересчёт селективности индекса может быть выполнен под высоко параллельной нагрузкой без риска его повреждения. Тем не менее следует помнить, что при высоком параллелизме рассчитанная статистика может устареть, как только закончится выполнение оператора SET STATISTICS.

Кто может обновить статистику?

Выполнить оператор SET STATISTICS могут:

• Администраторы
• Владелец таблицы, для которой построен индекс;
• Пользователи с привилегией ALTER ANY TABLE.

Примеры

Example 134. Пересчёт селективности индекса IDX_UPDATER

```
SET STATISTICS INDEX IDX_UPDATER;
```

См. также:
CREATE INDEX, ALTER INDEX.

5.6. VIEW

Представление (view)—виртуальная таблица, которая по своей сути является именованным запросом SELECT выборки данных произвольной сложности. Выборка данных может осуществляться из одной и более таблиц, других представлений, а также селективных хранимых процедур.

В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельным набором данных, хранящимся в базе данных. Результат в виде набора
данных динамически создается при обращении к представлению.

Метаданные представлений доступны для генерации двоичного кода хранимых процедур, функций, пакетов и триггеров так, как будто они являются обычной таблицей, хранящей постоянные данные.

**5.6.1. CREATE VIEW**

**Назначение**
Создание нового представления.

**Доступно в**
DSQL

**Синтаксис**

```
CREATE VIEW viewname [<full_column_list>] AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

**Table 42. Параметры оператора CREATE VIEW**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Имя представления. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>select_statement</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>full_column_list</td>
<td>Список столбцов представления.</td>
</tr>
<tr>
<td>colname</td>
<td>Имя столбца представления. Дубликаты имён столбцов не позволяются.</td>
</tr>
</tbody>
</table>

Оператор CREATE VIEW создаёт новое представление. Имя представления должно быть уникальным среди имён всех представлений, таблиц и хранимых процедур базы данных.

После имени создаваемого представления может идти список имён столбцов, получаемых в результате обращения к представлению. Имена в списке могут быть никак не связаны с именами столбцов базовых таблиц. При этом их количество должно точно соответствовать количеству столбцов в списке выбора главного оператора SELECT представления.

Если список столбцов представления отсутствует, то будут использоваться имена столбцов базовых таблиц или псевдонимов (алиасов) полей оператора SELECT. Если имена полей повторяются или присутствуют выражения столбцов без псевдонимов, которые делают невозможным получение допустимого списка столбцов, то создание представления завершается ошибкой.

Количество столбцов в списке столбцов представления должно совпадать с количеством столбцов указанном в списке выбора оператора SELECT указанного в определении представления.
Дополнительные моменты

• Если указан полный список столбцов, то задание псевдонимов в операторе `SELECT` не имеет смысла, поскольку они будут переопределены именами из списка столбцов;

• Список столбцов необязателен при условии, что все столбцы в операторе `SELECT` имеют явное имя, и эти имена будут уникальными в списке столбцов.

Обновляемые представления

Представление может быть обновляемым и только для чтения. Если представление обновляемое, то данные, полученные при обращении к такому представлению, можно изменить при помощи DML операторов `INSERT`, `UPDATE`, `DELETE`, `UPDATE OR INSERT`, `MERGE`. Изменения, выполняемые над представлением, применяются к базовой таблице(ам).

Представление только для чтения можно сделать обновляемым при помощи вспомогательных триггеров. После того как на представлении будет определён один или несколько триггеров, то изменения не будут автоматически попадать в базовую таблицу, даже если перед этим представление было обновляемым. В этом случае ответственность за обновление (удаление или вставку) записей базовых таблиц, лежит на программисте, определяющем триггеры.

Для того чтобы представление было обновляемым, необходимо выполнение следующих условий:

• оператор выборки `SELECT` обращается только к одной таблице или одному изменяемому представлению;

• оператор выборки `SELECT` не должен обращаться к хранимым процедурам;

• все столбцы базовой таблицы или обновляемого представления, которые не присутствуют в данном представлении, должны удовлетворять одному из следующих условий:
  ◦ позволять значение NULL
  ◦ `NOT NULL` столбцы должны иметь значение по умолчанию
  ◦ значение `NOT NULL` столбцов должны быть инициализированы в триггерах базовых таблиц

• оператор выборки `SELECT` не содержит полей определённых через подзапросы или другие выражения;

• оператор выборки `SELECT` не содержит полей определённых через агрегатные функции (`MIN`, `MAX`, `AVG`, `COUNT`, `LIST`), статистические функции (`CORR`, `COVAR_POP`, `COVAR_SAMP` и др.), функции линейной регрессии (`REGR_AVGX`, `REGR_AVGY` и др.) и все виды оконных функций;

• оператор выборки `SELECT` не содержит предложений `ORDER BY`, `GROUP BY`, `HAVING`;

• оператор выборки `SELECT` не содержит ключевого слова `DISTINCT` и ограничений количества строк с помощью `ROWS`, `FIRST/SKIP`, `OFFSET/FETCH.`
WITH CHECK OPTIONS

Необязательное предложение WITH CHECK OPTIONS задаёт для изменяемого представления требования проверки вновь введённых или модифицируемых данных условию, указанному в предложении WHERE оператора выборки SELECT. При попытке вставки новой записи или модификации записи проверяется, выполняется ли для этой записи условие в предложении WHERE, если условие не выполняется, то вставка/модификация не выполняется и будет выдано соответствующее диагностическое сообщение.

Предложение WITH CHECK OPTION может задаваться в операторе создания представления только в том случае, если в главном операторе SELECT представления указано предложение WHERE. Иначе будет выдано сообщение об ошибке.

Если используется предложение WITH CHECK OPTIONS, то система проверяет входные значения на соответствие условию в предложении WHERE до того как они будут переданы в базовую таблицу. Таким образом, если входные значения не проходят проверку, то предложения DEFAULT или триггеры на базовой таблице, не могут исправить входные значения, поскольку действия никогда не будут выполнены.

Кроме того, поля представления не указанные в операторе INSERT передаются в базовую таблицу как значения NULL, независимо от их наличия или отсутствия в предложении WHERE. В результате значения по умолчанию, определённые на таких полях базовой таблицы, не будут применены. С другой стороны, триггеры будут вызываться и работать как ожидалось.

Для представлений у которых отсутствует предложение WITH CHECK OPTIONS, поля, отсутствующие в операторе INSERT, не передаются вовсе, поэтому любые значения по умолчанию будут применены.

Привилегии выполнения

Выполнение SQL кода представлений всегда осуществляется с привилегиями определяющего пользователя (владельца).

Кто может создать представление?

Выполнить оператор CREATE VIEW могут:

- Администраторы
- Пользователи с привилегией CREATE VIEW.

Пользователь, создавший представление, становится его владельцем.

Для создания представления пользователями, которые не имеют административных привилегий, необходимы также привилегии на чтение (SELECT) данных из базовых таблиц и представлений, и привилегии на выполнение (EXECUTE) используемых селективных хранимых процедур.

Для разрешения вставки, обновления и удаления через представление, необходимо чтобы
создатель (владелец) представления имел привилегии INSERT, UPDATE и DELETE на базовые объекты метаданных.

Предоставить привилегии на представление другим пользователям возможно только если владелец представления сам имеет эти привилегии на базовых объектах. Она будет всегда, если владелец представления является владельцем базовых объектов метаданных.

Примеры

Example 135. Создание представления

```sql
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE
FROM JOB
WHERE MAX_SALARY < 15000;
```

Example 136. Создание представления с проверкой условия фильтрации

Создание представления возвращающего столбцы JOB_CODE и JOB_TITLE только для тех работ, где MAX_SALARY меньше $15000. При вставке новой записи или изменении существующей будет осуществляться проверка условия MAX_SALARY < 15000, если условие не выполняется, то вставка/изменение будет отвергнуто.

```sql
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE
FROM JOB
WHERE MAX_SALARY < 15000
WITH CHECK OPTIONS;
```

Example 137. Создание представления с использованием списка столбцов

```sql
CREATE VIEW PRICE_WITH_MARKUP (CODE_PRICE, COST, COST_WITH_MARKUP) AS
SELECT CODE_PRICE, COST, COST * 1.1
FROM PRICE;
```
Example 138. Создание представления с использованием псевдонимов полей

```
CREATE VIEW PRICE_WITH_MARKUP AS
SELECT
    CODE_PRICE,
    COST,
    COST * 1.1 AS COST_WITH_MARKUP
FROM PRICE;
```

Example 139. Создание необновляемого представления с использованием хранимой процедуры

```
CREATE VIEW GOODS_PRICE AS
SELECT
    goods.name AS goodsname,
    price.cost AS cost,
    b.quantity AS quantity
FROM
    goods
    JOIN price ON goods.code_goods = price.code_goods
    LEFT JOIN sp_get_balance(goods.code_goods) b ON 1 = 1;
```

Example 140. Создание обновляемого представления с использованием триггеров

```
-- базовые таблицы
RECREATE TABLE t_films(id INT PRIMARY KEY, title VARCHAR(100));
RECREATE TABLE t_sound(id INT PRIMARY KEY, audio BLOB);
RECREATE TABLE t_video(id INT PRIMARY KEY, video BLOB);
COMMIT;

-- создание необновляемого представления
RECREATE VIEW v_films AS
SELECT f.id, f.title, s.audio, v.video
FROM t_films f
LEFT JOIN t_sound s ON f.id = s.id
LEFT JOIN t_video v ON f.id = v.id;

/* Для того чтобы сделать представление обновляемым создадим
триггер, который будет производить манипуляции над базовыми
таблицами.
*/
SET TERM ^;
CREATE OR ALTER TRIGGER v_films_biud FOR v_films
ACTIVE BEFORE INSERT OR UPDATE OR DELETE POSITION 0 AS
BEGIN
    IF (INSERTING) THEN
        new.id = COALESCE(new.id, GEN_ID(g_films, 1));
```
IF (NOT DELETING) THEN
BEGIN
  UPDATE OR INSERT INTO t_films(id, title)
  VALUES(new.id, new.title)
  MATCHING(id);
  UPDATE OR INSERT INTO t_sound(id, audio)
  VALUES(new.id, new.audio)
  MATCHING(id);
  UPDATE OR INSERT INTO t_video(id, video)
  VALUES(new.id, new.video)
  MATCHING(id);
END
ELSE
BEGIN
  DELETE FROM t_films WHERE id = old.id;
  DELETE FROM t_sound WHERE id = old.id;
  DELETE FROM t_video WHERE id = old.id;
END
END
SET TERM ;^
### Table 43. Параметры оператора ALTER VIEW

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Имя существующего представления.</td>
</tr>
<tr>
<td>select_statement</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>full_column_list</td>
<td>Список столбцов представления.</td>
</tr>
<tr>
<td>colname</td>
<td>Имя столбца представления. Дубликаты имён столбцов не позволяются.</td>
</tr>
</tbody>
</table>

Оператор ALTER VIEW изменяет определение существующего представления, существующие права на представления и зависимости при этом сохраняются. Синтаксис оператора ALTER VIEW полностью аналогичен синтаксису оператора CREATE VIEW.

Будьте осторожны при изменении количества столбцов представления. Существующий код приложения может стать неработоспособным. Кроме того, PSQL модули, использующие изменённое представление, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

### Кто может изменить представление?

Выполнить оператор ALTER VIEW могут:

- Администраторы
- Владелец представления;
- Пользователи с привилегией ALTER ANY VIEW.

### Примеры

**Example 141. Изменение представления**

```sql
ALTER VIEW PRICE_WITH_MARKUP (CODE_PRICE, COST, COST_WITH_MARKUP) AS
SELECT CODE_PRICE, COST, COST * 1.15
FROM PRICE;
```

См. также:
CREATE VIEW, CREATE OR ALTER VIEW, RECREATE VIEW.
5.6.3. CREATE OR ALTER VIEW

Назначение
Создание нового или изменение существующего представления.

Доступно в
DSQL

Синтаксис

```
CREATE OR ALTER VIEW viewname [ <full_column_list> ]
AS <select_statement>
[ WITH CHECK OPTION ];

<full_column_list> ::= ( colname [ , colname ... ] )
```

Table 44. Параметры оператора CREATE OR ALTER VIEW

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Имя представления. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>select_statement</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>full_column_list</td>
<td>Список столбцов представления.</td>
</tr>
<tr>
<td>colname</td>
<td>Имя столбца представления. Дубликаты имен столбцов не позволяются.</td>
</tr>
</tbody>
</table>

Оператор CREATE OR ALTER VIEW создаёт представление, если оно не существует. В противном случае он изменит представление с сохранением существующих зависимостей.

Примеры

Example 142. Создание нового или изменение существующего представления

```
CREATE OR ALTER VIEW PRICE_WITH_MARKUP ( 
    CODE_PRICE, 
    COST, 
    COST_WITH_MARKUP 
) AS 
SELECT 
    CODE_PRICE, 
    COST, 
    COST * 1.15 
FROM PRICE;
```

См. также:
CREATE VIEW, ALTER VIEW, RECREATE VIEW.
5.6.4. DROP VIEW

Назначение
Удаление существующего представления.

Доступно в
DSQL

Синтаксис

DROP VIEW viewname

Table 45. Параметры оператора DROP VIEW

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Имя представления.</td>
</tr>
</tbody>
</table>

Оператор DROP VIEW удаляет существующее представление. Если представление имеет зависимости, то удаление не будет произведено.

Кто может удалить представление?

Выполнить оператор DROP VIEW могут:

- Администраторы
- Владелец представления;
- Пользователи с привилегией DROP ANY VIEW.

Примеры

Example 143. Удаление представления

```sql
DROP VIEW PRICE_WITH_MARKUP;
```

См. также:

CREATE VIEW, RECREATE VIEW.

5.6.5. RECREATE VIEW

Назначение
Создание нового или пересоздание существующего представления.

Доступно в
DSQL
Синтаксис

```sql
CREATE VIEW viewname [full_column_list] AS select_statement [WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Table 46. Параметры оператора CREATE VIEW

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Имя представления. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>select_statement</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>full_column_list</td>
<td>Список столбцов представления.</td>
</tr>
<tr>
<td>colname</td>
<td>Имя столбца представления. Дубликаты имён столбцов не позволяются.</td>
</tr>
</tbody>
</table>

Создаёт или пересоздаёт представление. Если представление с таким именем уже существует, то оператор `RECREATE VIEW` попытается удалить его и создать новое. Оператор `RECREATE VIEW` не выполнится, если существующее представление имеет зависимости.

Примеры

**Example 144. Создание нового или пересоздание существующего представления**

```sql
RECREATE VIEW PRICE_WITH_MARKUP (
    CODE_PRICE,
    COST,
    COST_WITH_MARKUP
) AS
SELECT
    CODE_PRICE,
    COST,
    COST * 1.15
FROM PRICE;
```

См. также:

CREATE VIEW, CREATE OR VIEW, DROP VIEW.

**5.7. TRIGGER**

Триггер (trigger) — это хранимая процедура особого типа, которая не вызывается непосредственно, а исполнение которой обусловлено наступлением одного из событий, относящегося к одной конкретной таблице (представлению), или наступлению одного из событий базы данных.
Триггер — это особый тип хранимой процедуры, которая не вызывается напрямую, а выполняется, когда в связанной таблице или представлении происходит указанное событие. DML трягер специфичен для одного и только одного отношения (таблица или представление) и одной фазы во времени события (ДО или ПОСЛЕ). Его можно задать для выполнения для одного конкретного события (вставка, обновление, удаление) или для некоторой комбинации двух или трех из этих событий.

Помимо DML трягеров существуют также: * Триггеры на события базы данных, которые происходят при начале или завершении соединения, или транзакции. * DDL трягеры, которые срабатывают до или после выполнения одного или нескольких типов DDL операторов.

### 5.7.1. CREATE TRIGGER

#### Назначение
Создание нового трягера.

#### Доступно в
DSQL, ESQL

#### Синтаксис

```plaintext
CREATE TRIGGER trigname {
    <relation_trigger_legacy>
    | <relation_trigger_sql2003>
    | <database_trigger>
    | <ddl_trigger> }
<routine body>

<relation_trigger_legacy> ::= FOR {tablename | viewname}
    [ACTIVE | INACTIVE]
    {BEFORE | AFTER} <mutation_list>
    [POSITION number]

<relation_trigger_sql2003> ::= [ACTIVE | INACTIVE]
    {BEFORE | AFTER} <mutation_list>
    ON {tablename | viewname}
    [POSITION number]

(database_trigger) ::= [ACTIVE | INACTIVE]
    ON db_event
    [POSITION number]

<ddl_trigger> ::= [ACTIVE | INACTIVE]
    {BEFORE | AFTER} <ddl_events>
    [POSITION number]
```
<mutation_list> ::= <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= INSERT | UPDATE | DELETE

<db_event> ::= CONNECT | DISCONNECT
| TRANSACTION {START | COMMIT | ROLLBACK}

<ddl_events> ::= {
    ANY DDL STATEMENT
    | <ddl_event_item> [{OR <ddl_event_item}> ...]
}

<ddl_event_item> ::= {CREATE | ALTER | DROP} TABLE
| {CREATE | ALTER | DROP} PROCEDURE
| {CREATE | ALTER | DROP} FUNCTION
| {CREATE | ALTER | DROP} TRIGGER
| {CREATE | ALTER | DROP} EXCEPTION
| {CREATE | ALTER | DROP} VIEW
| {CREATE | ALTER | DROP} DOMAIN
| {CREATE | ALTER | DROP} ROLE
| {CREATE | ALTER | DROP} SEQUENCE
| {CREATE | ALTER | DROP} USER
| {CREATE | ALTER | DROP} INDEX
| {CREATE | DROP} COLLATION
| ALTER CHARACTER SET
| {CREATE | ALTER | DROP} PACKAGE
| {CREATE | DROP} PACKAGE BODY
| {CREATE | ALTER | DROP} MAPPING

<routine-body> ::=<psql-routine-spec>
| <external-routine-spec>

<psql-routine-spec> ::= [<rights-clause>] <psql-routine-body>

<rights-clause> ::= SQL SECURITY {DEFINER | INVOKER}

<psql-routine-body> ::= См. Синтаксис тела модуля

<external-routine-spec> ::= <external-routine-reference>
| AS <extbody>
Оператор `CREATE TRIGGER` создаёт новый триггер. Триггер может быть создан для события (или событий) отношения (таблицы или представления), для события (событий) изменения метаданных или для одного из событий базы данных.

Оператор `CREATE TRIGGER`, как и его родственники `ALTER TRIGGER`, `CREATE OR ALTER TRIGGER` и `RECREATE TRIGGER` являются составными операторами, содержащими заголовок и тело.

Заголовок определяет имя триггера, а также содержит имя отношения (для табличных триггеров), фазу триггера, событие (или события) на которые срабатывает триггер и позицию. Имя триггера должно быть уникальным среди имен других триггеров.

**Привилегии выполнения**

Необязательное предложение `SQL SECURITY` позволяет задать с какими привилегиями
выполняется триггер. Если выбрана опция INVOKER, то триггер выполняется с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то триггер выполняется с привилегиями определяющего пользователя (владельца). Эти привилегии будут дополнены привилегиями выданные самому триггеру с помощью оператора GRANT. По умолчанию триггер наследует привилегии выполнения указанные для таблицы. Триггера на события базы данных по умолчанию выполняются с привилегиями определяющего пользователя (владельца).

Тело триггера

Тело триггера состоит из необязательных объявлений локальных переменных, подпрограмм и именованных курсоров, и одного или нескольких операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN и заканчивается ключевым словом END. Объявления и внутренние операторы завершаются точкой с запятой (;).

Терминатор оператора

Некоторые редакторы SQL-операторов — в частности утилита isql из комплекта Firebird, и возможно некоторые сторонние редакторы — используют внутреннее соглашение, которое требует, чтобы все операторы были завершены с точкой с запятой.

Это создает конфликт с синтаксисом PSQL при кодировании в этих средах. Если вы не знакомы с этой проблемой и её решением, пожалуйста, изучите детали в главе PSQL в разделе, озаглавленном Изменение терминатора в isql.

DML триггеры (на таблицу или представление)

DML триггеры выполняются на уровне строки (записи) каждый раз, когда изменяется образ строки. Они могут быть определены и для таблиц и представлений.

Форма объявления

Объявление DML триггера существует в двух вариантах:

• унаследованная форма;
• SQL-2003 совместимая (рекомендуемая).

В настоящее время рекомендуется использовать SQL-2003 совместимую форму.

Для DML триггера обязательно указывается фаза и одно или несколько событий.

Состояние триггера

Триггер может быть в одном из двух состояний активном (ACTIVE) или неактивном (INACTIVE). Запускаются только активные триггеры. По умолчанию триггеры создаются в активном состоянии.
Фаза

Триггер может выполняться в одной из двух фаз, связанных с запрошенными изменениями состояния данных. Ключевое слово BEFORE означает, что триггер вызывается до наступления соответствующего события (событий, если их указано несколько), AFTER — после наступления события (событий).

События

Для DML триггера может быть указано одно из событий таблицы (представления) — INSERT (добавление), UPDATE (изменение), DELETE (удаление) — или несколько событий, разделённых ключевым словом OR, при которых вызывается триггер. При создании триггера каждое событие (INSERT, UPDATE или DELETE) не должно упоминаться более одного раза.

Контекстные переменные INSERTING, UPDATING и DELETING логического типа могут быть использованы в теле триггера для определения события, которое вызвало срабатывание триггера.

Порядок срабатывания

Ключевое слово POSITION позволяет задать порядок, в котором будут выполняться триггеры с одинаковой фазой и событием (или группы событий). По умолчанию позиция равна 0. Если позиции для триггеров не заданы, или несколько триггеров имеют одно и то же значение позиции, то такие триггеры будут выполняться в алфавитном порядке их имен.

Тело триггера

После ключевого слова AS следует тело триггера.

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции <declarations> описаны локальные переменные триггера, именованные курсоры и подпрограммы (подпроцедуры и подфункции). Подробности вы можете посмотреть в главе “Процедурный язык PSQL” в разделах DECLARE VARIABLE и DECLARE CURSOR, DECLARE PROCEDURE, DECLARE FUNCTION.

После необязательной секции деклараций обязательно следует составной оператор. Составной оператор состоит из одного или нескольких PSQL операторов, заключенных между ключевыми словами BEGIN и END. Составной оператор может содержать один или несколько других составных операторов. Вложенность ограничена 512 уровнями. Любой из BEGIN … END блоков может быть пустым, в том числе и главный блок.

Внешние триггеры

Триггер может быть расположена во внешнем модуле. В этом случае вместо тела триггера указывается место его расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR. После ключевого слова AS может быть указан строковый литерал — "тело" внешнего
триггера, оно может быть использовано внешним модулем для различных целей.

Кто может создать DML триггер?

DML триггеры могут создать:

- Администраторы
- Владелец таблицы (представления);
- Пользователи с привилегией ALTER ANY {TABLE | VIEW}.

Примеры

*Example 145. Создание DML триггера в Legacy стиле*

```sql
CREATE TRIGGER SET_CUST_NO FOR CUSTOMER
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.CUST_NO IS NULL) THEN
    NEW.CUST_NO = GEN_ID(CUST_NO_GEN, 1);
END
```

*Example 146. Создание DML триггера согласно стандарту SQL-2003*

```sql
CREATE TRIGGER set_cust_no
ACTIVE BEFORE INSERT ON customer POSITION 0
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```

*Example 147. Создание DML триггера выполняющегося с правами определяющего пользователя*

```sql
CREATE TRIGGER set_cust_no
ACTIVE BEFORE INSERT ON customer POSITION 0
SQL SECURITY DEFINER
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```
**Example 148. Создание DML триггера на несколько событий**

```sql
CREATE TRIGGER TR_CUST_LOG
ACTIVE AFTER INSERT OR UPDATE OR DELETE
ON CUSTOMER POSITION 10
AS
BEGIN
    INSERT INTO CHANGE_LOG (LOG_ID, ID_TABLE, TABLE_NAME, MUTATION)
    VALUES (NEXT VALUE FOR SEQ_CHANGE_LOG, OLD.CUST_NO, 'CUSTOMER',
            CASE
                WHEN INSERTING THEN 'INSERT'
                WHEN UPDATING THEN 'UPDATE'
                WHEN DELETING THEN 'DELETE'
            END);
END
```

См. также:

**ALTER TRIGGER, DROP TRIGGER.**

**Триггеры на событие базы данных**

Триgger может быть создан для одного из событий базы данных:

- CONNECT (соединение с базой данных или после сброса сеанса);
- DISCONNECT (отсоединение от базы данных или перед сбросом сеанса);
- TRANSACTION START (старт транзакции);
- TRANSACTION COMMIT (подтверждение транзакции);
- TRANSACTION ROLLBACK (откат транзакции).

Контекстная переменная **RESETTING** может использоваться в триггерах на события CONNECT и DISCONNECT для того, чтобы отличить сброс сеанса от подключения/отключения от базы данных.

Указать для триггера несколько событий базы данных невозможно.

**Выполнение триггеров на событие базы данных и обработка исключений**

Триггеры на события CONNECT и DISCONNECT выполняются в специально созданной для этого транзакции. Если при обработке триггера не было вызвано исключение, то транзакция подтверждается. Не перехваченные исключения откатят транзакцию и:

- в случае триггера на событие CONNECT соединение разрывается, а исключения
возвращается клиенту;
• для триггера на событие DISCONNECT соединение разрывается, как это и предусмотрено, но исключения не возвращается клиенту.

Триггеры на события CONNECT и DISCONNECT срабатывают также при выполнении оператора сброса сессионного окружения. Особенности обработки ошибок в триггерах на события CONNECT и DISCONNECT смотрите в секции ALTERN SESSION RESET.

Триггеры на события транзакций срабатывают при старте транзакции, её подтверждении или откате. Не перехваченные исключения обрабатываются в зависимости от типа события:

• для события TRANSACTION START исключение возвращается клиенту, а транзакция отменяется;
• для события TRANSACTION COMMIT исключение возвращается клиенту, действия, выполненные триггером, и транзакция отменяются;
• для события TRANSACTION ROLLBACK исключение не возвращается клиенту, а транзакция, как и предусмотрено, отменяется.

**Ловушки**

Из вышеизложенного следует, что нет прямого способа узнать, какой триггер (DISCONNECT или ROLLBACK) вызвал исключение. Также ясно, что вы не сможете подключиться к базе данных в случае исключения в триггере на событие CONNECT, а также отменяется старт транзакции при исключении в триггере на событие TRANSACTION START. В обоих случаях база данных эффективно блокируется до тех пор, пока вы не отключите триггеры базы данных и не исправите ошибочный код.

**Отключение триггеров**

В некоторые утилиты командной строки Firebird были добавлены новые ключи для отключения триггеров на базу данных:

```bash
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Эти ключи могут использоваться только SYSDBA или владельцем базы данных.

**Двухфазное подтверждение транзакций**

В случае двухфазных транзакций триггеры на событие TRANSACTION START срабатывают в фазе подготовки (prepare), а не в фазе commit.

**Предостережения**

1. Триггеры для событий базы данных DISCONNECT и ROLLBACK не будут вызваны при отключении клиентов через таблицы мониторинга (DELETE FROM MON$ATTACHMENTS).
2. Использование оператора `IN AUTONOMOUS TRANSACTION DO` в триггерах на событие базы данных связанные с транзакциями (COMMIT, ROLLBACK, START) может привести к его зацикливанию.

Кто может создать триггеры на события базы данных?

Триггеры для событий базы данных могут создать:

- **Администраторы**
- Владелец базы данных;
- Пользователи с привилегией `ALTER DATABASE`.

Примеры

**Example 149. Создание триггера на событие подключения к БД для логирования события**

```sql
CREATE TRIGGER tr_log_connect
INACTIVE ON CONNECT POSITION 0
AS
BEGIN
    INSERT INTO LOG_CONNECT (ID, USERNAME, ATIME)
    VALUES (NEXT VALUE FOR SEQ_LOG_CONNECT, CURRENT_USER, CURRENT_TIMESTAMP);
END
```

**Example 150. Создание триггера на событие подключения к БД для контроля доступа**

```sql
CREATE EXCEPTION E_INCORRECT_WORKTIME 'Рабочий день ещё не начался';

CREATE TRIGGER TR_LIMIT_WORKTIME
ON CONNECT POSITION 1
AS
BEGIN
    IF ((CURRENT_USER <> 'SYSDBA') AND NOT (CURRENT_TIME BETWEEN time '9:00' AND time '17:00')) THEN
        EXCEPTION E_INCORRECT_WORKTIME;
END
```

См. также:

`ALTER TRIGGER, DROP TRIGGER.`
Триггеры на события изменения метаданных

Триггеры на события изменения метаданных (DDL триггеры) предназначены для обеспечения ограничений, которые будут распространены на пользователей, которые пытаются создать, изменить или удалить DDL объект. Другое их назначение — ведение журнала изменений метаданных.

Триггеры на события изменения метаданных являются одним из подвидов триггеров на события базы данных.

Особенности:

1. BEFORE триггеры запускаются до изменений в системных таблицах. AFTER триггеры запускаются после изменений в системных таблицах.

2. Когда оператор DDL запускает триггер, в котором возбуждается исключение (BEFORE или AFTER, преднамеренно или неумышленно), оператор не будет фиксирован. Т.е. исключения могут использоваться, чтобы гарантировать, что оператор DDL будет отменен, если некоторые условия не будут соблюдены.

3. Действия DDL триггеров выполняются только при фиксации транзакции, в которой работает затронутая DDL команда. Никогда не забывайте о том, что в AFTER триггере, возможно сделать только то, что возможно сделать после DDL команды без автоматической фиксации транзакций. Вы не можете, например, создать таблицу в триггере и использовать её там.

4. Для операторов CREATE OR ALTER … триггер срабатывает один раз для события CREATE или события ALTER, в зависимости от того существовал ли ранее объект. Для операторов RECREATE триггер вызывается для события DROP, если объект существовал, и после этого для события CREATE.

5. Если объект метаданных не существует, то обычно триггер на события ALTER и DROP не запускаются. Исключения описаны в пункте 6.

6. Исключением из правила 5 являются BEFORE {ALTER | DROP} USER триггеры, которые будут вызваны, даже если имя пользователя не существует. Это вызвано тем, что эти команды выполняются для базы данных безопасности, для которой не делается проверка существования пользователей перед их выполнением. Данное поведение, вероятно, будет отличаться для встроенных пользователей, поэтому не пишите код, который зависит от этого.

7. Если некоторое исключение возбуждено после того как начала выполняться DDL команда и до того как запущен AFTER триггер, то AFTER триггер не запускается.

8. Для процедур и функций в составе пакетов не запускаются индивидуальные триггеры {CREATE | ALTER | DROP} {PROCEDURE | FUNCTION}.

9. Оператор ALTER DOMAIN old name TO new name устанавливает контекстные переменные OLD_OBJECT_NAME и NEW_OBJECT_NAME в обоих триггерах BEFORE и AFTER. Контекстная переменная OBJECT_NAME будет содержать старое имя объекта метаданных в триггере BEFORE, и новое — в триггере AFTER.

Если в качестве события указано предложение ANY DDL STATEMENT, то триггер будет вызван при наступлении любого из DDL событий.
Пространство имен DDL_TRIGGER

Во время работы DDL триггера доступно пространство имен DDL_TRIGGER для использования в функции RDB$GET_CONTEXT. Его использование также допустимо в хранимых процедурах и функциях, вызванных триггерами DDL.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом. В случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имён DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Переменные доступные в пространстве имен DDL_TRIGGER

- EVENT_TYPE – тип события (CREATE, ALTER, DROP)
- OBJECT_TYPE – тип объекта (TABLE, VIEW и др.)
- DDL_EVENT – имя события (<ddl_event_item>),
  где <ddl_event_item> = EVENT_TYPE || ' ' || OBJECT_TYPE
- OBJECT_NAME – имя объекта метаданных
- OLD_OBJECT_NAME – имя объекта метаданных до переименования
- NEW_OBJECT_NAME – имя объекта метаданных после переименования
- SQL_TEXT – текст SQL запроса

Отключение триггеров

В некоторые утилиты командной строки Firebird были добавлены новые ключи для отключения триггеров на базу данных:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Эти ключи могут использоваться только SYSDBA или владельцем базы данных.

Кто может создать триггеры на события изменения метаданных?

Триггеры на события изменения метаданных могут создать:

- Администраторы
- Владелец базы данных;
- Пользователи с привилегией ALTER DATABASE.
Example 151. Контроль наименования объектов базы данных с помощью DDL триггера

```sql
CREATE EXCEPTION e_invalid_sp_name
  'Неверное имя хранимой процедуры (должно начинаться с SP_)';

SET TERM !;

CREATE TRIGGER trig_ddl_sp BEFORE CREATE PROCEDURE
AS
BEGIN
  IF (rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME')
      NOT STARTING 'SP_') THEN
    EXCEPTION e_invalid_sp_name;
END!

-- Test
CREATE PROCEDURE sp_test
AS
BEGIN
END!

CREATE PROCEDURE test
AS
BEGIN
END!

-- Statement failed, SQLSTATE = 42000
-- exception 1
-- -E_INVALID_SP_NAME
-- -Неверное имя хранимой процедуры (должно начинаться с SP_)
-- -At trigger 'TRIG_DDL_SP' line: 4, col: 5

SET TERM ;!
```

Example 152. Контроль безопасности DDL операторов

```sql
CREATE EXCEPTION e_access_denied 'Access denied';

SET TERM !;

CREATE TRIGGER trig_ddl BEFORE ANY DDL STATEMENT
AS
BEGIN
  IF (current_user <> 'SUPER_USER') THEN
```
Chapter 5. Операторы определения данных (DDL)

```
CREATE PROCEDURE sp_test
AS
BEGIN
END!
```

```
-- The last command raises this exception and procedure SP_TEST is not created
-- Statement failed, SQLSTATE = 42000
-- exception 1
-- -E_ACCESS_DENIED
-- -Access denied
-- -At trigger 'TRIG_DDL' line: 4, col: 5
```

```
SET TERM ;!
```

В Firebird существуют привилегии на DDL операторы, поэтому прибегать к написанию DDL триггера нужно только в случае, если того же самого эффекта невозможно достичь стандартными методами.

**Example 153. Использование DDL триггеров для регистрации событий изменения метаданных**

```
CREATE SEQUENCE ddl_seq;

CREATE TABLE ddl_log (  
id BIGINT NOT NULL PRIMARY KEY,
moment TIMESTAMP NOT NULL,
user_name VARCHAR(63) NOT NULL,
event_type VARCHAR(25) NOT NULL,
object_type VARCHAR(25) NOT NULL,
ddl_event VARCHAR(25) NOT NULL,
object_name VARCHAR(63) NOT NULL,
old_object_name VARCHAR(63),
new_object_name VARCHAR(63),
sql_text BLOB sub_type text NOT NULL,
ok CHAR(1) NOT NULL
);

SET TERM !;

CREATE TRIGGER trig_ddl_log_before BEFORE ANY DDL STATEMENT
AS
DECLARE id TYPE OF COLUMN ddl_log.id;
BEGIN
```
мы должны производить изменения в AUTONOMOUS TRANSACTION, таким образом, если произойдет исключение и команда -- не будет запущена, она всё равно будет зарегистрирована.

IN AUTONOMOUS TRANSACTION DO

BEGIN
INSERT INTO ddl_log (id, moment, user_name, event_type, object_type, ddl_event, object_name, old_object_name, new_object_name, sql_text, ok)
VALUES (NEXT VALUE FOR ddl_seq,
  current_timestamp, current_user,
  rdb$get_context('DDL_TRIGGER', 'EVENT_TYPE'),
  rdb$get_context('DDL_TRIGGER', 'OBJECT_TYPE'),
  rdb$get_context('DDL_TRIGGER', 'DDL_EVENT'),
  rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME'),
  rdb$get_context('DDL_TRIGGER', 'OLD_OBJECT_NAME'),
  rdb$get_context('DDL_TRIGGER', 'NEW_OBJECT_NAME'),
  rdb$get_context('DDL_TRIGGER', 'SQL_TEXT'),
  'N')
RETURNING id INTO id;
  rdb$set_context('USER_SESSION', 'trig_ddl_log_id', id);
END
END!

-- Примечание:
-- созданный выше триггер будет запущен для этой DDL.
-- Хорошой идеей является использование –nodbtriggers
-- при работе с ним
CREATE TRIGGER trig_ddl_log_after AFTER ANY DDL STATEMENT
AS
BEGIN
    -- Здесь нам требуется автономная транзакция,
    -- потому что в оригинальной транзакции
    -- мы не увидим запись, вставленную в
    -- BEFORE триггере в автономной транзакции,
    -- если пользовательская транзакция не запущена
    -- с режимом изоляции READ COMMITTED.
    IN AUTONOMOUS TRANSACTION DO
        UPDATE ddl_log SET ok = 'Y'
        WHERE
            id = rdb$get_context('USER_SESSION', 'trig_ddl_log_id');
    END!
    COMMIT!
    SET TERM ;!

    -- Удаляем запись о создании trig_ddl_log_after.
    DELETE FROM ddl_log;
    COMMIT;

    -- Тест
Эта команда будет зарегистрирована единожды
(т.к. T1 не существует, RECREATE вызовет событие CREATE)
с OK = Y.

```sql
RECREATE TABLE t1 (
    n1 INTEGER,
    n2 INTEGER
);
```

Оператор не выполнится, т.к. T1 уже существует,
таким образом OK будет иметь значение N.

```sql
CREATE TABLE t1 (
    n1 INTEGER,
    n2 INTEGER
);
```

T2 не существует. Это действие не будет зарегистрировано.

```sql
DROP TABLE t2;
```

Это действие будет зарегистрировано дважды
(т.к. T1 существует, действие RECREATE рассматривается
как DROP и CREATE) с полем OK = Y.

```sql
RECREATE TABLE t1 (
    n INTEGER
);
```

CREATE DOMAIN dom1 AS INTEGER;

ALTER DOMAIN dom1 TYPE BIGINT;

ALTER DOMAIN dom1 TO dom2;

COMMIT;

```
SELECT
    id,
    ddl_event,
    object_name as name,
    sql_text,
    ok
FROM ddl_log
ORDER BY id;
```
См. также:
ALTER TRIGGER, DROP TRIGGER.

5.7.2. ALTER TRIGGER

Назначение
Изменение существующего триггера.

Доступно в
DSQL, ESQL

Синтаксис

ALTER TRIGGER trigname
[ACTIVE | INACTIVE]
[{{BEFORE | AFTER} <mutation_list>]
[POSITION number]
[SQL SECURITY {DEFINER | INVOKER} | DROP SQL SECURITY]
[<routine-body>]

<mutation_list> ::= <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }
Допустимые изменения

В операторе изменения триггера можно изменить:

- Состояние активности (ACTIVE | INACTIVE);
- Фазу (BEFORE | AFTER);
- Событие(я);
- Позицию срабатывания;
- Привилегии выполнения триггера: вызывающего пользователя (SQL SECURITY INVOKER), определяющего пользователя (SQL SECURITY DEFINER) или наследует у таблицы (DROP SQL SECURITY);
- Код тела триггера.

Если какой-либо элемент не указан, то он остаётся без изменений.

DML триgger невозможно изменить в триgger на событие базы данных и наоборот.

Событие в триggerе базы данных невозможно изменить.

Помните

Триgger с ключевым словом BEFORE наступает до соответствующего события, с ключевым словом AFTER — после соответствующего события.

Один DML триgger может содержать более одного события (INSERT, UPDATE, DELETE). События должны быть разделены ключевым словом OR. Каждое из событий может быть указано не более одного раза.

Ключевое слово POSITION позволяет задать дополнительный порядок выполнения с одинаковыми фазой и событием. По умолчанию позиция равна 0. Если позиция не задана, или если несколько триггеров имеют один и тот же номер позиции, то триггеры будут выполнены в алфавитном порядке их наименований.

Кто может изменить триггеры?

DML триггеры могут изменить:

- Администраторы
- Владелец таблицы (представления);
- Пользователи с привилегией ALTER ANY {TABLE | VIEW}.

Триггеры для событий базы данных и триггеры событий на изменение метаданных могут
изменить:

- Администраторы
- Владелец базы данных;
- Пользователь, имеющий привилегию ALTER DATABASE.

Примеры

Example 154. Отключение (перевод в неактивное состояние) триггера

```
ALTER TRIGGER set_cust_no INACTIVE;
```

Example 155. Изменение позиции триггера

```
ALTER TRIGGER set_cust_no POSITION 14;
```

Example 156. Перевод триггера в неактивное состояние и изменение списка событий

```
ALTER TRIGGER TR_CUST_LOG
INACTIVE AFTER INSERT OR UPDATE;
```

Example 157. Изменение привилегий выполнения триггера

После выполнения данного оператора триггер будет выполняться с привилегиями определяющего пользователя (владельца).

```
ALTER TRIGGER TR_CUST_LOG
SQL SECURITY DEFINER;
```

Example 158. Удаление привилегий выполнения триггера

После удаления привилегий выполнения триггера, триггер выполняется с привилегиями унаследованными от таблицы. Если у таблицы не определены привилегии выполнения, то триггер будет выполняться с привилегиями вызывающего пользователя.

```
ALTER TRIGGER TR_CUST_LOG
DROP SQL SECURITY;
```
Example 159. Перевод триггера в активное состояние, изменение его позиции и его тела

```
ALTER TRIGGER tr_log_connect
ACTIVE POSITION 1
AS
BEGIN
    INSERT INTO LOG_CONNECT (ID, USERNAME, ROLENAME, ATIME)
    VALUES (NEXT VALUE FOR SEQ_LOG_CONNECT, CURRENT_USER, CURRENT_ROLE, CURRENT_TIMESTAMP);
END
```

См. также:
CREATE TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER.

5.7.3. CREATE OR ALTER TRIGGER

Назначение
Создание нового или изменение существующего триггера.

Доступно в
DSQL, ESQL

Синтаксис

```
CREATE OR ALTER TRIGGER trigname {<relation_trigger_legacy>
| <relation_trigger_sql2003>
| <database_trigger>
| <ddl_trigger> }
<routine-body>
```

Полное описание оператора см. CREATE TRIGGER.

Оператор CREATE OR ALTER TRIGGER создаёт новый триггер, если он не существует, или изменяет и перекомпилирует его в противном случае, при этом существующие права и зависимости сохраняются.

Примеры
Example 160. Создание нового или изменение существующего триггера

```sql
CREATE OR ALTER TRIGGER set_cust_no
ACTIVE BEFORE INSERT ON customer POSITION 0
AS
BEGIN
    IF (NEW.cust_no IS NULL) THEN
        NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```

См. также:
CREATE TRIGGER, ALTER TRIGGER, RECREATE TRIGGER.

5.7.4. DROP TRIGGER

Назначение
Удаление существующего триггера.

Доступно в
DSQL, ESQL

Синтаксис

```
DROP TRIGGER triname
```

Table 48. Параметры оператора DROP TRIGGER

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>triname</td>
<td>Имя триггера.</td>
</tr>
</tbody>
</table>

Оператор DROP TRIGGER удаляет существующий триггер.

Кто может удалить триггеры?

DML триггеры могут удалять:

- **Администраторы**
- Владелец таблицы (представления);
- Пользователи с привилегией ALTER ANY {TABLE | VIEW}.

Триггеры для событий базы данных и триггеры событий на изменение метаданных могут удалять:

- **Администраторы**
- Владелец базы данных;
- Пользователь, имеющий привилегию ALTER DATABASE.
Примеры

Example 161. Удаление триггера

```
DROP TRIGGER set_cust_no;
```

См. также:

CREATE TRIGGER, ALTER TRIGGER.

5.7.5. RECREATE TRIGGER

Назначение

Создание нового или пересоздание существующего триггера.

Доступно в

DSQL, ESQl

Синтаксис

```
RECREATE TRIGGER triname { 
  <relation_trigger_legacy> 
  | <relation_trigger_sql2003> 
  | <database_trigger> 
  | <ddl_trigger> }
  <routine-body>
```

Полное описание оператора см. CREATE TRIGGER.

Оператор RECREATE TRIGGER создаёт новый триггер, если триггер с указанным именем не существует, в противном случае оператор RECREATE TRIGGER попытается удалить его и создать новый.

Примеры

Example 162. Создание или пересоздание триггера

```
RECREATE TRIGGER set_cust_no 
ACTIVE BEFORE INSERT ON customer POSITION 0
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```

См. также:
CREATE TRIGGER, DROP TRIGGER, CREATE OR ALTER TRIGGER.

5.8. PROCEDURE

Хранимая процедура (ХП) — это программный модуль, который может быть вызван с клиента, из другой процедуры, функции, выполнимого блока (executable block) или триггера. Хранимые процедуры, хранимые функции, исполняемые блоки и триггеры пишутся на процедурном языке SQL (PSQL). Большинство операторов SQL доступно и в PSQL, иногда с ограничениями или расширениями. Заметными исключениями являются DDL и операторы управления транзакциями.

Хранимые процедуры могут принимать и возвращать множество параметров.

5.8.1. CREATE PROCEDURE

Назначение
Создание новой хранимой функции.

Доступно в
DSQL, ESQL

Синтаксис

```
CREATE PROCEDURE procname [[(<inparam> [, <inparam> ...])] [ RETURNS (<outparam> [, <outparam> ...]) ]
<routine body>

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]

<outparam> ::= <param_decl>

<value> ::= {literal | NULL | context_var}

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<type> ::= <datatype> | [TYPE OF] domain_name | TYPE OF COLUMN rel.col

<datatype> ::= <scalar_datatype> | <blob_datatype>

<scalar_datatype> ::= См. Синтаксис скалярных типов данных

<blob_datatype> ::= См. Синтаксис типа данных BLOB

<routine-body> ::= <psql-routine-spec> | <external-routine-spec>

<psql-routine-spec> ::= [<rights-clause>] <psql-routine-body>
```
<rights-clause> ::=  
   SQL SECURITY {DEFINER | INVOKER}

<psql-routine-body> ::=  
   См. Синтаксис тела модуля

<external-routine-spec> ::=  
   <external-routine-reference>  
   [AS <extbody>]

<external-routine-reference> ::= EXTERNAL NAME <extname> ENGINE <engine>

<extname> ::= '<module-name>!<routine-name>[!<misc-info>]'  

Table 49. Параметры оператора CREATE PROCEDURE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>procname</td>
<td>Имя хранимой процедуры. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
<tr>
<td>outparam</td>
<td>Описание выходного параметра.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал, совместимый по типу с параметром.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом параметра.</td>
</tr>
<tr>
<td>paramname</td>
<td>Имя входного или выходного параметра процедуры. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры, а также её локальных переменных.</td>
</tr>
<tr>
<td>extbody</td>
<td>Тело внешней процедуры. Строковый литерал который может использоваться UDR для различных целей.</td>
</tr>
<tr>
<td>module-name</td>
<td>Имя внешнего модуля, в котором расположена функция.</td>
</tr>
<tr>
<td>routine-name</td>
<td>Внутреннее имя функции внутри внешнего модуля.</td>
</tr>
<tr>
<td>misc-info</td>
<td>Определяемая пользователем информация для передачи в функцию внешнего модуля.</td>
</tr>
<tr>
<td>engine</td>
<td>Имя движка для использования внешних функций. Обычно указывается имя UDR.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
<tr>
<td>rel</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>col</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
</tbody>
</table>
Оператор `CREATE PROCEDURE` создаёт новую хранимую процедуру. Имя хранимой процедуры должно быть уникальным среди имён всех хранимых процедур, таблиц и представлений базы данных.

Желательно также, чтобы имя хранимой процедуры было уникальным и среди имён процедур расположенных в PSQL пакетах (package), хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

CREATE PROCEDURE является составным оператором, состоящий из заголовка и тела.

Заголовок определяет имя хранимой процедуры и объявляет входные и выходные параметры, если они должны быть возвращены процедурой.

Тело процедуры состоит из необязательных объявлений локальных переменных, подпрограмм и именованных курсоров, и одного или нескольких операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова `BEGIN`, и завершается ключевым словом `END`. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (";").

**Терминатор оператора**

Некоторые редакторы SQL-операторов — в частности утилита `isql` из комплекта Firebird, и возможно некоторые сторонние редакторы — используют внутреннее соглашение, которое требует, чтобы все операторы были завершены с точкой с запятой.

Это создает конфликт с синтаксисом PSQL при кодировании в этих средах. Если вы не знакомы с этой проблемой и её решением, пожалуйста, изучите детали в главе PSQL в разделе, озаглавленном Изменение терминатора в `isql`.

**Параметры**

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение `NOT NULL`, тем самым запретив передавать в него значение `NULL`.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения `COLLATE`.

**Входные параметры**

Входные параметры заключаются в скобки после имени хранимой процедуры. Они передаются в процедуру по значению, то есть любые изменения входных параметров внутри процедуры никак не повлияет на значения этих параметров в вызывающей программе.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.
Выходные параметры

Необязательное предложение `RETURNS` позволяет задать список выходных параметров хранимой процедуры.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение `TYPE OF`, то используется только тип данных домена — не проверяются его ограничения `NOT NULL` и `CHECK` (если они есть) и не используется значение по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение `TYPE OF COLUMN`, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании `TYPE OF COLUMN` наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Привилегии выполнения

Необязательное предложение `SQL SECURITY` позволяет задать с какими привилегиями выполняется хранимая процедура. Если выбрана опция `INVOKER`, то хранимая процедура выполняется с привилегиями вызывающего пользователя. Если выбрана опция `DEFINER`, то хранимая процедура выполняется с привилегиями определяющего пользователя (владельца ХП). Эти привилегии будут дополнены привилегиями выданные самой хранимой процедуре с помощью оператора `GRANT`. По умолчанию хранимая процедура выполняется с привилегиями вызывающего пользователя.

Тело хранимой процедуры

После ключевого слова `AS` следует тело хранимой процедуры.

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции `<declarations>` описаны локальные переменные процедуры, подпрограммы и именованные курсоры. В отношении спецификации типа данных
локальные переменные подчиняются тем же правилам, что и входные и выходные параметры процедуры. Подробности вы можете посмотреть в главе “Процедурный язык PSQL” в разделах DECLARE VARIABLE и DECLARE CURSOR, DECLARE PROCEDURE, DECLARE FUNCTION.

После необязательной секции деклараций обязательно следует составной оператор. Составной оператор состоит из одного или нескольких PSQL операторов, заключенных между ключевыми словами BEGIN и END. Составной оператор может содержать один или несколько других составных операторов. Вложенность ограничена 512 уровнями. Любой из BEGIN … END блоков может быть пустым, в том числе и главный блок.

Внешние хранимые процедуры

Хранимая процедура может быть расположена во внешнем модуле. В этом случае вместо тела процедуры указывается место её расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR. После ключевого слова AS может быть указан строковый литерал — "тело" внешней процедуры, оно может быть использовано внешним модулем для различных целей.

Кто может создать хранимую процедуру?

Выполнить оператор CREATE PROCEDURE могут:

- Администраторы
- Пользователи с привилегией CREATE PROCEDURE.

Пользователь, создавший хранимую процедуру, становится её владельцем.

Примеры

Example 163. Создание хранимой процедуры

```sql
CREATE PROCEDURE ADD_BREED (  
    NAME D_BREEDNAME,  
    NAME_EN TYPE OF D_BREEDNAME, 
    SHORTNAME TYPE OF COLUMN BREED.SHORTNAME, 
    REMARK VARCHAR(120) CHARACTER SET WIN1251 COLLATE PXW_CYRL,  
    CODE_ANIMAL INT NOT NULL DEFAULT 1  
)  
RETURNS (  
    CODE_BREED INT  
)  
AS  
BEGIN  
    INSERT INTO BREED ( 
```
CREATE PROCEDURE ADD_BREED (  
    NAME D_BREEDNAME, /* Наследуются характеристики домена */  
    NAME_EN TYPE OF D_BREEDNAME, /* Наследуется только тип домена */  
    SHORTNAME TYPE OF COLUMN BREED.SHORTNAME, /* Наследуется тип столбца таблицы */  
    REMARK VARCHAR(120) CHARACTER SET WIN1251 COLLATE PXW_CYRL,  
    CODE_ANIMAL INT NOT NULL DEFAULT 1  
)  
RETURNS (  
    CODE_BREED INT  
)  
SQL SECURITY DEFINER  
AS  
BEGIN  
    INSERT INTO BREED (  
        CODE_ANIMAL, NAME, NAME_EN, SHORTNAME, REMARK)  
    VALUES (  
        :CODE_ANIMAL, :NAME, :NAME_EN, :SHORTNAME, :REMARK)  
    RETURNING CODE_BREED INTO CODE_BREED;  
END  

Example 164. Создание внешней хранимой процедуры

Создание процедуры находящейся во внешнем модуле (UDR). Реализация процедуры расположена во внешнем модуле udrcpp_example. Имя процедуры внутри модуля — gen_rows.

CREATE PROCEDURE gen_rows (  
    start_n INTEGER NOT NULL,  
    end_n INTEGER NOT NULL  
)  
RETURNS (  
    n INTEGER NOT NULL  
)  
EXTERNAL NAME 'udrcpp_example!gen_rows'  
ENGINE udr;
См. также: CREATE OR ALTER PROCEDURE, ALTER PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE.

5.8.2. ALTER PROCEDURE

Назначение
Изменение существующей хранимой процедуры.

Доступно в
DSQL, ESQL

Синтаксис

```
ALTER PROCEDURE procsame [(<inparam> [, <inparam> ...])] |
[ RETURNS (<outparam> [, <outparam> ...]) ] |
<routine-body>
```

Подробнее см. CREATE PROCEDURE.

Оператор ALTER PROCEDURE позволяет изменять состав и характеристики входных и выходных параметров, локальных переменных, именованных курсоров и тело хранимой процедуры. Для внешних процедур (UDR) вы можете изменить точку входа и имя движка. После выполнения существующие привилегии и зависимости сохраняются.

Будьте осторожны при изменении количества и типов входных и выходных параметров хранимых процедур. Существующий код приложения может стать неработоспособным из-за того, что формат вызова процедуры несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую процедуру, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

Кто может изменить хранимую процедуру?

Выполнить оператор ALTER PROCEDURE могут:

• Администраторы
• Владелец хранимой процедуры;
• Пользователи с привилегией ALTER ANY PROCEDURE.

Примеры

**Example 165. Изменение хранимой процедуры**

```
ALTER PROCEDURE GET_EMP_PROJ ( 
    EMP_NO SMALLINT) 
RETURNS ( 
```
См. также:
CREATE PROCEDURE, CREATE OR ALTER PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE.

5.8.3. CREATE OR ALTER PROCEDURE

Назначение
Создание новой или изменение существующей хранимой процедуры.

Доступно в
DSQL, ESQL

Синтаксис

CREATE OR ALTER PROCEDURE proname [(<inparam> [, <inparam> ...])]
[ RETURNS (<outparam> [, <outparam> ...]) ]
<routine-body>

Подробнее см. CREATE PROCEDURE.

Оператор CREATE OR ALTER PROCEDURE создаёт новую или изменяет существующую хранимую процедуру. Если хранимая процедура не существует, то она будет создана с использованием предложения CREATE PROCEDURE. Если она уже существует, то она будет изменена и откомпилирована, при этом существующие привилегии и зависимости сохраняются.

Примеры

Example 166. Создание или изменение хранимой процедуры

CREATE OR ALTER PROCEDURE GET_EMP_PROJ (  
    EMP_NO SMALLINT)  
RETURNS (  
    PROJ_ID VARCHAR(20))  
AS
BEGIN
FOR SELECT
    PROJ_ID
FROM
    EMPLOYEE_PROJECT
WHERE
    EMP_NO = :emp_no
INTO :proj_id
DO
    SUSPEND;
END

См. также:
CREATE PROCEDURE, ALTER PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE.

5.8.4. DROP PROCEDURE

Назначение
Удаление существующей хранимой процедуры.

Доступно в
DSQL, ESQL

Синтаксис

DROP PROCEDURE procname

Таблица 50. Параметры оператора DROP PROCEDURE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>procname</td>
<td>Имя хранимой процедуры.</td>
</tr>
</tbody>
</table>

Оператор DROP PROCEDURE удаляет существующую хранимую процедуру. Если от хранимой процедуры существуют зависимости, то при попытке удаления такой процедуру будет выдана соответствующая ошибка.

Кто может удалить хранимую процедуру?

Выполнить оператор DROP PROCEDURE могут:

• Администраторы
• Владелец хранимой процедуры;
• Пользователи с привилегией DROP ANY PROCEDURE.

Примеры
**Example 167. Удаление хранимой процедуры**

```sql
DROP PROCEDURE GET_EMP_PROJ;
```

См. также:
CREATE PROCEDURE, RECREATE PROCEDURE.

### 5.8.5. RECREATE PROCEDURE

**Назначение**

Создание новой или пересоздание существующей хранимой процедуры.

**Доступно в**

DSQL, ESQL

**Синтаксис**

```sql
RECREATE PROCEDURE procsname [(<inparam> [, <inparam> ...])]
[ RETURNS (<outparam> [, <outparam> ...]) ]
<routine-body>
```

Подробнее см. CREATE PROCEDURE.

Оператор RECREATE PROCEDURE создаёт новую или пересоздаёт существующую хранимую процедуру. Если процедура с таким именем уже существует, то оператор попытается удалить её и создать новую процедуру. Операция закончится неудачей при подтверждении транзакции, если процедура имеет зависимости.

Имейте в виду, что ошибки зависимостей не обнаруживаются до фазы подтверждения транзакции.

После пересоздания процедуры привилегии на выполнение хранимой процедуры и привилегии самой хранимой процедуры не сохраняются.

**Примеры**

**Example 168. Создание новой или пересоздание существующей хранимой процедуры**

```sql
RECREATE PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
    PROJ_ID VARCHAR(20))
AS
BEGIN
    FOR SELECT
        PROJ_ID
```
FROM EMPLOYEE_PROJECT
WHERE EMP_NO = :emp_no
INTO :proj_id
DO SUSPEND;
END

См. также:
CREATE PROCEDURE, CREATE OR ALTER PROCEDURE, DROP PROCEDURE.

5.9. FUNCTION

Хранимая функция является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой функции самой к себе такая хранимая функция называется рекурсивной.

В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Для возврата значения из хранимой функции используется оператор RETURN, который немедленно прекращает выполнение функции.

5.9.1. CREATE FUNCTION

Назначение
Создание новой хранимой функции.

Доступно в
DSQL

Синтаксис

CREATE FUNCTION funcname [((inparam) [, inparam] ...)]
    RETURNS <type> [NOT NULL] [COLLATE collation]
    [DETERMINISTIC]
    <routine-body>

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]

<value> ::= {<literal> | NULL | <context_var>}

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<type> ::= <datatype>
    | [TYPE OF] domain
Table 51. Параметры оператора CREATE FUNCTION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>funcname</td>
<td>Имя хранимой функции. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал, совместимый по типу с параметром.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом параметра.</td>
</tr>
<tr>
<td>paramname</td>
<td>Имя входного параметра функции. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных параметров функции, а также её локальных переменных.</td>
</tr>
<tr>
<td>module-name</td>
<td>Имя внешнего модуля, в котором расположена функция.</td>
</tr>
<tr>
<td>routine-name</td>
<td>Внутреннее имя функции внутри внешнего модуля.</td>
</tr>
<tr>
<td>misc-info</td>
<td>Определяемая пользователем информация для передачи в функцию внешнего модуля.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>extbody</td>
<td>Тело внешней функции. Строковый литерал который может использоваться UDR для различных целей.</td>
</tr>
<tr>
<td>engine</td>
<td>Имя движка для использования внешних функций. Обычно указывается имя UDR.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
<tr>
<td>rel</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>col</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
</tbody>
</table>

Оператор CREATE FUNCTION создаёт новую хранимую функцию. Имя хранимой функции должно быть уникальным среди имён всех хранимых функций и внешних (UDF) функций. Если только это не внутренняя функция (“подпрограмма”). Для внутренних функций достаточно уникальности только в рамках модулей, которые их “охватывают”.

Желательно также, чтобы имя хранимой функции было уникальным и среди имён функций расположенных в PSQL пакетах (package), хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

CREATE FUNCTION является составным оператором, состоящий из заголовка и тела. Заголовок определяет имя хранимой функции, объявляет входные параметры и тип возвращаемого значения.

Тело функции состоит из необязательных объявлений локальных переменных, подпрограмм и именованных курсоров, и одного или нескольких операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN, и завершается ключевым словом END. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (;).

**Терминатор оператора**

Некоторые редакторы SQL-операторов — в частности утилита isql, которая идёт в комплекте с Firebird, и возможно некоторые сторонние редакторы — используют внутреннее соглашение, которое требует, чтобы все операторы были завершены с точкой с запятой.

Это создает конфликт с синтаксисом PSQL при кодировании в этих средах. Если вы не знакомы с этой проблемой и её решением, пожалуйста, изучите детали в главе PSQL в разделе, озаглавленном Изменение терминатора в isql.
Входные параметры

Входные параметры заключаются в скобки после имени хранимой функции. Они передаются в функцию по значению, то есть любые изменения входных параметров внутри функции никак не повлияет на значения этих параметров в вызывающей программе.

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение TYPE OF, то используется только тип данных домена — не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Возвращаемое значение

Предложение RETURNS задаёт тип возвращаемого значения хранимой функции. Если функция возвращает значение строкового типа, то существует возможность задать порядок сортировки с помощью предложения COLLATE. В качестве типа выходного значения можно указать имя домена, ссылку на его тип (с помощью предложения TYPE OF) или ссылку на тип столбца таблицы (с помощью предложения TYPE OF COLUMN).

Детерминированные функции

Необязательное предложение DETERMINISTIC указывает, что функция детерминированная. Детерминированные функции каждый раз возвращают один и тот же результат, если предоставлять им один и тот же набор входных значений. Недетерминированные функции могут возвращать каждый раз разные результаты, даже если предоставлять им один и тот
же набор входных значений. Если для функции указано, что она является детерминированной, то такая функция не вычисляется заново, если она уже была вычислена однажды с данным набором входных аргументов, а берет свои значения из кэша метаданных (если они там есть).

На самом деле в текущей версии Firebird, не существует кэша хранимых функций с маппингом входных аргументов на выходные значения.

Указание инструкции DETERMINISTIC на самом деле нечто вроде “обещания”, что код функции будет возвращать одно и то же. В данный момент детерминистическая функция считается инвариантом и работает по тем же принципам, что и другие инварианты. Т.е. вычисляется и кэшируется на уровне текущего выполнения данного запроса.

Это легко демонстрируется таким примером:

```sql
CREATE FUNCTION FN_T
RETURNS DOUBLE PRECISION DETERMINISTIC
AS
BEGIN
    RETURN rand();
END

-- функция будет вычислена дважды и вернёт 2 разных значения
SELECT fn_t() FROM rdb$database
UNION ALL
SELECT fn_t() FROM rdb$database

-- функция будет вычислена единожды и вернёт 2 одинаковых значения
WITH t(n) AS (
    SELECT 1 FROM rdb$database
    UNION ALL
    SELECT 2 FROM rdb$database
)
SELECT n, fn_t() FROM t
```

**Привилегии выполнения**

Необходимое предложение SQL SECURITY позволяет задать с какими привилегиями выполняется хранимая функция. Если выбрана опция INVOKER, то хранимая функция выполняется с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то хранимая функция выполняется с привилегиями определяющего пользователя (владельца функции). Эти привилегии будут дополнены привилегиями выданные самой хранимой функции с помощью оператора GRANT. По умолчанию хранимая функция выполняется с привилегиями вызывающего пользователя.

Привилегии выполнения по умолчанию для вновь создаваемых объектов метаданных можно изменить с помощью оператора...
Тело хранимой функции

После ключевого слова AS следует тело хранимой функции.

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции <declarations> описаны локальные переменные, именованные курсоры и подпрограммы (подпроцедуры и подфункции). Локальные переменные подчиняются тем же правилам, что и входные параметры функции в отношении спецификации типа данных. Подробности вы можете посмотреть в главе “Процедурный язык PSQL” в разделах DECLARE VARIABLE и DECLARE CURSOR, DECLARE PROCEDURE, DECLARE FUNCTION.

После необязательной секции деклараций обязательно следует составной оператор. Составной оператор состоит из одного или нескольких SQL операторов, заключенных между ключевыми словами BEGIN и END. Составной оператор может содержать один или несколько других составных операторов. Вложенность ограничена 512 уровнями. Любой из BEGIN … END блоков может быть пустым, в том числе и главный блок.

Внешние функции

Хранимая функция может быть расположена во внешнем модуле. В этом случае вместо тела функции указывается место расположения функции во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя функции внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR. После ключевого слова AS может быть указан строковый литерал — "тело" внешней функции, оно может быть использовано внешним модулем для различных целей.

Не следует путать внешние функции, объявленные как DECLARE EXTERNAL FUNCTION, так же известные как UDF, с функциями расположенными во внешних модулях объявленных как CREATE FUNCTION … EXTERNAL NAME, называемых UDR (User Defined Routine). Первые являются унаследованными (Legacy) из предыдущих версий Firebird. Их возможности существенно уступают возможностям нового типа внешних функций. В Firebird 4.0 UDF объявлены устаревшими.

Кто может создать функцию?

Выполнить оператор CREATE FUNCTION могут:

- Администраторы
- Пользователи с привилегией CREATE FUNCTION.
Пользователь, создавший хранимую функцию, становится её владельцем.

Примеры

Example 169. Создание хранимой функции

```sql
CREATE FUNCTION ADD_INT(A INT, B INT DEFAULT 0)
RETURNS INT
AS
BEGIN
    RETURN A+B;
END
```

Вызов в запросе:

```sql
SELECT ADD_INT(2, 3) AS R FROM RDB$DATABASE
```

Вызов внутри PSQL кода, второй необязательный параметр не указан:

```sql
MY_VAR = ADD_INT(A);
```

Example 170. Создание детерминистической хранимой функции

```sql
CREATE FUNCTION FN_E()
RETURNS DOUBLE PRECISION DETERMINISTIC
AS
BEGIN
    RETURN EXP(1);
END
```

Example 171. Создание хранимой функции с параметрами типа столбца таблицы

Функция, возвращающая имя мнемоники по имени столбца и значения мнемоники.

```sql
CREATE FUNCTION GET_MNEMONIC ( 
    AFIELD_NAME TYPE OF COLUMN RDB$TYPES.RDB$FIELD_NAME, 
    ATYPE TYPE OF COLUMN RDB$TYPES.RDB$TYPE) 
RETURNS TYPE OF COLUMN RDB$TYPES.RDB$TYPE_NAME 
AS
BEGIN
    RETURN (SELECT RDB$TYPE_NAME 
            FROM RDB$TYPES 
            WHERE RDB$FIELD_NAME = :AFIELD_NAME 
            AND RDB$TYPE = :ATYPE);
```
Глава 5. Операторы определения данных (DDL)

То же самое, но хранимая функция будет выполняться с привилегиями определяющего пользователя (владельца функции).

```sql
CREATE FUNCTION GET_MNEMONIC (  
    AFIELD_NAME TYPE OF COLUMN RDB$TYPES.RDB$FIELD_NAME,  
    ATYPE TYPE OF COLUMN RDB$TYPES.RDB$TYPE)  
RETURNS TYPE OF COLUMN RDB$TYPES.RDB$TYPE_NAME  
SQL SECURITY DEFINER  
AS  
BEGIN  
    RETURN (SELECT RDB$TYPE_NAME  
              FROM RDB$TYPES  
              WHERE RDB$FIELD_NAME = :AFIELD_NAME  
              AND RDB$TYPE = :ATYPE);  
END
```

**Example 172. Создание внешней хранимой функции**

Создание функции находящейся во внешнем модуле (UDR). Реализация функции расположена во внешнем модуле udr.cpp_example. Имя функции внутри модуля — `wait_event`.

```sql
CREATE FUNCTION wait_event (  
    event_name varchar(63) CHARACTER SET ascii  
) RETURNS INTEGER  
EXTERNAL NAME 'udr.cpp_example!wait_event'  
ENGINE udr
```

**Example 173. Создание хранимой функции содержащую подфункцию**

Создание функции для перевода числа в шестнадцатеричный формат.

```sql
CREATE FUNCTION INT_TO_HEX (  
    ANumber BIGINT,  
    AByte_Per_Number SMALLINT = 8)  
RETURNS CHAR(66)  
AS  
DECLARE VARIABLE xMod SMALLINT;  
DECLARE VARIABLE xResult VARCHAR(64);  
DECLARE FUNCTION TO_HEX(ANum SMALLINT) RETURNS CHAR  
AS  
BEGIN  
    RETURN CASE ANum
```
WHEN 0 THEN '0'
WHEN 1 THEN '1'
WHEN 2 THEN '2'
WHEN 3 THEN '3'
WHEN 4 THEN '4'
WHEN 5 THEN '5'
WHEN 6 THEN '6'
WHEN 7 THEN '7'
WHEN 8 THEN '8'
WHEN 9 THEN '9'
WHEN 10 THEN 'A'
WHEN 11 THEN 'B'
WHEN 12 THEN 'C'
WHEN 13 THEN 'D'
WHEN 14 THEN 'E'
WHEN 15 THEN 'F'
ELSE NULL
END;

BEGIN
xMod = MOD(ANumber, 16);
ANumber = ANumber / 16;
xResult = TO_HEX(xMod);
WHILE (ANumber > 0) DO
    BEGIN
        xMod = MOD(ANumber, 16);
        ANumber = ANumber / 16;
        xResult = TO_HEX(xMod) || xResult;
    END
END
RETURN '0x' || LPAD(xResult, AByte_Per_Number * 2, '0');
END

См. также:
CREATE OR ALTER FUNCTION, ALTER FUNCTION, RECREATE FUNCTION, DROP FUNCTION.

5.9.2. ALTER FUNCTION

Назначение
Изменение существующей хранимой функции.

Доступно в
DSQL

Синтаксис

ALTER FUNCTION funcname
[(<inparam> [, <inparam> ...])]  
RETURNS <type> [COLLATE collation]  
[DETERMINISTIC]
Оператор ALTER FUNCTION позволяет изменять состав и характеристики входных параметров, типа выходного значения, локальных переменных, именованных курсоров, подпрограмм и тело хранимой функции. Для внешних функций (UDR) вы можете изменить точку входа и имя движка. Внешние функции, объявленные как DECLARE EXTERNAL FUNCTION, так же известные как UDF, невозможно преобразовать в PSQL функции и наоборот. После выполнения существующие привилегии и зависимости сохраняются.

Будьте осторожны при изменении количества и типов входных параметров хранимых функций. Существующий код приложения может стать неработоспособным из-за того, что формат вызова функции несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую функцию, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

Если у вас уже есть внешняя функция в Legacy стиле (DECLARE EXTERNAL FUNCTION), то оператор ALTER FUNCTION изменит её на обычную функцию без всяких предупреждений. Это было сделано умышленно для облегчения миграции на новый стиль написания внешних функций известных как UDR.

Кто может изменить функцию?

Выполнить оператор ALTER FUNCTION могут:

- Администраторы
- Владелец хранимой функции;
- Пользователи с привилегией ALTER ANY FUNCTION.

Примеры

*Example 174. Изменение хранимой функции*

```
ALTER FUNCTION ADD_INT(A INT, B INT, C INT)
RETURNS INT
AS
BEGIN
  RETURN A+B+C;
END
```

См. также:

CREATE FUNCTION, CREATE OR ALTER FUNCTION, DROP FUNCTION.
5.9.3. CREATE OR ALTER FUNCTION

Назначение
Создание новой или изменение существующей хранимой функции.

Доступно в
DSQL

Синтаксис

CREATE OR ALTER FUNCTION funcname
c((<inparam> [, <inparam> ...]))
RETURNS <type> [COLLATE collation]
[DETERMINISTIC]
<routine-body>

Подробнее см. CREATE FUNCTION.

Оператор CREATE OR ALTER FUNCTION создаёт новую или изменяет существующую хранимую функцию. Если хранимая функция не существует, то она будет создана с использованием предложения CREATE FUNCTION. Если она уже существует, то она будет изменена и перекомпилирована, при этом существующие привилегии и зависимости сохраняются.

Если у вас уже есть внешняя функция в Legacy стиле (DECLARE EXTERNAL FUNCTION), то оператор CREATE OR ALTER FUNCTION изменит её на обычную функцию без всяких предупреждений. Это было сделано умышленно для облегчения миграции на новый стиль написания внешних функций известных как UDR.

Примеры

Example 175. Создание новой или изменение существующей хранимой функции

```sql
CREATE OR ALTER FUNCTION ADD_INT(A INT, B INT DEFAULT 0)
RETURNS INT
AS
BEGIN
    RETURN A+B;
END
```

См. также:
CREATE FUNCTION, ALTER FUNCTION.

5.9.4. DROP FUNCTION

Назначение
Удаление хранимой функции.

Доступно в
DSQL

Синтаксис

```
DROP FUNCTION funcname
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>funcname</td>
<td>Имя хранимой функции.</td>
</tr>
</tbody>
</table>

Оператор DROP FUNCTION удаляет существующую хранимую функцию. Если от хранимой функции существуют зависимости, то при попытке удаления такой функции будет выдана соответствующая ошибка.

Кто может удалить функцию?

Выполнить оператор DROP FUNCTION могут:

- Администраторы
- Владелец хранимой функции;
- Пользователи с привилегией DROP ANY FUNCTION.

Примеры

Example 176. Удаление хранимой функции

```
DROP FUNCTION ADD_INT;
```

См. также:
CREATE FUNCTION.

5.9.5. RECREATE FUNCTION

Назначение

Создание новой или пересоздание существующей хранимой функции.

Доступно в
DSQL

Синтаксис

```
RECREATE FUNCTION funcname
```
Оператор RECREATE FUNCTION создаёт новую или пересоздаёт существующую хранимую функцию. Если функция с таким именем уже существует, то оператор попытается удалить её и создать новую функцию. Операция закончится неудачей при подтверждении транзакции, если функция имеет зависимости.

Имейте в виду, что ошибки зависимостей не обнаруживаются до фазы подтверждения транзакции.

После пересоздания функции привилегии на выполнение хранимой функции и привилегии самой хранимой функции не сохраняются.

Примеры

**Example 177. Создание или пересоздание хранимой функции**

```sql
RECREATE FUNCTION ADD_INT(A INT, B INT DEFAULT 0)
RETURNS INT
AS
BEGIN
    RETURN A+B;
END
```

См. также: CREATE FUNCTION, DROP FUNCTION.

### 5.10. PACKAGE

Пакет — группа процедур и функций, которая представляет собой один объект базы данных.

Пакеты Firebird состоят из двух частей: заголовка (ключевое слово PACKAGE) и тела (ключевые слова PACKAGE BODY). Такое разделение очень сильно напоминает модули Delphi, заголовок соответствует интерфейсной части, а тело — части реализации.

Сначала создаётся заголовок (CREATE PACKAGE), а затем — тело (CREATE PACKAGE BODY).

#### 5.10.1. CREATE PACKAGE

**Назначение**

Создание заголовка пакета.
Table 53. Параметры оператора CREATE PACKAGE
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Имя пакета. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>function_decl</td>
<td>Объявление функции.</td>
</tr>
<tr>
<td>procedure_decl</td>
<td>Объявление процедуры.</td>
</tr>
<tr>
<td>proc_name</td>
<td>Имя процедуры. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>func_name</td>
<td>Имя функции. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
<tr>
<td>outparam</td>
<td>Описание выходного параметра.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал, совместимый по типу с параметром.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом параметра.</td>
</tr>
<tr>
<td>paramname</td>
<td>Имя входного или выходного параметра процедуры/функции. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры/функции, а также её локальных переменных.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
<tr>
<td>rel</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>col</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
</tbody>
</table>

Оператор CREATE PACKAGE создаёт новый заголовок пакета. Имя пакета должно быть уникальным среди имён всех пакетов.

Процедуры и функции, объявленные в заголовке пакета, доступны вне тела пакета через полный идентификатор имён процедур и функций (package_name.procedure_name и package_name.function_name). Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имена процедур и функций, объявленные в заголовке пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

Желательно чтобы имена хранимых процедур и функций пакета не пересекались с именами хранимых процедур и функций из глобального пространства имен, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/фунция пакета.
Привилегии выполнения

Необязательное предложение SQL SECURITY позволяет задать с какими привилегиями выполняется процедуры и функции пакета. Если выбрана опция INVOKER, то процедуры и функции пакета выполняются с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то процедуры и функции пакета выполняются с привилегиями определяющего пользователя (владельца пакета). Эти привилегии будут дополнены привилегиями выданные самому пакету с помощью оператора GRANT. По умолчанию процедуры и функции пакета выполняются с привилегиями вызывающего пользователя. Переопределять привилегии выполнения для процедур и функций пакета запрещено.

Терминатор оператора

Некоторые редакторы SQL-операторов — в частности утилита isql из комплекта Firebird, и возможно некоторые сторонние редакторы — используют внутреннее соглашение, которое требует, чтобы все операторы были завершены с точкой с запятой.

Это создает конфликт с синтаксисом PSQL при кодировании в этих средах. Если вы не знакомы с этой проблемой и её решением, пожалуйста, изучите детали в главе PSQL в разделе, озаглавленном Изменение терминатора в isql.

Параметры процедур и функций

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Входные параметры

Входные параметры заключаются в скобки после имени хранимой процедуры. Они передаются в процедуру по значению, то есть любые изменения входных параметров внутри процедуры никак не повлияет на значения этих параметров в вызывающей программе.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.

Выходные параметры

Для хранимых процедур список выходных параметров задаётся в необязательное предложение RETURNS.
Для хранимых функций в обязательном предложении `RETURNS` задаётся тип возвращаемого значения.

**Использование доменов при объявлении параметров**

В качестве типа параметра можно указать имя домена. В этом случае параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение `TYPE OF`, то используется только тип данных домена — не проверяются его ограничения `NOT NULL` и `CHECK` (если они есть), а также не используется значение по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

**Использование типа столбца при объявлении параметров**

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение `TYPE OF COLUMN`, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании `TYPE OF COLUMN` наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

**Детерминированные функции**

Необязательное предложение `DETERMINISTIC` в объявлении функции указывает, что функция детерминированная. Детерминированные функции каждый раз возвращают один и тот же результат, если предоставлять им один и тот же набор входных значений. Недетерминированные функции могут возвращать каждый раз разные результаты, даже если предоставлять им один и тот же набор входных значений. Если для функции указано, что она является детерминированной, то такая функция не вычисляется заново, если она уже была вычислена однажды с данным набором входных аргументов, а берет свои значения из кэша метаданных (если они там есть).

На самом деле в текущей версии Firebird, не существует кэша хранимых функций с маппингом входных аргументов на выходные значения.

Указание инструкции `DETERMINISTIC` на самом деле нечто вроде “обещания”, что код функции будет возвращать одно и то же. В данный момент детерминистическая функция считается инвариантом и работает по тем же принципам, что и другие инварианты. Т.е. вычисляется и кэшируется на уровне текущего выполнения данного запроса.

**Кто может создать пакет?**

Выполнить оператор `CREATE PACKAGE` могут:

- Администраторы
- Пользователи с привилегией `CREATE PACKAGE`. 
Пользователь, создавший заголовок пакета становится владельцем пакета.

**Примеры**

*Example 178. Создание заголовка пакета*

```sql
CREATE PACKAGE APP_VAR
AS
BEGIN
    FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
    FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
    PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

То же самое, но процедуры и функции пакета будут выполняться с правами определяющего пользователя (владельца пакета).

```sql
CREATE PACKAGE APP_VAR
SQL SECURITY DEFINER
AS
BEGIN
    FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
    FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
    PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также:

CREATE PACKAGE BODY, ALTER PACKAGE, DROP PACKAGE.

### 5.10.2. ALTER PACKAGE

**Назначение**

Изменение заголовка пакета.

**Доступно в**

DSQL

**Синтаксис**

```sql
ALTER PACKAGE package_name
[<rights clause>]
AS
BEGIN
    [...]
END
```

`<package_item> ::=`
Оператор ALTER PACKAGE изменяет заголовок пакета. Позволяется изменять количество и состав процедур и функций, их входных и выходных параметров. При этом исходный код тела пакета сохраняется. Состояние соответствия тела пакета его заголовку отображается в столбце RDB$PACKAGES.RDB$VALID_BODY_FLAG.

Кто может изменить заголовок пакета?

Выполнить оператор ALTER PACKAGE могут:

• Администраторы
• Владелец пакета;
• Пользователи с привилегией ALTER ANY PACKAGE.

Примеры

Example 179. Изменение заголовка пакета

```sql
ALTER PACKAGE APP_VAR
AS
BEGIN
    FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
    FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
    PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также:

CREATE PACKAGE, DROP PACKAGE, RECREATE PACKAGE BODY.

5.10.3. CREATE OR ALTER PACKAGE

Назначение

Создание нового или изменение существующего заголовка пакета.
Доступно в
DSQL

Синтаксис

CREATE OR ALTER PACKAGE package_name
[<rights clause>]
AS
BEGIN
  [<package_item> ...]
END

<package_item> ::=  
  <function_decl>;  
  | <procedure_decl>;

<function_decl> ::=  
  FUNCTION func_name [(<in_params>)]
  RETURNS <type> [COLLATE collation]
  [DETERMINISTIC]

<procedure_decl> ::=  
  PROCEDURE proc_name [(<in_params>)]
  [RETURNS (<out_params>)]

Подробнее см. CREATE PACKAGE

Оператор CREATE OR ALTER PACKAGE создаёт новый или изменяет существующий заголовок пакета. Если заголовок пакета не существует, то он будет создан с использованием предложения CREATE PACKAGE. Если он уже существует, то он будет изменен и перекомпилирован, при этом существующие привилегии и зависимости сохраняются.

Примеры

Example 180. Создание нового или изменение существующего заголовка пакета

CREATE OR ALTER PACKAGE APP_VAR
AS
BEGIN
  FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
  FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END

См. также:
CREATE PACKAGE, ALTER PACKAGE, RECREATE PACKAGE BODY.
5.10.4. DROP PACKAGE

Назначение
Удаление заголовка пакета.

Доступно в
DSQL

Синтаксис

```
DROP PACKAGE package_name
```

Таблица 54. Параметры оператора DROP PACKAGE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Имя пакета.</td>
</tr>
</tbody>
</table>

Оператор DROP PACKAGE удаляет существующий заголовок пакета. Перед удалением заголовка пакета (DROP PACKAGE), необходимо выполнить удаление тела пакета (DROP PACKAGE BODY), иначе будет выдана ошибка. Если от заголовка пакета существуют зависимости, то при попытке удаления такого заголовка будет выдана соответствующая ошибка.

Кто может удалить заголовок пакета?

Выполнить оператор DROP PACKAGE могут:

- Администраторы
- Владелец пакета;
- Пользователи с привилегией DROP ANY PACKAGE.

Примеры

Пример 181. Удаление заголовка пакета

```
DROP PACKAGE APP_VAR;
```

См. также:
CREATE PACKAGE, ALTER PACKAGE, DROP PACKAGE BODY.

5.10.5. RECREATE PACKAGE

Назначение
Создание нового или пересоздание существующего заголовка пакета.

Доступно в
DSQL
Синтаксис

```sql
RECREATE PACKAGE package_name
[<rights clause>]
AS
BEGIN
    [<package_item> ...]
END

<package_item> ::=<function_decl>;
    | <procedure_decl>;

<function_decl> ::=FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation]
    [DETERMINISTIC]

<procedure_decl> ::=PROCEDURE proc_name [(<in_params>)]
    [RETURNS (<out_params>)]
```

Подробнее см. CREATE PACKAGE

Оператор RECREATE PACKAGE создает новый или пересоздает существующий заголовок пакета. Если заголовок пакета с таким именем уже существует, определение попытается удалить его и создать новый заголовок пакета. Пересоздать заголовок пакета невозможно, если у существующей заголовка пакета имеются зависимости или существует тело этого пакета. После пересоздания заголовка пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета не сохраняются.

Примеры

Example 182. Создание нового или пересоздание существующего заголовка пакета

```sql
RECREATE PACKAGE APP_VAR
AS
BEGIN
    FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
    FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
    PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также:

CREATE PACKAGE, DROP PACKAGE, RECREATE PACKAGE BODY.
5.11. PACKAGE BODY

5.11.1. CREATE PACKAGE BODY

Назначение
Создание тела пакета.

Доступно в
DSQL

Синтаксис

```sql
CREATE PACKAGE BODY package_name
AS
BEGIN
  [<package_item> ...]
  [<package_body_item> ...]
END

<package_item> ::=<function_decl>;
  | <procedure_decl>;

<function_decl> ::=FUNCTION func_name [(<in_params>)]
  RETURNS <type> [NOT NULL] [COLLATE collation]
  [DETERMINISTIC]

<procedure_decl> ::=PROCEDURE proc_name [(<in_params>)]
  [RETURNS (<out_params>)]

<package_body_item> ::=<function_impl>
  | <procedure_impl>

<function_impl> ::=FUNCTION func_name [(<impl_params>)]
  RETURNS <type> [NOT NULL] [COLLATE collation]
  [DETERMINISTIC]
  <routine-body>

<procedure_impl> ::=PROCEDURE proc_name [(<impl_params>)]
  [RETURNS (<out_params>)]
  <routine-body>

<in_params> ::= <inparam> [, <inparam> ...]

<inparam> ::= <param_decl> [={ | DEFAULT} <value>]
```
Table 55. Параметры оператора CREATE PACKAGE BODY

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Имя пакета. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>function_decl</td>
<td>Объявление функции.</td>
</tr>
<tr>
<td>procedure_decl</td>
<td>Объявление процедуры.</td>
</tr>
<tr>
<td>function_impl</td>
<td>Реализация функции.</td>
</tr>
<tr>
<td>procedure_impl</td>
<td>Реализация процедуры.</td>
</tr>
<tr>
<td>proc_name</td>
<td>Имя процедуры. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>func_name</td>
<td>Имя функции. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
</tbody>
</table>
### Операторы определения данных (DDL)

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>outparam</td>
<td>Описание выходного параметра.</td>
</tr>
<tr>
<td>subfunc_impl</td>
<td>Реализация подпрограммы–функции.</td>
</tr>
<tr>
<td>subproc_impl</td>
<td>Реализация подпрограммы–процедуры.</td>
</tr>
<tr>
<td>module-name</td>
<td>Имя внешнего модуля, в котором расположена процедура/функция.</td>
</tr>
<tr>
<td>routine-name</td>
<td>Внутреннее имя процедуры/функции внутри внешнего модуля.</td>
</tr>
<tr>
<td>misc-info</td>
<td>Определяемая пользователем информация для передачи в функцию внешнего модуля.</td>
</tr>
<tr>
<td>extbody</td>
<td>Тело внешней процедуры или функции. Строковый литерал, который может использоваться UDR для различных целей.</td>
</tr>
<tr>
<td>engine</td>
<td>Имя движка для использования внешних функций. Обычно указывается имя UDR.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал, совместимый по типу с параметром.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом параметра.</td>
</tr>
<tr>
<td>paramname</td>
<td>Имя входного или выходного параметра процедуры/функции. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры/функции, а также её локальных переменных.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
<tr>
<td>domain_name</td>
<td>Имя домена.</td>
</tr>
<tr>
<td>rel</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>col</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
</tbody>
</table>

Оператор `CREATE PACKAGE BODY` создаёт новое тело пакета. Тело пакета может быть создано только после того как будет создан заголовок пакета. Если заголовка пакета с именем `package_name` не существует, то будет выdana соответствующая ошибка.

Все процедуры и функции, объявленные в заголовке пакета, должны быть реализованы в теле пакета. Кроме того, должны быть реализованы и все процедуры и функции, объявленные в теле пакета. Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имена процедур и функций, объявленные в теле пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

Желательно чтобы имена хранимых процедур и функций пакета не пересекались с именами хранимых процедур и функций из глобального пространства имен, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имен внутри пакета, если в пакете объявлена одноименная
функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

Правила:

• В теле пакеты должны быть реализованы все подпрограммы, стой же сигнатурой, что и объявленные в заголовке и в начале тела пакета.

• Значения по умолчанию для параметров процедур, которые указываются в <package_item>, не могут быть переопределены. Это означает, что они могут быть в <package_body_item> только для частных процедур, которые не были объявлены.

UDF деклараций (DECLARE внешняя функция) в настоящее время не поддерживается внутри пакетов.

Кто может создать тело пакета?

Выполнить оператор CREATE PACKAGE BODY могут:

• Администраторы
• Владелец пакета;
• Пользователи с привилегией ALTER ANY PACKAGE.

Примеры

Example 183. Создание тела пакета

```
CREATE PACKAGE BODY APP_VAR AS
BEGIN
  -- Возвращает дату начала периода
  FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC AS BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEBEGIN');
  END
  -- Возвращает дату окончания периода
  FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC AS BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEEND');
  END
  -- Устанавливает диапазон дат рабочего периода
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE) AS BEGIN
    RDB$SET_CONTEXT('USER_SESSION', 'DATEBEGIN', ADATEBEGIN);
    RDB$SET_CONTEXT('USER_SESSION', 'DATEEND', ADATEEND);
  END
```
См. также:
DROP PACKAGE BODY, CREATE PACKAGE.

5.11.2. DROP PACKAGE BODY

Назначение
Удаление тела пакета.

Доступно в
DSQL

Синтаксис

```
DROP PACKAGE BODY package_name
```

Таблица 56. Параметры оператора DROP PACKAGE BODY

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Имя пакета</td>
</tr>
</tbody>
</table>

Оператор DROP PACKAGE BODY удаляет тело пакета.

Кто может удалить тело пакета?

Выполнить оператор DROP PACKAGE BODY могут:

- Администраторы
- Владелец пакета;
- Пользователи с привилегией ALTER ANY PACKAGE.

Примеры

Example 184. Удаление тела пакета

```
DROP PACKAGE BODY APP_VAR;
```

См. также:
CREATE PACKAGE BODY, DROP PACKAGE.

5.11.3. RECREATE PACKAGE BODY

Назначение
Создание нового и пересоздание существующего тела пакета.

Доступно в DSQL

Синтаксис

```sql
RECREATE PACKAGE BODY package_name
AS
BEGIN
    [<package_item> ...]
    [<package_body_item> ...]
END

<package_item> ::=  
    <function_decl>;
  | <procedure_decl>;

<funtion_decl> ::=  
    FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation]
    [<function_options>]

<procedure_decl> ::=  
    PROCEDURE proc_name [(<in_params>)]
    [RETURNS (<out_params>)]
    [<procedure_options>]

<package_body_item> ::=  
    <function_impl>
  | <procedure_impl>

<function_impl> ::=  
    FUNCTION func_name [(<in_impl_params>)]
    RETURNS <type> [COLLATE collation]
    [DETERMINISTIC]
    <routine-body>

<procedure_impl> ::=  
    PROCEDURE proc_name [(<impl_params>)]
    [RETURNS (<out_params>)]
    <routine-body>
```

Подробнее см. CREATE PACKAGE BODY.

Оператор `RECREATE PACKAGE BODY` создаёт новое или пересоздаёт существующее тело пакета. Если тело пакета с таким именем уже существует, то оператор попытается удалить его и создать новое тело пакета. После пересоздания тела пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета сохраняются.
Примеры

Example 185. Пересоздание тела пакета

```sql
RECREATE PACKAGE BODY APP_VAR
AS
BEGIN
-- Возвращает дату начала периода
FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC
AS
BEGIN
RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEBEGIN');
END
-- Возвращает дату окончания периода
FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC
AS
BEGIN
RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEEND');
END
-- Устанавливает диапазон дат рабочего периода
PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE)
AS
BEGIN
RDB$SET_CONTEXT('USER_SESSION', 'DATEBEGIN', ADATEBEGIN);
RDB$SET_CONTEXT('USER_SESSION', 'DATEEND', ADATEEND);
END
END
```

См. также:
CREATE PACKAGE BODY, DROP PACKAGE BODY.

5.12. EXTERNAL FUNCTION

Внешние функции, также известные как функции определяемые пользователем (User Defined Function) — это программы, написанные на любом языке программирования, и хранящиеся в динамических библиотеках. После того как функция объявлена в базе данных, она становится доступной в динамических и процедурных операторах, как будто они реализованы внутри языка SQL.

Внешние функции существенно расширяют возможности SQL по обработке данных. Для того чтобы функции были доступны в базе данных, их необходимо объявить с помощью оператора DECLARE EXTERNAL FUNCTION.

После объявления функции, содержащая её библиотека будет загружаться при первом обращении к любой из функций, включённой в библиотеку.

Внешние функции (UDF) объявлены устаревшими в Firebird 4:
• По умолчанию для параметра конфигурации UdfAccess установлено значение None. Для того чтобы запускать UDF, теперь потребуется явная конфигурация UdfAccess = Restrict path-list.

• UDF библиотеки (ib_udf, fbudf) больше не входят в состав установочных комплектов.

• Большинство функций в библиотеках, ранее распространявшихся в общих (динамических) библиотеках ib_udf и fbudf, уже заменены на встроенные функции. Несколько оставшихся UDF были заменены либо аналогичными функциями в новой UDR библиотеке под названием udf_compat, либо преобразованы в сохраненные функции.

Обратитесь к разделу “Прекращение поддержки внешних функций (UDF)” в главе “Совместимость” Firebird 4.0 Release Notes для получения подробной информации и инструкций по обновлению для использования безопасных функций.

• Настоятельно рекомендуется заменить UDF на UDR или сохраненные функции. См. CREATE FUNCTION.

UDF принципиально небезопасны. Мы рекомендуем по возможности избегать их использования и отключать UDF в конфигурации вашей базы данных (UdfAccess = None в firebird.conf, значение по умолчанию начиная с Firebird 4.0). Если вам действительно нужно вызвать собственный код из вашей базы данных, используйте вместо этого механизм UDR.

5.12.1. DECLARE EXTERNAL FUNCTION

Назначение
Объявление в базе данных функции определённой пользователем (UDF).

Доступно в
DSQL, ESQL

Синтаксис

DECLARE EXTERNAL FUNCTION funcname
[ { <arg_desc_list> | ( <arg_desc_list> ) } ]
RETURNS { <return_value> | ( <return_value> ) }
ENTRY_POINT 'entry_point' MODULE_NAME 'library_name'

<arg_desc_list> ::=<arg_type_decl> [, <arg_type_decl> ...]

<arg_type_decl> ::=<udf_data_type> [BY {DESCRIPTOR | SCALAR_ARRAY} | NULL]

<udf_data_type> ::=<scalar_datatype>
Оператор DECLARE EXTERNAL FUNCTION делает доступным внешнюю функцию, определенную пользователем (UDF), в базе данных. Внешние функции должны быть объявлены в каждой базе данных, которая собирается их использовать. Не нужно объявлять UDF, если вы никогда не будете её использовать.

Имя внешней функции должно быть уникальным среди всех имён функций. Оно может отличаться от имени функции указанной в аргументе ENTRY_POINT.

Входные параметры функции перечисляются через запятую сразу после имени функции. Для каждого параметра указывается SQL тип данных. Помимо SQL типов можно указать тип CSTRING. В этом случае параметр является нуль терминальной строкой с максимальной длиной length байт. Существует несколько механизмов передачи параметра из движка Firebird во внешнюю функцию, каждый из этих механизмов будет рассмотрен отдельно.

По умолчанию входные параметры передаются по ссылке. Не существует отдельного предложения для явного указания, что параметр передаётся по ссылке.

При передаче NULL значения по ссылке оно преобразуется в эквивалент нуля, например, число 0 или пустую строку. Если после указанного параметра указано ключевое слово NULL, то при передаче значение NULL оно попадёт в функцию в виде нулевого указателя (NULL).

Обратите внимание на то, что объявление функции с ключевым словом NULL не гарантирует вам, что эта функция правильно обработает входной параметр со значением NULL. Любая функция должна быть написана или переписана таким образом, чтобы правильно обрабатывать значения NULL. Всегда смотрите и используйте объявления функции, предоставленные её
Если указано предложение BY DESCRIPTOR, то входной параметр передаётся по дескриптору. В этом случае параметр UDF получит указатель на внутреннюю структуру, известную как дескриптор, несущую информацию о типе данных, подтипе, точности, наборе символов и сортировке, масштабе, указателе на сами данные и некоторой информации о типе данных, подтипе, точности, наборе символов и сортировке, масштабе, указателе на сами данные и некоторых флагах, в том числе NULL индикаторе. Отметим, что это объявление работает только в том случае, если внешняя функция написана с использованием дескриптора.

Предложение BY SCALAR_ARRAY используется при передаче массивов в качестве входных параметров. В отличие от других типов, вы не можете вернуть массив из UDF.

Обязательное предложение RETURNS описывает выходной параметр возвращаемый функцией. Функция всегда возвращает только один параметр. Выходной параметр может быть любым SQL типом (кроме массива и элемента массива) или нуль терминальной строкой (CSTRING).

Выходной параметр может быть передан по ссылке, по дескриптору или по значению. По умолчанию выходной параметр передаётся по ссылке. Если указано предложение BY DESCRIPTOR, то выходной параметр передаётся по дескриптору. Если указано предложение BY VALUE, то выходной параметр передаётся по значению.

Ключевое слово PARAMETER указывает, что функция возвращает значение из параметра с номером param_num. Такая необходимость возникает, если необходимо возвращать значение типа BLOB.

Ключевое слово FREE_IT означает, что память, выделенная для хранения возвращаемого значения, будет освобождена после завершения выполнения функции. Применяется только в том случае, если эта память в UDF выделялась динамически. В такой UDF память должна выделяться при помощи функции ib_util malloc из модуля ib_util. Это необходимо для совместимости функций выделения и освобождения памяти используемого в коде Firebird и коде UDF.

Предложение ENTRY_POINT указывает имя точки входа (имя экспортируемой функции) в модуле.

Предложение MODULE_NAME задаёт имя модуля, в котором находится экспортируемая функция. В ссылке на модуль может отсутствовать полный путь и расширение файла. Это позволяет легче переносить базу данных между различными платформами. По умолчанию динамические библиотеки пользовательских функций должны располагаться в папке UDF корневого каталога сервера. Параметр UDFAccess в файле firebird.conf позволяет изменить ограничения доступа к библиотекам внешних функций.

Кто может объявить внешнюю функцию?

Выполнить оператор DECLARE EXTERNAL FUNCTION могут:
• Администраторы
• Пользователи с привилегией CREATE FUNCTION.

Пользователь, объявивший внешнюю функцию, становится её владельцем.

Примеры

Example 186. Объявление внешней функции с передачей входных и выходных параметров по ссылке

```
DECLARE EXTERNAL FUNCTION addDay
TIMESTAMP, INT
RETURNS TIMESTAMP
ENTRY_POINT 'addDay' MODULE_NAME 'fbudf';
```

Example 187. Объявление внешней функции с передачей входных и выходных параметров по дескриптору

```
DECLARE EXTERNAL FUNCTION invl
INT BY DESCRIPTOR, INT BY DESCRIPTOR
RETURNS INT BY DESCRIPTOR
ENTRY_POINT 'idNvl' MODULE_NAME 'fbudf';
```

Example 188. Объявление внешней функции с передачей входных параметров по ссылке, выходных по значению

```
DECLARE EXTERNAL FUNCTION isLeapYear
TIMESTAMP
RETURNS INT BY VALUE
ENTRY_POINT 'isLeapYear' MODULE_NAME 'fbudf';
```

Example 189. Объявление внешней функции с передачей входных и выходных параметров по дескриптору. В качестве выходного параметра используется второй параметр функции.

```
DECLARE EXTERNAL FUNCTION i64Truncate
NUMERIC(18) BY DESCRIPTOR, NUMERIC(18) BY DESCRIPTOR
RETURNS PARAMETER 2
ENTRY_POINT 'fbtruncate' MODULE_NAME 'fbudf';
```

См. также:

ALTER EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION, CREATE FUNCTION.
5.12.2. ALTER EXTERNAL FUNCTION

Назначение
Изменение точки входа и/или имени модуля для функции определённой пользователем (UDF).

Доступно в
DSQL

Синтаксис

```
ALTER EXTERNAL FUNCTION funcname
[ENTRY_POINT 'new_entry_point']
[MODULE_NAME 'new_library_name'];
```

Таблица 58. Параметры оператора ALTER EXTERNAL FUNCTION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>funcname</td>
<td>Имя внешней функции.</td>
</tr>
<tr>
<td>new_entry_point</td>
<td>Новое имя экспортируемой функции (точки входа).</td>
</tr>
<tr>
<td>new_library_name</td>
<td>Новое имя модуля, в котором расположена функция.</td>
</tr>
</tbody>
</table>

Оператор ALTER EXTERNAL FUNCTION изменяет точку вход и/или имя модуля для функции определённой пользователем (UDF). При этом существующие зависимости сохраняются.

Предложение ENTRY_POINT позволяет указать новую точку входа (имя экспортируемой функции).

Предложение MODULE_NAME позволяет указать новое имя модуля, в котором расположена экспортируемая функция.

Кто может изменить внешнюю функцию?
Выполнить оператор ALTER EXTERNAL FUNCTION могут:

- Администраторы
- Владелец внешней функции;
- Пользователи с привилегией ALTER ANY FUNCTION.

Примеры

Example 190. Изменение точки входа для внешней функции

```
ALTER EXTERNAL FUNCTION invl ENTRY_POINT 'intNvl';
```
Example 191. Изменение имени модуля для внешней функции

```sql
ALTER EXTERNAL FUNCTION invl MODULE_NAME 'fbudf2';
```

См. также:

DECLARE EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION.

5.12.3. DROP EXTERNAL FUNCTION

Назначение
Удаление объявления функции определённой пользователем (UDF) из базы данных.

Доступно в
DSQL, ESQL.

Синтаксис

```sql
DROP EXTERNAL FUNCTION funcname
```

Table 59. Параметры оператора DROP EXTERNAL FUNCTION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>funcname</td>
<td>Имя внешней функции.</td>
</tr>
</tbody>
</table>

Оператор DROP EXTERNAL FUNCTION удаляет объявление функции определённой пользователем из базы данных. Если есть зависимости от внешней функции, то удаления не произойдёт и будет выдана соответствующая ошибка.

Кто может удалить внешнюю функцию?

Выполнить оператор DROP EXTERNAL FUNCTION могут:

- Администраторы
- Владелец внешней функции;
- Пользователи с привилегией DROP ANY FUNCTION.

Примеры

Example 192. Удаление внешней функции

```sql
DROP EXTERNAL FUNCTION addDay;
```

См. также:

DECLARE EXTERNAL FUNCTION.
5.13. FILTER

BLOB фильтр — объект базы данных, являющийся, по сути, специальным видом внешних функций с единственным назначением: получение объекта BLOB одного формата и преобразования его в объект BLOB другого формата. Форматы объектов BLOB задаются с помощью подтипов BLOB.

Внешние функции для преобразования BLOB типов хранятся в динамических библиотеках и загружаются по необходимости.

Подробнее о подтипах BLOB см. в разделе Бинарные типы данных.

5.13.1. DECLARE FILTER

Назначение
Объявление в базе данных BLOB фильтра.

Доступно в
DSQL, ESQL

Синтаксис

DECLARE FILTER filtername
  INPUT_TYPE <sub_type> OUTPUT_TYPE <sub_type>
  ENTRY_POINT 'function_name' MODULE_NAME 'library_name';

Table 60. Параметры оператора DECLARE FILTER

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>filtername</td>
<td>Имя фильтра. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>sub_type</td>
<td>Подтип BLOB. См. Подтипы BLOB.</td>
</tr>
<tr>
<td>number</td>
<td>Номер подтипа BLOB. См. Подтипы BLOB.</td>
</tr>
<tr>
<td>mnemonic</td>
<td>Мнемоника подтипа BLOB. См. Подтипы BLOB.</td>
</tr>
<tr>
<td>function_name</td>
<td>Имя экспортируемой функции (точка входа).</td>
</tr>
<tr>
<td>library_name</td>
<td>Имя модуля, в котором расположен фильтр.</td>
</tr>
<tr>
<td>user_defined</td>
<td>Определяемая пользователем мнемоника подтипа BLOB.</td>
</tr>
</tbody>
</table>

Оператор DECLARE FILTER делает доступным BLOB фильтр в базе данных. Имя BLOB фильтра должно быть уникальным среди имён BLOB фильтров.
**Задание подтипов**

Подтип может быть задан в виде номера подтипа или мнемоники подтипа. Пользовательские подтипы должны быть представлены отрицательными числами (от -1 до -32768). Не допускается создание двух и более фильтров BLOB с одинаковыми комбинациями входных и выходных типов. Объявление фильтра с уже существующими комбинациями входных и выходных типов BLOB приведёт к ошибке.

Предложение `INPUT_TYPE` идентифицирует тип преобразуемого объекта (подтип BLOB).

Предложение `OUTPUT_TYPE` идентифицирует тип создаваемого объекта.

Если вы хотите определить мнемоники для собственных подтипов BLOB, вы можете добавить их в системную таблицу `RDB$TYPES`, как показано ниже. После подтверждения транзакции мнемоники могут быть использованы для декларации при создании новых фильтров.

```sql
INSERT INTO RDB$TYPES (RDB$FIELD_NAME, RDB$TYPE, RDB$TYPE_NAME)
VALUES ('RDB$FIELD_SUB_TYPE', -33, 'MIDI');
```

Значение поля `rdb$field_name` всегда должно быть 'RDB$FIELD_SUB_TYPE'. Если вы определяете мнемоники в верхнем регистре, то можете использовать их без учёта регистра и без кавычек при объявлении фильтра.

### Параметры `DECLARE FILTER`

Предложение `ENTRY_POINT` указывает имя точки входа (имя экспортируемой функции) в модуле.

Предложение `MODULE_NAME` задаёт имя модуля, в котором находится экспортируемая функция. По умолчанию модули должны располагаться в папке UDF корневого каталога сервера. Параметр `UDFAccess` в файле `firebird.conf` позволяет изменить ограничения доступа к библиотекам фильтров.

### Кто может создать BLOB фильтр?

Выполнить оператор `DECLARE FILTER` могут:

- Администраторы
- Пользователи с привилегией `CREATE FILTER`.

Пользователь, создавший BLOB фильтр, становится его владельцем.

### Примеры

**Example 193. Создание BLOB фильтра с использованием номеров подтипов**

```sql
DECLARE FILTER DESC_FILTER
```
Example 194. Создание BLOB фильтра с использованием мнемоник подтипов

```sql
DECLARE FILTER FUNNEL
INPUT_TYPE blr OUTPUT_TYPE text
ENTRY_POINT 'blr2asc' MODULE_NAME 'myfilterlib';
```

См. также:

DROP FILTER.

5.13.2. DROP FILTER

Назначение
Удаление объявления BLOB фильтра.

Доступно в
DSQL, ESQL

Синтаксис

```
DROP FILTER filtername
```

Table 61. Параметры оператора DROP FILTER

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>filtername</td>
<td>Имя BLOB фильтра.</td>
</tr>
</tbody>
</table>

Оператор DROP FILTER удаляет объявление BLOB фильтра из базы данных. Удаление BLOB фильтра из базы данных делает его не доступным из базы данных, при этом динамическая библиотека, в которой расположена функция преобразования, остаётся не тронутой.

Кто может удалить BLOB фильтр?

Выполнить оператор DROP FILTER могут:

- Администраторы
- Владелец BLOB фильтра;
- Пользователи с привилегией DROP ANY FILTER.
Примеры

Example 195. Удаление BLOB фильтра

```sql
DROP FILTER DESC_FILTER;
```

См. также:

DECLARE FILTER.

5.14. SEQUENCE (GENERATOR)

Последовательность (sequence) или генератор (generator)—объект базы данных, предназначенный для получения уникального числового значения. Термин последовательность является SQL совместимым. Ранее в Interbase и Firebird последовательности называли генераторами.

Независимо от диалекта базы данных последовательности (или генераторы) всегда хранятся как 64-битные целые значения.

Если клиент использует 1 диалект, то сервер передаёт ему значения последовательности, усечённые до 32-битного значения. Если значение последовательности передаются в 32-разрядное поле или переменную, то до тех пор, пока текущее значение последовательности не вышло за границы для 32-битного числа, ошибок не будет. В момент выхода значения последовательности за этот диапазон база данных 3-го диалекта выдаст сообщение об ошибке, а база данных 1-ого диалекта будет молча обрезать значения, что также может привести к ошибке—например, если поле, заполняемое генератором, является первичным или уникальным.

В данном разделе описываются вопросы создания, модификации (установка значения последовательности) и удаления последовательностей.

5.14.1. CREATE SEQUENCE

Назначение

Создание новой последовательности (генератора).

Доступно в

DSQL, ESQL

Синтаксис

```
CREATE {SEQUENCE | GENERATOR} seq_name
[START WITH start_value] [INCREMENT [BY] increment]
```

Table 62. Параметры оператора CREATE SEQUENCE
Таблица:

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_name</td>
<td>Имя последовательности (генератора). Может содержать до 63 символов.</td>
</tr>
<tr>
<td>start_value</td>
<td>Начальное значение последовательности (генератора). По умолчанию равно 1.</td>
</tr>
<tr>
<td>increment</td>
<td>Шаг приращения. 4 байтовое целое число. По умолчанию равно 1.</td>
</tr>
</tbody>
</table>

Оператор `CREATE SEQUENCE` создаёт новую последовательность. Слова `SEQUENCE` и `GENERATOR` являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать `SEQUENCE`.

В момент создания последовательности ей устанавливается значение, указанное в необязательном предложении `START WITH` минус значение приращения указанное в предложении `INCREMENT [BY]`. Если предложение `STARTING WITH` отсутствует, то последовательности устанавливается значение равное 1. Таким образом, если начальное значение последовательности равно 100, а приращение 10, то первое значение выданное оператором `NEXT VALUE FOR` будет равно 100.

До Firebird 4.0, первое значение выданное оператором `NEXT VALUE FOR` было равно 110.

Необязательное предложение `INCREMENT [BY]` позволяет задать шаг приращения для оператора `NEXT VALUES FOR`. По умолчанию шаг приращения равен единице. Приращение не может быть установлено в ноль для пользовательских последовательностей. Значение последовательности изменяется также при обращении к функции `GEN_ID`, где в качестве параметра указывается имя последовательности и значение приращения, которое может быть отлично от указанного в предложении `INCREMENT BY`.

Кто может создать последовательность?

Выполнить оператор `CREATE SEQUENCE (CREATE GENERATOR)` могут:

- Администраторы
- Пользователи с привилегией `CREATE SEQUENCE (CREATE GENERATOR)`.

Пользователь, создавший последовательность, становится её владельцем.

Примеры

*Example 196. Создание последовательности*

Создание последовательности `EMP_NO_GEN` с начальным значением 0 и шагом приращения равным единице.

```
CREATE SEQUENCE EMP_NO_GEN;
```
Example 197. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 5 и шагом приращения равным единице.

```
CREATE SEQUENCE EMP_NO_GEN START WITH 5;
```

Example 198. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 1 и шагом приращения равным 10.

```
CREATE SEQUENCE EMP_NO_GEN INCREMENT BY 10;
```

Example 199. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 5 и шагом приращения равным 10.

```
CREATE SEQUENCE EMP_NO_GEN START WITH 5 INCREMENT BY 10;
```

См. также:

ALTER SEQUENCE, SET GENERATOR, DROP SEQUENCE, NEXT VALUE FOR, GEN_ID.

5.14.2. ALTER SEQUENCE

Назначение

Изменение последовательности (генератора).

Доступно в

DSQL, ESQL

Синтаксис

```
ALTER {SEQUENCE | GENERATOR} seq_name
[RESTART [WITH newvalue]]
[INCREMENT [BY] increment]
```

Table 63. Параметры оператора ALTER SEQUENCE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_name</td>
<td>Имя последовательности (генератора).</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>newvalue</td>
<td>Новое значение последовательности (генератора). 64 битное целое в диапазоне от $-2^{63}$ до $2^{63} - 1$</td>
</tr>
<tr>
<td>increment</td>
<td>Шаг приращения. Не может быть равным 0.</td>
</tr>
</tbody>
</table>

Оператор `ALTER SEQUENCE` устанавливает значение последовательности или генератора в заданное значение и/или изменяет значение приращения.

Предложение `RESTART WITH` позволяет установить значение последовательности. Предложение `RESTART` может быть использовано самостоятельно (без `WITH`) для перезапуска значения последовательности с того значения с которого был начат старт генерации значений или предыдущий рестарт.

Неосторожное использование оператора `ALTER SEQUENCE` (изменение значения последовательности или генератора) может привести к нарушению логической целостности данных.

Предложение `INCREMENT [BY]` позволяет изменить шаг приращения последовательности для оператора `NEXT VALUES FOR`.

Изменение значения приращения — это возможность, которая вступает в силу для каждого запроса, который запускается после фиксаций изменения. Процедуры, которые вызваны впервые после изменения приращения, будут использовать новое значение, если они будут содержать операторы `NEXT VALUE FOR`. Процедуры, которые уже работают, не будут затронуты, потому что они кэшируются. Процедуры, использующие `NEXT VALUE FOR`, не должны быть перекомпилированы, чтобы видеть новое приращение, но если они уже работают или загружены, то никакого эффекта не будет. Конечно процедуры, использующие `gen_id(gen, <expression>)`, не затронут при изменении приращения.

**Кто может изменить последовательность?**

Выполнить оператор `ALTER SEQUENCE (ALTER GENERATOR)` могут:

- Администраторы
- Владелец последовательности (генератора);
- Пользователи с привилегией `ALTER ANY SEQUENCE (ALTER ANY GENERATOR)`.

**Примеры**

*Example 200. Изменение последовательности*

Установка для последовательности `EMP_NO_GEN` значения 145.

```sql
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
```
Example 201. Изменение последовательности

Сброс значения последовательности в то, которое было установлено при создании последовательности (или при предыдущей установке значения).

```
ALTER SEQUENCE EMP_NO_GEN RESTART;
```

Example 202. Изменение последовательности

Изменение значения приращения последовательности EMP_NO_GEN.

```
ALTER SEQUENCE EMP_NO_GEN INCREMENT BY 10;
```

См. также:
SET GENERATOR, CREATE SEQUENCE, DROP SEQUENCE, NEXT VALUE FOR, GEN_ID.

5.14.3. CREATE OR ALTER SEQUENCE

Назначение

Создание новой или изменение существующей последовательности (генератора).

Доступно в
DSQL, ESQL

Синтаксис

```
CREATE OR ALTER {SEQUENCE | GENERATOR} seq_name
[{{START WITH start_value | RESTART}}]
[INCREMENT [BY] increment]
```

Таблица 64. Параметры оператора CREATE OR ALTER SEQUENCE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_name</td>
<td>Имя последовательности (генератора). Может содержать до 63 символов.</td>
</tr>
<tr>
<td>start_value</td>
<td>Начальное значение последовательности (генератора). По умолчанию равно 1.</td>
</tr>
<tr>
<td>increment</td>
<td>Шаг приращения. 4 байтовое целое число. По умолчанию равно 1.</td>
</tr>
</tbody>
</table>

Если последовательности не существует, то она будет создана. Уже существующая последовательность будет изменена, при этом существующие зависимости последовательности будут сохранены.

⚠️ Оператор CREATE OR ALTER SEQUENCE требует, чтобы хотя бы одно из
необязательных предложений было указано.

Примеры

Example 203. Создание новой или изменение существующей последовательности

```
CREATE OR ALTER SEQUENCE EMP_NO_GEN
START WITH 10
INCREMENT BY 1;
```

См. также:
CREATE SEQUENCE, ALTER SEQUENCE, SET GENERATOR.

5.14.4. DROP SEQUENCE

Назначение
Удаление последовательности (генератора).

Доступно в
DSQL, ESQL

Синтаксис

```
DROP {SEQUENCE | GENERATOR} seq_name
```

Table 65. Параметры оператора DROP SEQUENCE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_name</td>
<td>Имя последовательности (генератора).</td>
</tr>
</tbody>
</table>

Оператор DROP SEQUENCE удаляет существующую последовательность (генератор). Слова `SEQUENCE` и `GENERATOR` являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать `SEQUENCE`. При наличии зависимостей для существующей последовательности (генератора) удаления не будет выполнено.

Кто может удалить генератор?

Выполнить оператор DROP SEQUENCE (DROP GENERATOR) могут:

- Администраторы
- Владелец последовательности (генератора);
- Пользователи с привилегией DROP ANY SEQUENCE (DROP ANY GENERATOR).

Примеры
5.14.5. RECREATE SEQUENCE

Назначение

Создание или пересоздание последовательности (генератора).

Доступно в

DSQL, ESQL

Синтаксис

RECREATE {SEQUENCE | GENERATOR} seq_name
[START WITH start_value]
[INCREMENT [BY] increment];

Table 66. Параметры оператора RECREATE SEQUENCE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_name</td>
<td>Имя последовательности (генератора). Может содержать до 63 символов.</td>
</tr>
<tr>
<td>start_value</td>
<td>Начальное значение последовательности (генератора).</td>
</tr>
<tr>
<td>increment</td>
<td>Шаг приращения. 4 байтное целое число.</td>
</tr>
</tbody>
</table>

Оператор RECREATE SEQUENCE создаёт или пересоздаёт последовательность (генератор). Если последовательность с таким именем уже существует, то оператор RECREATE SEQUENCE попытается удалить её и создать новую последовательность. При наличии зависимостей для существующей последовательности оператор RECREATE SEQUENCE не выполнится.

Примеры

Example 205. Пересоздание последовательности

RECREATE SEQUENCE EMP_NO_GEN
START WITH 10
INCREMENT BY 1;
5.14.6. SET GENERATOR

Назначение
Устанавливает значение последовательности или генератора в заданное значение.

Доступно в
DSQL, ESQL

Синтаксис

SET GENERATOR seq_name TO new_val

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_name</td>
<td>Имя последовательности (генератора).</td>
</tr>
<tr>
<td>new_val</td>
<td>Новое значение последовательности (генератора). 64 битное целое в диапазоне от $-2^{63}$ до $2^{63} - 1$</td>
</tr>
</tbody>
</table>

Оператор SET GENERATOR устанавливает значение последовательности или генератора в заданное значение.

Оператор SET GENERATOR считается устаревшим и оставлен ради обратной совместимости. В настоящее время вместо него рекомендуется использовать стандарт-совместимый оператор ALTER SEQUENCE.

Неосторожное использование оператора SET GENERATOR (изменение значения последовательности или генератора) может привести к потере логической целостности данных.

Кто может изменить значение генератора?

Выполнить оператор SET GENERATOR могут:

- Администраторы
- Владелец последовательности (генератора);
- Пользователи с привилегией ALTER ANY SEQUENCE (ALTER ANY GENERATOR).

Примеры

Example 206. Установка значения для последовательности

```
SET GENERATOR EMP_NO_GEN TO 145;
```

То же самое можно сделать, используя оператор ALTER SEQUENCE.
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;

См. также:
ALTER SEQUENCE, NEXT VALUE FOR, GEN_ID.

5.15. EXCEPTION

Пользовательское исключение (exception) — объект базы данных, описывающий сообщение об ошибке. Исключение можно вызывать и обрабатывать в PSQL коде (см. EXCEPTION, WHEN ... DO).

В данном разделе описываются операторы создания, модификации и удаления исключений.

5.15.1. CREATE EXCEPTION

Назначение

Создание пользовательского исключения для использования в PSQL модулях.

Доступно в
DSQL, ESQL

Синтаксис

```
CREATE EXCEPTION exception_name '<message>'

<message> ::= <message-part> [<message-part> ...]

<message-part> ::= <text>  
| @<slot>

<slot> ::= one of 1..9
```

Table 68. Параметры оператора CREATE EXCEPTION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception_name</td>
<td>Имя исключения. Максимальная длина 63 символов.</td>
</tr>
<tr>
<td>message</td>
<td>Сообщение об ошибке. Максимальная длина ограничена 1021 символом.</td>
</tr>
<tr>
<td>text</td>
<td>Текст.</td>
</tr>
<tr>
<td>slot</td>
<td>Номер слота для параметра. Нумерация начинается с 1. Максимальный номер слота равен 9.</td>
</tr>
</tbody>
</table>

Оператор CREATE EXCEPTION создаёт новое пользовательское исключение для использования в PSQL модулях. Исключение должно отсутствовать в базе данных, иначе будет выдана
соответствующая ошибка.

Имя исключения является стандартным идентификатором. В диалекте 3 оно может быть заключено в двойные кавычки, что делает его чувствительным к регистру. Подробности см. Идентификаторы.

Сообщение исключения сохраняется в наборе символов NONE, т.е. любых символов из однобайтовых наборов символов. Текст сообщения может быть переопределён в PSQL коде во время возбуждения исключения.

Сообщение об ошибке может содержать слоты для параметров, которые заполняются при возбуждении исключения.

Внимание!
Если в тексте сообщения, встретится номер слота параметра больше 9, то второй и последующий символ будут восприняты как литералы. Например, @10 будет воспринято как @1, после которого следует литерал 0.

Пользовательские исключения хранятся в таблице RDB$EXCEPTION.

Кто может создать исключение?
Выполнить оператор CREATE EXCEPTION могут:

• Администраторы
• Пользователи с привилегией CREATE EXCEPTION.

Пользователь, создавший исключение, становится его владельцем.

Примеры

Example 207. Создание пользовательского исключения

```sql
CREATE EXCEPTION E_LARGE_VALUE 'Значение превышает предельно допустимое';
```

Example 208. Создание параметризованного исключения

```sql
CREATE EXCEPTION E_INVALID_VALUE
'Неверное значение @1 для поля @2';
```

См. также:
ALTER EXCEPTION, CREATE OR ALTER EXCEPTION, DROP EXCEPTION, RECREATE EXCEPTION, EXCEPTION.
5.15.2. ALTER EXCEPTION

Назначение
Изменение текста сообщения пользовательского исключения.

Доступно в
DSQL, ESQl

Синтаксис

```
ALTER EXCEPTION exception_name '<message>'
```

Подробное см. в синтаксисе CREATE EXCEPTION

Оператор ALTER EXCEPTION изменяет текст сообщения пользовательского исключения.

Кто может изменить исключение?

Выполнить оператор ALTER EXCEPTION могут:

- Администраторы
- Владелец исключения;
- Пользователи с привилегией ALTER ANY EXCEPTION.

Примеры

Example 209. Изменение текста сообщения пользовательского исключения

```
ALTER EXCEPTION E_LARGE_VALUE 'Значение превышает максимально допустимое';
```

См. также:
CREATE EXCEPTION, CREATE OR ALTER EXCEPTION, RECREATE EXCEPTION.

5.15.3. CREATE OR ALTER EXCEPTION

Назначение
Создание нового или изменение существующего исключения.

Доступно в
DSQL

Синтаксис

```
CREATE OR ALTER EXCEPTION exception_name '<message>'
```

Подробное см. в синтаксисе CREATE EXCEPTION
Если исключения не существует, то оно будет создано. Уже существующее исключение будет изменено, при этом существующие зависимости исключения будут сохранены.

Примеры

Example 210. Создание или изменение пользовательского исключения

```sql
CREATE OR ALTER EXCEPTION E_LARGE_VALUE
'Значение превышает максимально допустимое';
```

См. также:
CREATE EXCEPTION, ALTER EXCEPTION, RECREATE EXCEPTION.

5.15.4. DROP EXCEPTION

Назначение
Удаление пользовательского исключения.

Доступно в
DSQL, ESQL

Синтаксис

```
DROP EXCEPTION exception_name
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception_name</td>
<td>Имя исключения.</td>
</tr>
</tbody>
</table>

Оператор DROP EXCEPTION удаляет пользовательское исключение. При наличии зависимостей для существующего исключения удаления не будет выполнено.

Кто может удалить исключение?

Выполнить оператор DROP EXCEPTION могут:

- Администраторы
- Владелец исключения;
- Пользователи с привилегией DROP ANY EXCEPTION.

Примеры
Example 211. Удаление пользовательского исключения

```sql
DROP EXCEPTION E_LARGE_VALUE;
```

См. также:
CREATE EXCEPTION, RECREATE EXCEPTION.

5.15.5. RECREATE EXCEPTION

Назначение
Создание или пересоздание пользовательского исключения.

Доступно в
DSQL

Синтаксис

```sql
RECREATE EXCEPTION exception_name '<message>'
```

Подробнее см. в синтаксисе CREATE EXCEPTION

Оператор RECREATE EXCEPTION создаёт или пересоздаёт пользовательское исключение. Если исключение с таким именем уже существует, то оператор RECREATE EXCEPTION попытается удалить его и создать новое исключение. При наличии зависимостей для существующего исключения оператор RECREATE EXCEPTION не выполнится.

Примеры

Example 212. Создание или пересоздание пользовательского исключения

```sql
RECREATE EXCEPTION E_LARGE_VALUE
    'Значение превышает максимально допустимое';
```

См. также:
CREATE EXCEPTION, ALTER EXCEPTION, CREATE OR ALTER EXCEPTION.

5.16. COLLATION

В SQL текстовые строки принадлежат к сортируемым объектам. Это означает, что они подчиняются своим внутренним правилам упорядочения, например, алфавитному порядку. К таким текстовым строкам можно применять операции сравнения (например, “меньше чем” или “больше чем”), при этом значения выражения должны вычисляться согласно определённой последовательности сортировки. Например, выражение 'a'>'b' означает, что 'a' предшествует 'b' в последовательности сортировки. Под выражением 'c'>'b' имеется в
виду, что в последовательности сортировки 'с' определено после 'б'. Текстовые строки, включающие больше одного символа, сортируются путём последовательного сравнения символов: сначала сравниваются первые символы двух строк, затем вторые символы и так далее, до тех пор, пока не будет найдено различие между двумя строками. Такое различие управляет порядком сортировки.

Под сравнением (сортировкой) (COLLATION) принято понимать такой объект схемы, который определяет упорядочивающую последовательность (или последовательность сортировки).

5.16.1. CREATE COLLATION

Назначение
Добавление новой сортировки (сравнения) для набора символов поддерживаемого в базе данных.

Доступно в
DSQL

Синтаксис

```
CREATE COLLATION collname
   FOR charset
   [FROM basecoll | FROM EXTERNAL ('extname')]
   [NO PAD | PAD SPACE]
   [CASE [IN]SENSITIVE]
   [ACCENT [IN]SENSITIVE]
   ['<specific-attributes>'];

<specific-attributes> ::= <attribute> [; <attribute> ...]

<attribute> ::= attrname=attrvalue
```

Table 70. Параметры оператора CREATE COLLATION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>collname</td>
<td>Имя сортировки (сравнения). Максимальная длина 63 символов.</td>
</tr>
<tr>
<td>charset</td>
<td>Набор символов.</td>
</tr>
<tr>
<td>basecoll</td>
<td>Базовая сортировка (сравнение).</td>
</tr>
<tr>
<td>extname</td>
<td>Имя сортировки из конфигурационного файла. Чувствительно к регистру.</td>
</tr>
</tbody>
</table>

Оператор CREATE COLLATION ничего не "создаёт", его целью является сделать сортировку известной для базы данных. Сортировка уже должна присутствовать в системе, как правило, в файле библиотеки, и должна быть зарегистрирована в файле fbintl.conf подкаталога intl корневой директории Firebird.

Необязательное предложение FROM указывает сортировку, на основе которой будет создана
новая сортировка. Такая сортировка должна уже присутствовать в базе данных. Если указано ключевое слово EXTERNAL, то будет осуществлён поиск сортировки из файла $fbroot/intl/fbintl.conf, при этом extname должно в точности соответствовать имени в конфигурационном файле (чувствительно к регистру).

Если предложение FROM отсутствует, то Firebird ищет в конфигурационном файле fbintl.conf подкаталога intl корневой директории сервера сортировку с именем, указанным сразу после CREATE COLLATION. Другими словами, отсутствие предложения FROM basecoll эквивалентно заданию FROM EXTERNAL (‘collname’).

При создании сортировки можно указать учитываются ли конечные пробелы при сравнении. Если указана опция NO PAD, то конечные пробелы при сравнении учитываются. Если указана опция PAD SPACE, то конечные пробелы при сравнении не учитываются.

Необязательное предложение CASE позволяет указать будет ли сравнение чувствительно к регистру.

Необязательное предложение ACCENT позволяет указать будет ли сравнение чувствительно к акцентированным буквам (например “е” и “ё”).

**Специфичные атрибуты**

В операторе CREATE COLLATION можно также указать специфичные атрибуты для сортировки. Ниже в таблице приведён список доступных специфичных атрибутов. Не все атрибуты применимы ко всем сортировкам. Если атрибут не применим к сортировке, но указан при её создании, то это не вызовет ошибки.

<table>
<thead>
<tr>
<th>Имя</th>
<th>Значение</th>
<th>Валидность</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE-COMPRESSIONS</td>
<td>0, 1</td>
<td>1 bpc</td>
<td>Отключает сжатия (инча сокращения). Сжатия заставляют определённые символьные последовательности быть сортированными как атомарные модули, например, испанские c + h как единственный символ ch.</td>
</tr>
<tr>
<td>Имя</td>
<td>Значение</td>
<td>Валидность</td>
<td>Описание</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DISABLE-EXPANSIONS</td>
<td>0, 1</td>
<td>1 bpc</td>
<td>Отключение расширений. Расширения позволяют рассматривать определённые символы (например, лигатуры или гласные умляуты) как последовательности символов и соответственно сортировать.</td>
</tr>
<tr>
<td>ICU-VERSION</td>
<td>default или M.m</td>
<td>UNI</td>
<td>Задаёт версию библиотеки ICU для использования. Допустимые значения определены в соответствующих элементах &lt;intl_module&gt; в файле intl/fbintl.conf. Формат: либо строка “default” или основной и дополнительный номер версии, как “3.0” (оба без кавычек).</td>
</tr>
<tr>
<td>LOCALE</td>
<td>xx_YY</td>
<td>UNI</td>
<td>Задаёт параметры сортировки языкового стандарта. Требуется полная версия библиотеки ICU. Формат строки: “du_NL” (без кавычек).</td>
</tr>
<tr>
<td>MULTI-LEVEL</td>
<td>0, 1</td>
<td>1 bpc</td>
<td>Использование нескольких уровней сортировки.</td>
</tr>
<tr>
<td>NUMERIC-SORT</td>
<td>0, 1</td>
<td>UNI</td>
<td>Обрабатывает непрерывные группы десятичных цифр в строке как атомарные модули и сортирует их в числовой последовательности (известна как естественная сортировка).</td>
</tr>
<tr>
<td>SPECIALS-FIRST</td>
<td>0, 1</td>
<td>1 bpc</td>
<td>Сортирует специальные символы (пробелы и т.д.) до буквенно-цифровых символов.</td>
</tr>
</tbody>
</table>

Если вы хотите добавить в базу данных новый набор символов с его умалчиваемой сортировкой, то зарегистрируйте и выполните хранимую процедуру `sp_register_character_name(name, max_bytes_per_character)` из подкаталога `misc/intl.sql` установки Firebird. Для нормальной работы с набором символов, он должен присутствовать в вашей операционной системе, и зарегистрирован в файле `fbintl.conf` поддиректории `intl`.

**Кто может создать сортировку?**

Выполнить оператор `CREATE COLLATION` могут:

- **Администраторы**
- **Пользователи с привилегией CREATE COLLATION.**
Пользователь, создавший сортировку, становится её владельцем.

Примеры

Example 213. Создание сортировки с использованием имени, найденном в файле fbintl.conf (чувствительной к регистру символов).

```
CREATE COLLATION ISO8859_1_UNICODE FOR ISO8859_1;
```

Example 214. Создание сортировки с использованием специального (заданного пользователем) названия ("external" имя должно в точности соответствовать имени в файле fbintl.conf).

```
CREATE COLLATION LAT_UNI
FOR ISO8859_1
FROM EXTERNAL ('ISO8859_1_UNICODE');
```

Example 215. Создание не чувствительной к регистру символов сортировки на основе уже присутствующей в базе данных.

```
CREATE COLLATION ES_ES_NOPAD_CI
FOR ISO8859_1
FROM ES_ES
NO PAD
CASE INSENSITIVE;
```

Example 216. Создание не чувствительной к регистру символов сортировки на основе уже присутствующей в базе данных со специфичными атрибутами.

```
CREATE COLLATION ES_ES_CI_COMPR
FOR ISO8859_1
FROM ES_ES
CASE INSENSITIVE
'DISABLE-COMPRESSIONS=0';
```

Example 217. Создание не чувствительной к регистру символов сортировки по значению чисел (так называемой натуральной сортировки).

```
CREATE COLLATION nums_coll FOR UTF8
FROM UNICODE
CASE INSENSITIVE 'NUMERIC-SORT=1';
```
CREATE DOMAIN dm_nums AS varchar(20) CHARACTER SET UTF8 COLLATE nums_coll; -- original (manufacturer) numbers

CREATE TABLE wares(id int primary key, articul dm_nums ...);

См. также:
DROP COLLATION.

5.16.2. DROP COLLATION

Назначение
Удаление существующей сортировки.

Доступно в
DSQL

Синтаксис

DROP COLLATION collname

Table 72. Параметры оператора DROP COLLATION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>collname</td>
<td>Имя сортировки.</td>
</tr>
</tbody>
</table>

Оператор DROP COLLATION удаляет указанную сортировку. Сортировка должна присутствовать в базе данных, иначе будет выдана соответствующая ошибка.

Если вы хотите удалить в базе данных набор символов со всеми его сортировками, то зарегистрируйте и выполните хранимую процедуру sp_unregister_character_set(name) из подкаталога misc/intl.sql установки Firebird.

Кто может удалить сортировку?

Выполнить оператор DROP COLLATION могут:

- Администраторы
- Владелец сортировки;
- Пользователи с привилегией DROP ANY COLLATION.

Примеры
**Example 218. Удаление сортировки**

```sql
DROP COLLATION ES_ES_NOPAD_CI;
```

См. также:

CREATE COLLATION.

### 5.17. CHARACTER SET

#### 5.17.1. ALTER CHARACTER SET

**Назначение**

Установка сортировки по умолчанию для набора символов.

**Доступно в**

DSQL

**Синтаксис**

```sql
ALTER CHARACTER SET charset
SET DEFAULT COLLATION collation
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>Набор символов.</td>
</tr>
<tr>
<td>collation</td>
<td>Сортировка.</td>
</tr>
</tbody>
</table>

Оператор `ALTER CHARACTER SET` изменяет сортировку по умолчанию для указанного набора символов. Это повлияет на использование набора символов в будущем, кроме случаев, когда явно переопределена сортировка `COLLATE`. Сортировка существующих доменов, столбцов и переменных PSQL при этом не будет изменена.

Если сортировка по умолчанию была изменена для набора символов базы данных (тот, что был указан при создании базы данных), то также изменяется и сортировка по умолчанию для базы данных.

Если сортировка по умолчанию была изменена для набора символов, который был указан при подключении, то строковые константы будут интерпретироваться в соответствии с новыми параметрами сортировки (если набор символов и/или сортировка не переопределяются).

**Примеры**
Example 219. Установка сортировки UNICODE_CI_AI по умолчанию для кодировки UTF8

```
ALTER CHARACTER SET UTF8 SET DEFAULT COLLATION UNICODE_CI_AI;
```

5.18. COMMENTS

Объекты базы данных и сама база данных могут содержать примечания. Это удобное средство документирования во время разработки базы данных и её поддержки.

5.18.1. COMMENT ON

Назначение

Документирование метаданных.

Доступно в

DSQL, ESQL

Синтаксис

```
COMMENT ON <object> IS {'sometext' | NULL}
<object> ::= DATABASE
          | <basic-type> objectname
          | USER username [USING PLUGIN plugin_name]
          | COLUMN relationname.fieldname
          | [PROCEDURE | FUNCTION] PARAMETER
          | [package_name.] routinename.paramname
          | {PROCEDURE | [EXTERNAL] FUNCTION}
          routinename

<basic-type> ::= CHARACTER SET
          | COLLATION
          | DOMAIN
          | EXCEPTION
          | FILTER
          | GENERATOR
          | INDEX
          | PACKAGE
          | ROLE
          | SEQUENCE
          | TABLE
          | TRIGGER
          | VIEW
```

Table 74. Параметры оператора COMMENT ON
Оператор COMMENT ON добавляет комментарии для объектов базы данных (метаданных). Комментарии при этом сохраняются в текстовые поля RDB$DESCRIPTION типа BLOB соответствующей системной таблицы (из этих полей клиентское приложение может просмотреть комментарии).

При добавлении комментария для пользователя вы можете уточнить в каком плагине управления пользователями он находится с помощью необязательного предложения USING PLUGIN. Если это предложение отсутствует, то предполагает что пользователь создан в плагине управления по умолчанию, то есть первого плагина указанного в параметре UserManager в файле firebird.conf или databases.conf.

Если вы вводите пустой комментарий (""), то он будет сохранен в базе данных как NULL.

**Кто может добавить комментарий?**

Выполнить оператор COMMENT ON могут:

- Администраторы
- Владелец объекта, для которого добавляется комментарий;
- Пользователи с привилегией ALTER ANY <object_type>.

**Примеры**

*Example 220. Добавление комментария для текущей базы данных.*

```
COMMENT ON DATABASE IS 'Это тестовая ("\'my.fdb\'\') БД';
```
Example 221. Добавление комментария для таблицы.

```
COMMENT ON TABLE METALS IS 'Справочник металлов';
```

Example 222. Добавление комментария для поля таблицы.

```
COMMENT ON COLUMN METALS.ISALLOY IS '0 = чистый металл, 1 = сплав';
```

Example 223. Добавление комментария для параметра процедуры.

```
COMMENT ON PARAMETER ADD_EMP_PROJ.EMP_NO IS 'Код сотрудника';
```

Example 224. Добавление комментария для пакета, его процедур и функций, и их параметров.

```
COMMENT ON PACKAGE APP_VAR IS 'Переменные приложения';

COMMENT ON FUNCTION APP_VAR.GET_DATEBEGIN IS 'Возвращает дату начала периода';

COMMENT ON PROCEDURE APP_VAR.SET_DATERANGE IS 'Установка диапазона дат';

COMMENT ON PROCEDURE PARAMETER APP_VAR.SET_DATERANGE.ADATEBEGIN IS 'Дата начала';
```

Example 225. Добавление комментария для пользователя.

```
COMMENT ON USER BOB35 IS 'Это Боб из плагина по умолчанию';

COMMENT ON USER JHON USING PLUGIN Legacy_UserManager IS 'Это Джон из плагина Legacy_UserManager';
```
Chapter 6. Операторы обработки данных (DML)

6.1. SELECT

Назначение
Выборка данных

Доступно в
DSQL, ESQL, PSQL

Синтаксис

```sql
WITH [RECURSIVE] <cte> [, <cte> ...]
SELECT
[FIRST m] [SKIP n]
[DISTINCT | ALL] <columns>
FROM
<source> [[AS] alias]
[joins]
[WHERE <condition>]
[GROUP BY <grouping-list>]
[HAVING <aggregate-condition>]
[WINDOW <window definition> [, <window definition>] ...]
[PLAN <plan-expr>]
[UNION [DISTINCT | ALL] <other-select>]
[ORDER BY <ordering-list>]
[   {ROWS m [TO n]}
    | {OFFSET n {ROW | ROWS}}
    | FETCH {FIRST | NEXT} [m] {ROW | ROWS} ONLY]

FOR UPDATE [OF <columns>]
[WITH LOCK]
INTO <variables>

:variables ::= varname varname ...
```

Описание

Оператор (команда) SELECT извлекает данные из базы данных и передаёт их в приложение или в вызывающую SQL команду. Данные возвращаются в виде набора строк (которых может быть 0 или больше), каждая строка содержит один или более столбцов или полей. Совокупность возвращаемых строк является результирующим набором данных команды.

Следующие части команды SELECT являются обязательными:

- Ключевое слово SELECT, за которым следует список полей. Эта часть определяет, что
запрашивается из базы данных;

- Ключевое слово FROM, за которым следует объект выборки (например, таблица). Эта часть сообщает серверу, где следует искать запрашиваемые данные.

В простейшей форме SELECT извлекает ряд полей из единственной таблицы, например:

```
SELECT id, name, address
FROM contacts
```

Или, для того чтобы извлечь все поля таблицы:

```
SELECT * FROM contacts
```

На практике команда SELECT обычно выполняется с выражением WHERE, которое ограничивает возвращаемый набор данных. Также, полученный набор данных обычно сортируется с помощью выражения ORDER BY, дополнительно ограничивается (с целью организации постраничного просмотра данных) выражениями FIRST … SKIP, OFFSET … FETCH или ROWS.

Список полей может содержать различные типы выражений вместо имен полей, а источник необязательно должен быть таблицей или представлением, он так же может быть производной таблицей (derived table), общим табличным выражением (CTE) или селективной хранимой процедурой.

Несколько источников данных могут быть соединены с помощью выражения JOIN, и несколько результирующих наборов данных могут быть скомбинированы с использованием выражения UNION.

В следующих секциях мы подробно рассмотрим все выражения для команды SELECT и их использование.

### 6.1.1. FIRST, SKIP

**Назначение**

Получение части строк из упорядоченного набора.

**Синтаксис**

```
SELECT [FIRST <m>] [SKIP <n>]
FROM ...
...

<m>, <n> ::=  
  <integer-literal>  
  | <query-parameter>  
  | (<integer-expression>)
```
Table 75. Параметры предложений FIRST и SKIP

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer-literal</td>
<td>Целочисленный литерал.</td>
</tr>
<tr>
<td>query-parameter</td>
<td>Параметр запроса. ? — в DSQL и :paramname — в PSQL.</td>
</tr>
<tr>
<td>integer-expression</td>
<td>Выражение, возвращающее целочисленное значение.</td>
</tr>
</tbody>
</table>

FIRST и SKIP используются только в Firebird, они не включены в стандарт SQL. Рекомендуется использовать FETCH, OFFSET везде, где это возможно.

Выражение FIRST ограничивает результирующий набор данным указанным числом записей m.

Выражение SKIP пропускает указанное число записей n перед выдачей результирующего набора данных.

FIRST и SKIP являются необязательными выражениями.

Когда эти выражения используются совместно, например “FIRST m SKIP n”, то в результате n записей будет пропущено и, из оставшихся, m записей будет возвращено в результирующем наборе данных.

Особенности использования

- Разрешается использовать SKIP 0 – в этом случае 0 записей будет пропущено;
- В случае использования FIRST 0 будет возвращён пустой набор записей;
- Отрицательные значения FIRST и SKIP вызовут ошибку;
- Если указанное в SKIP значение превышает размер результирующего набора данных, то вернётся пустой набор данных;
- Если число записей в наборе данных (или остаток после применения SKIP) меньше, чем заданное в FIRST значение, то соответственно меньшее количество записей будет возвращено;
- Любой аргумент FIRST или SKIP, который не является целым числом или параметром SQL должен был заключён в круглые скобки. Это, означает, что в случае использования вложенной команды SELECT в качестве параметра для FIRST или SKIP, он должен быть вложен в две пары скобок.

Примеры

Следующий запрос вернёт первые 10 имён из таблицы PEOPLE (имена также будут отсортированы, см. ниже раздел ORDER BY):

```
SELECT FIRST 10 id, name
FROM People
ORDER BY name ASC
```
Следующий запрос вернёт все записи из таблицы PEOPLE, за исключением первых 10 имен:

```sql
SELECT SKIP 10 id, name
FROM People
ORDER BY name ASC
```

А этот запрос вернёт последние 10 записей (обратите внимание на двойные скобки):

```sql
SELECT SKIP ((SELECT COUNT(*) - 10 FROM People))
    id, name
FROM People
ORDER BY name ASC
```

Этот запрос вернёт строки 81-100 из таблицы PEOPLE:

```sql
SELECT FIRST 20 SKIP 80 id, name
FROM People
ORDER BY name ASC
```

См. также:
“FETCH`, OFFSET”, `ROWS.

### 6.1.2. Список полей SELECT

Список полей содержит одно или более выражений, разделённых запятыми. Результатом каждого выражения является значение соответствующего поля в наборе данных команды SELECT. Исключением является выражение * (“звездочка”), которое возвращает все поля отношения.

**Синтаксис**

```sql
SELECT [...] [DISTINCT | ALL] <select_list>
    [...] FROM ...

<select_list> ::= * | <output_column> [, <output_column> ...]

<output_column> ::= <qualifier>.*
    | <value_expression> [COLLATE collation] [[AS] alias]

<value-expression> ::= [<qualifier>.]col_name
```
Таблица 76. Параметры списка полей оператора SELECT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>qualifier</td>
<td>Имя таблицы (представления) или псевдоним таблицы (представления, хранимой процедуры, производной таблицы).</td>
</tr>
<tr>
<td>collation</td>
<td>Существующее имя сортировки (только для столбцов символьных типов).</td>
</tr>
<tr>
<td>alias</td>
<td>Псевдоним поля.</td>
</tr>
<tr>
<td>col_name</td>
<td>Столбец таблицы или представления.</td>
</tr>
<tr>
<td>selectable-SP-outparm</td>
<td>Выходной параметр селективной хранимой процедуры.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context-variable</td>
<td>Контекстная переменная.</td>
</tr>
<tr>
<td>function-call</td>
<td>Вызов скалярной, агрегатной или оконной функции.</td>
</tr>
<tr>
<td>single-value-subselect</td>
<td>Подзапрос, возвращающий единственное скалярное значение.</td>
</tr>
<tr>
<td>CASE-construct</td>
<td>Конструкция CASE.</td>
</tr>
<tr>
<td>other-single-value-expr</td>
<td>Любое другое выражение, возвращающее единственное значение типа данных Firebird или NULL.</td>
</tr>
</tbody>
</table>

Хорошим тоном является уточнять имя поля (или "*"") именем таблицы/представления/хранимой процедуры (или их псевдонимом), к которой это поле принадлежит. Например, `relationname.columnname`, `relationname.*`, `alias.columnname`, `alias.*`. Уточнение имени становится обязательным в случае, если поле с одним и тем же именем находится в более чем одном отношении, участвующей в объединении. Уточнение для "*" всегда обязательна, если это не единственный элемент в списке столбцов.

Обратите внимание

Алиасы (псевдонимы) заменяют оригинальное имя таблицы/представления/ хранимой процедуры: как только определён алиас для соответствующего отношения, использовать оригинальное имя нельзя.

В начале списка полей могут быть добавлены ключевые слова DISTINCT или ALL:

- DISTINCT удаляет дубликаты строк: то есть, если две или более записей содержат одинаковые значения во всех соответствующих полях, только одна из этих строк будет включена в результатирующий набор данных.
- ALL включает все строки в результатирующий набор данных. ALL включено по
умолчанию и поэтому редко используется: явное указание поддерживается для совместимости со стандартом SQL.

Выражение COLLATE не изменяет содержимое поля, однако, если указать COLLATE для определённого поля, то это может изменить чувствительность к регистру символов или к акцентам (accent sensitivity), что, в свою очередь, может повлиять на:

- Порядок сортировки, в случае если это поле указано в выражении ORDER BY;
- Группировку, в случае если это поле указано в выражении GROUP BY;
- Количество возвращаемых строк, если используется DISTINCT.

Примеры операторов SELECT с различными типами полей

Простой SELECT использующий только имена полей:

```
SELECT cust_id, cust_name, phone
FROM customers
WHERE city = 'London'
```

Запрос с конкатенацией и вызовом функции в списке полей:

```
SELECT 'Mr./Mrs. ' || lastname,
       street,
       zip,
       upper(city)
FROM contacts
WHERE date_last_purchase(id) = current_date
```

Запрос с двумя подзапросами:

```
SELECT p.fullname,
       (SELECT name FROM classes c
            WHERE c.id = p.class) AS class,
       (SELECT name FROM mentors m
            WHERE m.id = p.mentor) AS mentor
FROM pupils p
```

Следующий запрос делает то же самое, что и предыдущий, только с использованием соединения таблиц (JOIN) вместо подзапросов:

```
SELECT p.fullname,
       c.name AS class,
       m.name AS mentor
```
FROM pupils p
JOIN classes c ON c.id = p.class
JOIN mentors m ON m.id = p.mentor

Этот запрос использует конструкцию CASE для определения корректного обращения, например, при рассылке сообщений конкретному человеку:

SELECT
CASE upper(sex)
  WHEN 'F' THEN 'Mrs.'
  WHEN 'M' THEN 'Mr.'
  ELSE ''
END AS title,
lastname,
address
FROM employees

Запрос с использованием оконной функции. Выводит сотрудников отранжированных по заработной плате:

SELECT
  id,
salary,
name,
  DENSE_RANK() OVER (ORDER BY salary) AS EMP_RANK
FROM employees
ORDER BY salary;

Запрос к хранимой процедуре:

SELECT *
FROM interesting_transactions(2010, 3, 'S')
ORDER BY amount

Выборка полей производной таблицы. Производная таблица – это заключённый в скобки оператор SELECT, результат которого используется в запросе уровнем выше, как будто является обычной таблицей или представлением.
Запрос к контекстной переменной CURRENT_TIME:

```
SELECT current_time FROM rdb$database
```

Для тех, кто не знаком с RDB$DATABASE: это системная таблица, которая всегда существует во всех базах данных Firebird и всегда содержит только одну строку. И, хотя эта таблица не была создана специально для этой цели, стало распространённой практикой среди разработчиков Firebird выполнять запросы к этой таблице в случае, если нужно выполнить запрос, не привязанный ни к какой таблице, в котором результат получается из выражений, указанных в списке полей оператора SELECT. Например:

```
SELECT power(12, 2) AS twelve_squared, 
      power(12, 3) AS twelve_cubed 
FROM rdb$database
```

И, наконец, пример запроса к самой таблице RDB$DATABASE, с помощью которого можно получить кодировку по умолчанию данной БД:

```
SELECT rdb$character_set_name FROM rdb$database
```

См. также:
Агрегатные функции, Оконные (аналитические) функции, Контекстные переменные, CASE, Подзапросы.

**6.1.3. FROM**

Выражение FROM определяет источники, из которых будут отобраны данные. В простейшей форме, это может быть единственная таблица или представление. Однако источниками также могут быть хранимая процедура, производная таблица или общее табличное выражение (CTE). Различные виды источников могут комбинироваться с использованием разнообразных видов соединений (JOIN).

Этот раздел посвящён запрос из единственного источника. Соединения рассматриваются в следующем разделе.

Синтаксис:

```
SELECT ...
FROM <source>
```
[<joins>]
[...]
подзапросов, которые соотнесены с главным запросом (обычно подзапросы являются коррелированными).

Примеры

```sql
SELECT id, name, sex, age
FROM actors
WHERE state = 'Ohio'

SELECT *
FROM birds
WHERE type = 'flightless'
ORDER BY family, genus, species

SELECT
    firstname,
    middlename,
    lastname,
    date_of_birth,
    (SELECT name FROM schools s WHERE p.school = s.id) schoolname
FROM pupils p
WHERE year_started = 2012
ORDER BY schoolname, date_of_birth
```

Если вы дадите таблице или представлению псевдоним (алиас), то вы должны везде использовать этот псевдоним, а не имя таблицы, при обращении к именам столбцов.

Корректное использование:

```sql
SELECT PEAR
FROM FRUIT

SELECT FRUIT.PEAR
FROM FRUIT

SELECT PEAR
FROM FRUIT F

SELECT F.PEAR
FROM FRUIT F
```

Некорректное использование:

```sql
SELECT FRUIT.PEAR
FROM FRUIT F
```
Выборка из селективной хранимой процедуры

Селективная хранимая процедура (т.е. с возможностью выборки) должна удовлетворять следующим условиям:

• Содержать, по крайней мере, один выходной параметр;
• Использовать ключевое слово SUSPEND таким образом, чтобы вызывающий запрос мог выбирать выходные строки одну за другой, так же как выбираются строки таблицы или представления.

Выходные параметры селективной хранимой процедуры с точки зрения команды SELECT соответствуют полям обычной таблицы.

Выборка из хранимой процедуры без входных параметров осуществляется точно так же, как обычная выборка из таблицы:

```sql
SELECT * 
FROM suspicious_transactions 
WHERE assignee = 'Dmitrii'
```

Если хранимая процедура требует входные параметры, то они должны быть указаны в скобках после имени процедуры:

```sql
SELECT name, az, alt 
FROM visible_stars('Brugge', current_date, '22:30') 
WHERE alt >= 20 
ORDER BY az, alt
```

Значения для опциональных параметров (то есть параметров, для которых определены значения по умолчанию) могут быть указаны или опущены.

Однако если параметры задаются частично, то пропущенные параметры должны быть в конце перечисления внутри скобок.

Если предположить, что процедура visible_stars из предыдущего примера имеет два опциональных параметра spectral_class (varchar(12)) и min_magn (numeric(3,1)), то следующие команды будут корректными:

```sql
SELECT name, az, alt 
FROM visible_stars('Brugge', current_date, '22:30')
```

```sql
SELECT name, az, alt 
FROM visible_stars('Brugge', current_date, '22:30', 4.0)
```

А вот этот запрос не будет корректным:
Алиас для селективной хранимой процедуры указывается после списка параметров:

```
SELECT number,
    (SELECT name FROM contestants c
        WHERE c.number = gw.number)
FROM get_winners('#34517', 'AMS') gw
```

Если вы указываете поле (выходной параметр) с полным именем процедуры, не включайте в это имя список параметров процедуры:

```
SELECT number,
    (SELECT name FROM contestants c
        WHERE c.number = get_winners.number)
FROM get_winners('#34517', 'AMS')
```

*См. также:*

Хранимые процедуры, CREATE PROCEDURE.

Выборка из производной таблицы (derived table)

Производная таблица — это корректная команда SELECT, заключённая в круглые скобки, опционально обозначенная псевдонимом таблицы и псевдонимами полей.

**Синтаксис**

```
<derived table> ::=  
    (<select-query>)  
    [[AS] derived-table-alias]  
    [((<derived-column-aliases>))]  

<derived-column-aliases> ::= column-alias [, column-alias ...]  

<lateral-derived-table> ::= LATERAL <derived-table>
```

Возвращаемый набор данных такого оператора представляет собой виртуальную таблицу, к которой можно составлять запросы, так как будто это обычная таблица.

Производная таблица в запросе ниже выводит список имён таблиц в базе данных и количество столбцов в них. Запрос к производной таблице выводит количество полей, и количество таблиц с таким количеством полей.

```
SELECT name, az, alt
FROM visible_stars('Brugge', current_date, 4.0)
```
Тривиальный пример, демонстрирующий использование псевдонима производной таблицы и списка псевдонимов столбцов (оба опциональные):

```sql
SELECT
    DBINFO.DESCRIPT, DBINFO.DEF_CHARSET
FROM (SELECT *
        FROM RDB$DATABASE) DBINFO (DESCR, REL_ID, SEC_CLASS, DEF_CHARSET)
```

Примечания:

- Производные таблицы могут быть вложенными;
- Производные таблицы могут быть объединениями и использоваться в объединениях. Они могут содержать агрегатные функции, подзапросы и соединения, и сами по себе могут быть использованы в агрегатных функциях, подзапросах и соединениях. Они также могут быть хранимыми процедурами или запросами из них. Они могут иметь предложения WHERE, ORDER BY и GROUP BY, указания FIRST, SKIP или ROWS и т.д.;
- Каждый столбец в производной таблице должен иметь имя. Если этого нет по своей природе (например, потому что это — константа), то надо в обычном порядке присвоить псевдоним или добавить список псевдонимов столбцов в спецификации производной таблицы;
- Список псевдонимов столбцов опциональный, но если он присутствует, то должен быть полным (т.е. он должен содержать псевдоним для каждого столбца производной таблицы);
- Оптимизатор может обрабатывать производные таблицы очень эффективно. Однако если производная таблица включена во внутреннее соединение и содержит подзапрос, то какой порядок соединения не может быть использован оптимизатором;
- Ключевое слово LATERAL позволяет производной таблице ссылаться на поля из ранее перечисленных таблиц в текущем <table reference list>. Подробнее смотрите в разделе Соединение с LATERAL производными таблицами.

Приведём пример того, как использование производных таблиц может упростить решение некоторой задачи.
Предположим, что у нас есть таблица COEFFS, которая содержит коэффициенты для ряда квадратных уравнений, которые мы собираемся решить. Она может быть определена примерно так:

```sql
CREATE TABLE coeffs (  
a DOUBLE PRECISION NOT NULL,  
b DOUBLE PRECISION NOT NULL,  
c DOUBLE PRECISION NOT NULL,  
CONSTRAINT chk_a_not_zero CHECK (a <> 0)
)
```

В зависимости от значений коэффициентов a, b и c, каждое уравнение может иметь ноль, одно или два решения. Мы можем найти эти решения с помощью одноуровневого запроса к таблице COEFFS, однако код такого запроса будет громоздким, а некоторые значения (такие, как дискриминанты) будут вычисляться несколько раз в каждой строке.

Если использовать производную таблицу, то запрос можно сделать гораздо более элегантным:

```sql
SELECT  
IIF (D >= 0, (-b - sqrt(D)) / denom, NULL) AS sol_1,  
IIF (D > 0, (-b + sqrt(D)) / denom, NULL) AS sol_2  
FROM  
(SELECT b, b*b - 4*a*c, 2*a FROM coeffs) (b, D, denom)
```

Если мы захотим показывать коэффициенты рядом с решениями уравнений, то мы можем модифицировать запрос следующим образом:

```sql
SELECT  
a, b, c,  
IIF (D >= 0, (-b - sqrt(D)) / denom, NULL) sol_1,  
IIF (D > 0, (-b + sqrt(D)) / denom, NULL) sol_2  
FROM  
(SELECT a, b, c, b*b - 4*a*c AS D, 2*a AS denom  
FROM coeffs)
```

Обратите внимание, что в первом запросе мы назначили алиасы для всех полей производной таблицы в виде списка после таблицы, а во втором, по мере необходимости, добавляем алиасы внутри запроса производной таблицы. Оба этих метода корректны, так как при правильном применении гарантируют, что каждое поле производной таблицы имеет уникальное имя.

На самом деле все столбцы, вычисляемые в производной таблице, будут перечислены столько раз, сколько раз они указываются в основном запросе. Это важно может привести к неожиданным результатам при использовании недетерминированных функций. Следующий пример
показывает сказанное:

```sql
SELECT
    UUID_TO_CHAR(X) AS C1,
    UUID_TO_CHAR(X) AS C2,
    UUID_TO_CHAR(X) AS C3
FROM (SELECT GEN_UUID() AS X
      FROM RDB$DATABASE) T;
```

результатом этого запроса будет

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
<tr>
<td>C2</td>
<td>C1214CD3-423C-406D-858D-95BF432E3E3</td>
</tr>
<tr>
<td>C3</td>
<td>EB176C10-F754-4689-8B84-6486663B1154</td>
</tr>
</tbody>
</table>

Для материализации результата функции GEN_UUID вы можете воспользоваться следующим способом:

```sql
SELECT
    UUID_TO_CHAR(X) AS C1,
    UUID_TO_CHAR(X) AS C2,
    UUID_TO_CHAR(X) AS C3
FROM (SELECT GEN_UUID() AS X
      FROM RDB$DATABASE
      UNION ALL
      SELECT NULL FROM RDB$DATABASE WHERE 1=0) T;
```

результатом этого запроса будет

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
<tr>
<td>C2</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
<tr>
<td>C3</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
</tbody>
</table>

или завернуть функцию GEN_UUID в подзапрос

```sql
SELECT
    UUID_TO_CHAR(X) AS C1,
    UUID_TO_CHAR(X) AS C2,
    UUID_TO_CHAR(X) AS C3
FROM (SELECT (SELECT GEN_UUID() FROM RDB$DATABASE) AS X
      FROM RDB$DATABASE) T;
```

Эта особенность текущей реализации и она может быть изменена в следующих версиях сервера.
Латеральные производные таблицы

Производная таблица, определенная с помощью ключевого слова LATERAL, называется латеральной производной таблицей. Если производная таблица определена как латеральная, то разрешается ссылаться на другие таблицы в том же предложении FROM, но только на те, которые были объявлены до этого в предложении FROM.

Example 226. Запросы с латеральными производными таблицами

```sql
select dt.population, dt.city_name, c.country_name
from (select distinct country_name from cities) AS c,
LATERAL (select first 1 city_name, population
from cities
where cities.country_name = c.country_name
order by population desc) AS dt;
```

```sql
select salespeople.name,
    max_sale.amount,
    customer_of_max_sale.customer_name
from salespeople,
LATERAL (select max(amount) as amount from all_sales
where all_sales.salesperson_id = salespeople.id
) as max_sale,
LATERAL (select customer_name from all_sales
where all_sales.salesperson_id = salespeople.id
    and all_sales.amount = max_sale.amount
) as customer_of_max_sale;
```

Выборка из общих табличных выражений (CTE)

Общие табличные выражения являются более сложной и более мощной вариацией производных таблиц. CTE состоит из преамбулы, начинающейся с ключевого слова WITH, которая определяет одно или более общих табличных выражений (каждое из которых может иметь список алиасов полей). Основной запрос, который следует за преамбулой, может обращаться к CTE так, как будто обычные таблицы. CTE доступны любой части запроса ниже точки своего объявления.

Подробно CTE описываются в разделе Общие табличные выражения CTE (WITH ... AS ... SELECT), а здесь приведены лишь некоторые примеры использования.

Следующий запрос представляет наш пример с производной таблицей в варианте для общих табличных выражений:

```sql
WITH vars (b, D, denom) AS (
    SELECT b, b*b - 4*a*c, 2*a
    FROM coeffs
)
```
Это не слишком большое улучшение по сравнению с вариантом с производными таблицами (за исключением того, что вычисления проводятся до основного запроса). Мы можем ещё улучшить запрос, исключив двойное вычисление \( \sqrt{D} \) для каждой строки:

```sql
WITH vars (b, D, denom) AS ( 
    SELECT b, b*b - 4*a*c, 2*a 
    FROM coeffs 
), 
vars2 (b, D, denom, sqrtD) AS ( 
    SELECT b, D, denom, 
        IIF (D >= 0, sqrt(D), NULL) 
    FROM vars 
) 
SELECT 
    IIF (D >= 0, (-b - sqrtD) / denom, NULL) AS sol_1, 
    IIF (D > 0, (-b + sqrtD) / denom, NULL) AS sol_2 
FROM vars2
```

Текст запроса выглядит более сложным, но он стал более эффективным (предполагая, что исполнение функции \( \text{SQRT} \) занимает больше времени, чем передача значений переменных \( b, d \) и \( denom \) через дополнительное CTE).

На самом деле все столбцы, вычисляемые в CTE, будут перевычислены столько раз, сколько раз они указываются в основном запросе. Это важно может привести к неожиданным результатам при использовании недетерминированных функций. Следующий пример показывает сказанное:

```sql
WITH T(X) AS (SELECT GEN_UUID() 
        FROM RDB$DATABASE) 
SELECT 
    UUID_TO_CHAR(X) as c1, 
    UUID_TO_CHAR(X) as c2, 
    UUID_TO_CHAR(X) as c3 
FROM T
```

результатом этого запроса будет

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
<td>C1214CD3-423C-406D-B5BD-95BF432ED3E3</td>
</tr>
</tbody>
</table>
Для материализации результата функции GEN_UUID вы можете воспользоваться следующим способом:

```sql
WITH T(X) 
AS (SELECT GEN_UUID() 
    FROM RDB$DATABASE 
    UNION ALL 
    SELECT NULL FROM RDB$DATABASE WHERE 1=0) 
SELECT UUID_TO_CHAR(X) as c1, 
    UUID_TO_CHAR(X) as c2, 
    UUID_TO_CHAR(X) as c3 
FROM T;
```

результатом этого запроса будет

<table>
<thead>
<tr>
<th></th>
<th>80AAECED-65CD-4C2F-90AB-5D548C3C7279</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
<tr>
<td>C2</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
<tr>
<td>C3</td>
<td>80AAECED-65CD-4C2F-90AB-5D548C3C7279</td>
</tr>
</tbody>
</table>

или завернуть функцию GEN_UUID в подзапрос

```sql
WITH T(X) 
AS (SELECT (SELECT GEN_UUID() FROM RDB$DATABASE) 
    FROM RDB$DATABASE) 
SELECT UUID_TO_CHAR(X) as c1, 
    UUID_TO_CHAR(X) as c2, 
    UUID_TO_CHAR(X) as c3 
FROM T;
```

Эта особенность текущей реализации и она может быть изменена в следующих версиях сервера.

Конечно, мы могли бы добиться такого результата и с помощью производных таблиц, но это потребовало бы вложить запросы один в другой.

См. также:
Общие табличные выражения CTE (WITH ... AS ... SELECT).

6.1.4. Соединения JOIN

Соединения объединяют данные из двух источников в один набор данных. Соединение данных осуществляется для каждой строки и обычно включает в себя проверку условия
соединения (join condition) для того, чтобы определить, какие строки должны быть объединены и оказаться в результирующем наборе данных.

Результат соединения также может быть соединён с другим набором данных с помощью следующего соединения.

Существует несколько типов (INNER, OUTER) и классов (квалифицированные, натуральные, и др.) соединений, каждый из которых имеет свой синтаксис и правила.

Синтаксис

```sql
SELECT
    ...
FROM <source>
[<joins>]
[...]

<source> ::= {
    table
    | view
    | selectable-stored-procedure [(<args>)]
    | <derived-table>
    | <lateral-derived-table>
    | <common-table-expression>
} [[AS] alias]

<joins> ::= <join> [<join> ...]

<join> ::= [
    [<join-type>] JOIN <source> <join-condition>
    | NATURAL [<join-type>] JOIN <source>
    | {CROSS JOIN | ,} <source>
]

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<join-condition> ::= ON <condition> | USING (<column-list>)
```

Таблица 78. Параметры предложения JOIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Таблица.</td>
</tr>
<tr>
<td>view</td>
<td>Представление.</td>
</tr>
<tr>
<td>selectable-stored-procedure</td>
<td>Селективная хранимая процедура.</td>
</tr>
<tr>
<td>args</td>
<td>Аргументы селективной хранимой процедуры.</td>
</tr>
<tr>
<td>derived-table</td>
<td>Производная таблица.</td>
</tr>
<tr>
<td>common-table-expression</td>
<td>Общее табличное выражение (CTE).</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>alias</td>
<td>Псевдоним (алиас) для одного из источников данных (таблицы, представления, процедуры, CTE, производной таблицы).</td>
</tr>
<tr>
<td>condition</td>
<td>Условие соединения.</td>
</tr>
<tr>
<td>column-list</td>
<td>Список столбцов по которым происходит эквисоединение.</td>
</tr>
</tbody>
</table>

**Внутренние (INNER) и внешние (OUTER) соединения**

Соединение всегда соединяет строки из двух наборов данных (которые обычно называются “левый” и “правый”). По умолчанию, только строки, которые удовлетворяет условию соединения (те, которым соответствует хотя бы одна строка из другого набора строк согласно применяемому условию) попадают в результирующий набор данных. Такой тип соединения (который является типом по умолчанию) называется внутренним (INNER JOIN).

Предположим, у нас есть 2 таблицы:

**Таблица A**

<table>
<thead>
<tr>
<th>ID</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>Just some text</td>
</tr>
<tr>
<td>35</td>
<td>Silence</td>
</tr>
</tbody>
</table>

**Таблица B**

<table>
<thead>
<tr>
<th>CODE</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>-23</td>
<td>56.7735</td>
</tr>
<tr>
<td>87</td>
<td>416.0</td>
</tr>
</tbody>
</table>

Если мы соединим эти таблицы с помощью вот такого запроса:

```sql
SELECT * FROM A JOIN B ON A.id = B.code
```

do результат будет:

<table>
<thead>
<tr>
<th>ID</th>
<th>S</th>
<th>CODE</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>Just some text</td>
<td>87</td>
<td>416.0</td>
</tr>
</tbody>
</table>

То есть, первая строка таблицы A была соединена со второй строкой таблицы B, потому что вместе они удовлетворяют условию соединения “A.id = B.code”. Другие строки не имеют соответствия и поэтому не включаются в соединение. Помните, что умолчанию соединение всегда внутреннее (INNER).

Мы можем сделать это явным, указав тип соединения:
но обычно слово INNER опускается.

Разумеется, возможны случаи, когда строке в левом наборе данных соответствует несколько строк в правом наборе данных (или наоборот).

В таких случаях все комбинации включаются в результирующих набор данных, и мы можем получить результат вроде этого:

<table>
<thead>
<tr>
<th>ID</th>
<th>S</th>
<th>CODE</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>Just some text</td>
<td>87</td>
<td>416.0</td>
</tr>
<tr>
<td>87</td>
<td>Just some text</td>
<td>87</td>
<td>-1.0</td>
</tr>
<tr>
<td>-23</td>
<td>Don't know</td>
<td>-23</td>
<td>56.7735</td>
</tr>
<tr>
<td>-23</td>
<td>Still don't</td>
<td>-23</td>
<td>56.7735</td>
</tr>
<tr>
<td>-23</td>
<td>I give up</td>
<td>-23</td>
<td>56.7735</td>
</tr>
</tbody>
</table>

Иногда необходимо включить в результат все записи из левого или правого набора данных, вне зависимости от того, есть ли для них соответствующая запись в парном наборе данных. В этом случае необходимо использовать внешние соединения.

Внешнее левое соединение (LEFT OUTER) включает все записи из левого набора данных, и те записи из правого набора, которые удовлетворяют условию соединения.

Внешнее правое соединение (RIGHT OUTER) включает все записи из правого набора данных и те записи из левого набора данных, которые удовлетворяют условию соединения.

Полное внешнее соединение (FULL OUTER) включает все записи из обоих наборов данных.

Во всех внешних соединениях, “дыры” (то есть поля набора данных, в которых нет соответствующей записи) заполняются NULL.

Для обозначения внешнего соединения используются ключевые слова LEFT, RIGHT или FULL с необязательным ключевым словом OUTER.

Рассмотрим различные внешние соединения на примере запросов с указанными выше таблицами A и B:

```
SELECT * 
FROM A 
LEFT OUTER JOIN B ON A.id = B.code
```

то же самое
**Точные соединения**

Явный синтаксис соединения требует указания условия соединения записей. Это условие
указывается явно в предложении ON или неявно при помощи предложения USING.

Синтаксис

\[
\text{<qualified-join>} ::= \text{[<join-type>] JOIN <source> <join-condition>}
\]

\[
\text{<join-type>} ::= \text{INNER | \{LEFT | RIGHT | FULL\} [OUTER]}
\]

\[
\text{<join-condition>} ::= \text{ON <condition> | USING (<column-list>)}
\]

Соединения с явными условиями

В синтаксисе явного соединения есть предложение ON, с условием соединения, в котором может быть указано любое логическое выражение, но, как правило, оно содержит условие сравнения между двумя участвующими источниками.

Довольно часто, это условие— проверка на равенство (или ряд проверок на равенство объединённых оператором AND) использующая оператор "=". Такие соединения называются эквисоединениями. (Примеры в главе Внутренние (INNER) и внешние (OUTER) соединения были эквисоединениями).

Примеры соединений с явными условиями:

```sql
/*
 * Выборка всех заказчиков из города Детройт, которые
 * сделали покупку.
 */
SELECT *
FROM customers c
JOIN sales s ON s.cust_id = c.id
WHERE c.city = 'Detroit'

/*
 * Тоже самое, но включает в выборку заказчиков, которые
 * не совершали покупки.
 */
SELECT *
FROM customers c
LEFT JOIN sales s ON s.cust_id = c.id
WHERE c.city = 'Detroit'

/*
 * Для каждого мужчины выбрать женщин, которые выше него.
 * Мужчины, для которых такой женщины не существуют,
 * не будут выключены в выборку.
 */
SELECT
    m.fullname AS man,
    f.fullname AS woman
FROM males m
```
JOIN females f ON f.height > m.height

/
* Выборка всех учеников, их класса и наставника.
* Ученики без наставника будут включены в выборку.
* Ученики без класса не будут включены в выборку.
*/

SELECT
  p.firstname,
  p.middlename,
  p.lastname,
  c.name,
  m.name
FROM pupils p
JOIN classes c ON c.id = p.class
LEFT JOIN mentors m ON m.id = p.mentor

Соединения именованными столбцами

Эквисоединения часто сравнивают столбцы, которые имеют одно и то же имя в обеих таблицах. Для таких соединений мы можем использовать второй тип явных соединений, называемый соединением именованными столбцами (Named Columns Joins). Соединение именованными столбцами осуществляется с помощью предложения USING, в котором перечисляются только имена столбцов.

Таким образом, следующий пример:

```
SELECT *
FROM flotsam f
JOIN jetsam j
  ON f.sea = j.sea AND f.ship = j.ship
```

can be rewritten as:

```
SELECT *
FROM flotsam
JOIN jetsam USING (sea, ship)
```

что значительно короче. Результирующий набор несколько отличается, по крайней мере, при использовании "SELECT *":

- Результат соединения с явным условием соединения в предложении ON будет содержать каждый из столбцов SEA и `SHIP дважды: один раз для таблицы FLOTSAM и один раз для таблицы JETSAM. Очевидно, что они будут иметь одинаковые значения;
- Результат соединения именованными столбцами, с помощью предложения USING, будет
Если вы хотите получить в результате соединения именованными столбцами все столбцы, перепишите запрос следующим образом:

```sql
SELECT f.*, j.*
FROM flotsam f
JOIN jetsam j USING (sea, ship)
```

Для внешних (OUTER) соединений именованными столбцами, существуют дополнительные нюансы, при использовании “SELECT *” или неполного имени столбца. Если столбец строки из одного источника не имеет совпадений со столбцом строки из другого источника, но все равно должен быть включен результат из-за инструкций LEFT, RIGHT или FULL, то объединяемый столбец получит не NULL значение. Это достаточно справедливо, но теперь вы не можете сказать из какого набора левого, правого или обоих пришло это значение. Это особенно обманчиво, когда значения пришли из правой части набора данных, потому что “*” всегда отображает для комбинированных столбцов значения из левой части набора данных, даже если используется RIGHT соединение.

Является ли это проблемой, зависит от ситуации. Если это так, используйте “f.*, j.*” подход, продемонстрированный выше, где f и j имена или алиасы двух источников. Или лучше вообще избегать “*” в серьезных запросах и перечислять все имена столбцов для соединяемых множеств. Такой подход имеет дополнительное преимущество, заставляя вас думать, о том какие данные вы хотите получить и откуда.

Вся ответственность за совместимость типов столбцов между соединяемыми источниками, имена которых перечислены в предложении USING, лежит на вас. Если типы совместимы, но не равны, то Firebird преобразует их в тип с более широким диапазоном значений перед сравнением. Кроме того, это будет типом данных объединённого столбца, который появится в результирующем наборе, если используются “SELECT *” или неполное имя столбца. Полные имена столбцов всегда будут сохранять свой первоначальный тип данных.

Если при соединении именованными столбцами вы используете столбцы соединения в условии отбора WHERE, то всегда используйте уточнённые имена столбцов. В противном случае индекс по этому столбцу не будет задействован.

```sql
SELECT 1 FROM t1 a JOIN t2 b USING(x) WHERE x = 0;
PLAN JOIN (A NATURAL, B INDEX (RDB$2))
```

Однако

```sql
SELECT 1 FROM t1 a JOIN t2 b USING(x) WHERE a.x = 0; -- или 'b.x'
PLAN JOIN (A INDEX (RDB$1), B INDEX (RDB$2))

SELECT 1 FROM t1 a JOIN t2 b USING(x) WHERE b.x = 0;
```
Естественные соединения (natural join)

Взяв за основу соединения именованными столбцами, следующим шагом будет естественное соединение, которое выполняет эквисоединение по всем одноименным столбцам правой и левой таблицы. Типы данных этих столбцов должны быть совместимыми.

Естественные соединения доступны только в диалекте 3.

Синтаксис

```
<natural-join> ::= NATURAL [<join-type>] JOIN <source>

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Даны две таблицы:

```sql
CREATE TABLE TA (    
    a BIGINT,    
    s VARCHAR(12),    
    ins_date DATE
);

CREATE TABLE TB (    
    a BIGINT,    
    descr VARCHAR(12),    
    x FLOAT,    
    ins_date DATE
);
```

Естественное соединение таблиц TA и TB будет происходить по столбцам a и ins_date и два следующих оператора дадут один и тот же результат:

```sql
SELECT *  
FROM TA  
NATURAL JOIN TB;

SELECT *  
FROM TA  
JOIN TB USING (a, ins_date);
```
Как и все соединения, естественные соединения являются внутренними соединениями по умолчанию, но вы можете превратить их во внешние соединения, указав LEFT, RIGHT или FULL перед ключевым словом JOIN.

Внимание
Если в двух исходных таблицах не будут найдены одноименные столбцы, то будет выполнен CROSS JOIN.

Неявные соединения
В стандарте SQL-89 таблицы, участвующие в соединении, задаются списком с разделяющими запятыми в предложении FROM. Условия соединения задаются в предложении WHERE среди других условий поиска. Такие соединения называются неявными.

Синтаксис неявного соединения может осуществлять только внутренние соединения.

Пример неявного соединения:

```sql
/*
 * Выборка всех заказчиков из города Детройт, которые
 * сделали покупку.
 */
SELECT *
FROM customers c, sales s
WHERE s.cust_id = c.id AND c.city = 'Detroit'
```

В настоящее время синтаксис неявных соединений не рекомендуется к использованию.

Смешивание явного и неявного соединения
Смешивание явных и неявных соединений не рекомендуется, но позволяет. Некоторые виды смешивания запрещены в Firebird.

Например, такой запрос вызовет ошибку "Column does not belong to referenced table"

```sql
SELECT *
FROM TA, TB
JOIN TC ON TA.COL1 = TC.COL1
WHERE TA.COL2 = TB.COL2
```

Это происходит потому, что явный JOIN не может видеть таблицу TA. Однако следующий запрос будет выполнен без ошибок, поскольку изоляция не нарушена.

```sql
SELECT *
FROM
```
Перекрёстное соединение (CROSS JOIN)

Перекрёстное соединение или декартово произведение. Каждая строка левой таблицы соединяется с каждой строкой правой таблицы.

Синтаксис

\[
\text{<cross-join>} ::= \{\text{CROSS JOIN} \mid ,\} \text{ <source>}
\]

Обратите внимание, что синтаксис с использованием запятой является устаревшим. Он поддерживается только для поддержания работоспособности унаследованного программного кода и может быть удалён в будущих версиях.

Перекрёстное соединение двух наборов эквивалентно их соединению по условию тавтологии (условие, которое всегда верно).

Следующие два запроса дадут один и тот же результат:

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad \text{TA} \\
\text{CROSS} & \quad \text{JOIN} \quad \text{TB}; \\
\text{SELECT} & \quad * \\
\text{FROM} & \quad \text{TA} \\
\text{JOIN} & \quad \text{TB} \quad \text{ON} \quad 1 = 1;
\end{align*}
\]

Перекрёстные соединения являются внутренними соединениями, потому что они отбирают строки, для которых есть соответствие — так уж случилось, что каждая строка соответствует! Внешнее перекрёстное соединение, если бы оно существовало, ничего не добавило бы к результату, потому что внешние соединения добавляют записи, по которым нет соответствия, а они не существуют в перекрёстном соединении.

Перекрёстные соединения редко полезны, кроме случаев, когда вы хотите получить список всех возможных комбинаций двух или более переменных. Предположим, вы продаёте продукт, который поставляется в различных размерах, различных цветов и из различных материалов. Если для каждой переменной значения перечислены в собственной таблице, то этот запрос будет возвращать все комбинации:

\[
\begin{align*}
\text{SELECT} & \quad \text{m.name,} \\
& \quad \text{s.size,} \\
& \quad \text{c.name} \\
\text{FROM} & \quad \text{materials m} \\
\text{CROSS} & \quad \text{JOIN} \quad \text{sizes s}
\end{align*}
\]
Неоднозначные имена полей в соединениях

Firebird отвергает неполные имена полей в запросе, если эти имена полей существуют в более чем одном наборе данных, участвующих в объединении. Это также верно для внутренних эквисоединений, в которых имена полей фигурируют в предложении ON:

```sql
SELECT a, b, c
FROM TA
JOIN TB ON TA.a = TB.a
```

Существует одно исключение из этого правила: соединения по именованным столбцам и естественные соединения, которые используют неполное имя поля в процессе подбора, могут использоваться законно. Это же относится и к одноименным объединяемым столбцам. Для соединений по именованным столбцам эти столбцы должны быть перечислены в предложении USING. Для естественных соединений это столбцы, имена которых присутствуют в обеих таблицах. Но снова замечу, что, особенно во внешних соединениях, плоское имя `colname` является не всегда тем же самым что `left.colname` или `right.colname`. Типы данных могут отличаться, и один из полных столбцов может иметь значение `NULL`, в то время как другой нет. В этом случае значение в объединённом, неполном столбце может замаскировать тот факт, что одно из исходных значений отсутствует.

Соединения с хранимыми процедурами

Если соединение происходит с хранимой процедурой, которая не коррелирована с другими потоками данных через входные параметры, то нет никаких особенностей.

В противном случае есть одна особенность: потоки, используемые во входных параметрах, должны быть описаны раньше соединения с хранимой процедурой:

```sql
SELECT *
FROM MY_TAB
JOIN MY_PROC(MY_TAB.F) ON 1 = 1
```

Запрос же написанный следующим образом вызовет ошибку:

```sql
SELECT *
FROM MY_PROC(MY_TAB.F)
JOIN MY_TAB ON 1 = 1
```

Соединения с LATERAL производными таблицами

Производная таблица, определенная с помощью ключевого слова LATERAL, называется латеральной производной таблицей. Если производная таблица определена как
латеральная, то разрешается ссылаться на другие таблицы в том же предложении FROM, но только на те, которые были объявлены раньше в предложении FROM. Без LATERAL каждый подзапрос выполняется независимо и поэтому не может обращаться к другим элементам FROM.

Элемент LATERAL может находиться на верхнем уровне списка FROM или в дереве JOIN. В последнем случае он может также ссылаться на любые элементы в левой части JOIN, справа от которого он находится.

Когда элемент FROM содержит ссылки LATERAL, то запрос выполняется следующим образом: сначала вычисляется значения всех столбцов о которых зависит производная таблица с ключевым словом LATERAL, затем вычисляется сама производная таблица с LATERAL для каждой полученной записи. Результирующие строки полученные из производной таблицы с LATERAL соединяются со строками из которых они получены.

В качестве соединений допускается следующие CROSS JOIN и LEFT OUTER JOIN. Внутреннее соединение также допустимо, но не рекомендуется, поскольку могут возникнуть проблемы при вычислении условия соединения потоков.

В качестве примера выведем результаты лошадей и их последние промеры. Если у лошади нет ни одного промера, то она не будет выведена:

```
SELECT
  HORSE.NAME,
  M.BYDATE,
  M.HEIGHT_HORSE,
  M.LENGTH_HORSE
FROM HORSE
CROSS JOIN LATERAL(
  SELECT *
  FROM MEASURE
  WHERE MEASURE.CODE_HORSE = HORSE.CODE_HORSE
  ORDER BY MEASURE.BYDATE DESC
  FETCH FIRST ROW ONLY) M
```

dругой вариант написание этого запроса

```
SELECT
  HORSE.NAME,
  M.BYDATE,
  M.HEIGHT_HORSE,
  M.LENGTH_HORSE
FROM HORSE,
  LATERAL(SELECT *
    FROM MEASURE
    WHERE MEASURE.CODE_HORSE = HORSE.CODE_HORSE
    ORDER BY MEASURE.BYDATE DESC
```
Если необходимо выводить лошадей, не зависимо есть ли у них хотя бы один промер, то необходимо заменить CROSS JOIN на LEFT JOIN:

```sql
SELECT
    HORSE.NAME,
    M.BYDATE,
    M.HEIGHT_HORSE,
    M.LENGTH_HORSE
FROM HORSE
LEFT JOIN LATERAL (
    SELECT *
    FROM MEASURE
    WHERE MEASURE.CODE_HORSE = HORSE.CODE_HORSE
    ORDER BY MEASURE.BYDATE DESC
    FETCH FIRST ROW ONLY) M ON TRUE
```

### 6.1.5. WHERE

Предложение WHERE предназначено для ограничения количества возвращаемых строк, теми которые нас интересуют. Условие после ключевого слова WHERE может быть простым, как проверка “AMOUNT = 3”, так и сложным, запутанным выражением, содержащим подзапросы, предикаты, вызовы функций, математические и логические операторы, контекстные переменные и многое другое.

Условие в предложении WHERE часто называют условием поиска, выражением поиска или просто поиск.

В DSQL и ESQL, выражение поиска могут содержать параметры. Это полезно, если запрос должен быть повторен несколько раз с разными значениями входных параметров. В строке SQL запроса, передаваемого на сервер, вопросительные знаки используются как заполнители для параметров. Их называют позиционными параметрами, потому что они не могут сказать ничего кроме как о позиции в строке. Библиотеки доступа часто поддерживают именованные параметры в виде :id, :amount, :a и т.д. Это более удобно для пользователя, библиотека заботится о трансляции именованных параметров в позиционные параметры, прежде чем передать запрос на сервер.

Условие поиска может также содержать локальные (PSQL) или хост (ESQL) имена переменных, предваряемых двоеточием.

**Синтаксис**

```sql
SELECT ...
FROM ...
[...]
WHERE <search-condition>
[...]
```
Только те строки, для которых условие поиска истинно будут включены в результатирующий набор. Будьте осторожны с возможными получаемыми значениями NULL: если вы отрицаете выражение, дающее NULL с помощью NOT, то результат такого выражения все равно будет NULL и строка не пройдёт. Это демонстрируется в одном из ниже приведённых примеров.

**Примеры**

```sql
SELECT genus, species
FROM mammals
WHERE family = 'Felidae'
ORDER BY genus;

SELECT *
FROM persons
WHERE birthyear IN (1880, 1881)
  OR birthyear BETWEEN 1891 AND 1898;

SELECT name, street, borough, phone
FROM schools s
WHERE EXISTS (SELECT * FROM pupils p WHERE p.school = s.id)
ORDER BY borough, street;

SELECT *
FROM employees
WHERE salary >= 10000 AND position <> 'Manager';

SELECT name
FROM wrestlers
WHERE region = 'Europe'
  AND weight > ALL (SELECT weight FROM shot_putters
               WHERE region = 'Africa');

SELECT id, name
FROM players
WHERE team_id = (SELECT id FROM teams
               WHERE name = 'Buffaloes');

SELECT SUM (population)
FROM towns
WHERE name LIKE '%dam'
  AND province CONTAINING 'land';

SELECT pass
FROM usertable
```
WHERE username = current_user;

Следующий пример показывает, что может быть, если условие поиска вычисляется как NULL.

Предположим у вас есть таблица, в которой находятся несколько детских имен и количество шариков, которыми они обладают.

<table>
<thead>
<tr>
<th>CHILD</th>
<th>MARBLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anita</td>
<td>23</td>
</tr>
<tr>
<td>Bob E.</td>
<td>12</td>
</tr>
<tr>
<td>Chris</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>Deirdre</td>
<td>1</td>
</tr>
<tr>
<td>Eve</td>
<td>17</td>
</tr>
<tr>
<td>Fritz</td>
<td>0</td>
</tr>
<tr>
<td>Gerry</td>
<td>21</td>
</tr>
<tr>
<td>Hadassah</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>Isaac</td>
<td>6</td>
</tr>
</tbody>
</table>

Первое, обратите внимание на разницу между NULL и 0. Известно, что Fritz не имеет шариков вовсе, однако неизвестно количество шариков у Chris и Hadassah.

Теперь, если ввести этот SQL оператор:

```
SELECT LIST(child) FROM marbletable WHERE marbles > 10
```

вы получите имена Anita, Bob E., Eve и Gerry. Все эти дети имеют более чем 10 шариков.

Если вы отрицаете выражение:

```
SELECT LIST(child) FROM marbletable WHERE NOT marbles > 10
```

запрос вернёт Deirdre, Fritz и Isaac. Chris и Hadassah не будут включены в выборку, так как не известно 10 у них шариков или меньше. Если вы измените последний запрос так:

```
SELECT LIST(child) FROM marbletable WHERE marbles <= 100
```

результат будет тем же самым, поскольку выражение NULL <= 10 даёт UNKNOWN. Это не то же самое что TRUE, поэтому Chris и Hadassah не отображены. Если вы хотите что бы в списке были перечислены все "бедные" дети, то измените запрос следующим образом:

```
SELECT LIST(child) FROM marbletable
```
WHERE marbles <= 10 OR marbles IS NULL

Теперь условие поиска становится истинным для Chris и Hadassah, потому что условие “marbles is null” возвращает TRUE в этом случае. Фактически, условие поиска не может быть NULL ни для одного из них.

Наконец, следующие два примера SELECT запросов с параметрами в условии поиска. Как определяются параметры запроса и возможно ли это, зависит от приложения. Обратите внимание, что запросы подобные этим не могут быть выполнены немедленно, они должны быть предварительно подготовлены. После того как параметризованный запрос был подготовлен, пользователь (или вызывающий код) может подставить значения параметров и выполнить его многократно, подставляя перед каждым вызовом новые значения параметров. Как вводятся значения параметров, и проходят ли они предобработку зависит от приложения. В GUI средах пользователь, как правило, вводит значения параметров через одно и более текстовых полей, и щелкает на кнопку "Execute", "Run" или "Refresh".

```
SELECT name, address, phone
FROM stores
WHERE city = ? AND class = ?
```

```
SELECT *
FROM pants
WHERE model = :model AND size = :size AND color = :col
```

Последний запрос не может быть передан непосредственно к движку сервера, приложение должно преобразовать его в другой формат, отображая именованные параметры на позиционные параметры.

**6.1.6. GROUP BY**

Предложение GROUP BY соединяет записи, имеющие одинаковую комбинацию значений полей, указанных в его списке, в одну запись. Агрегатные функции в списке выбора применяются к каждой группе индивидуально, а не для всего набора в целом.

Если список выборки содержит только агрегатные столбцы или столбцы, значения которых не зависят от отдельных строк основного множества, то предложение GROUP BY необязательно. Когда предложение GROUP BY опущено, результирующее множество будет состоять из одной строки (при условии, что хотя бы один агрегатный столбец присутствует).

Если в списке выборки содержатся как агрегатные столбцы, так и столбцы, чьи значения зависят от выбираемых строк, то предложение GROUP BY становится обязательным.

**Синтаксис**

```
SELECT ...
FROM ...
GROUP BY <grouping-item> [, <grouping-item> ...]
[HAVING <grouped-row-condition>] ...
```
Общее правило гласит, что каждый не агрегированный столбец в SELECT списке, должен быть так же включён в GROUP BY список. Вы можете это сделать тремя способами:

1. Копировать выражение дословно из списка выбора, например “class” или “'D:' || upper(doccode)”;

2. Указать псевдоним, если он существует;

3. Задать положение столбца в виде целого числа, которое находится в диапазоне от 1 до количества столбцов в списке SELECT. Целью значения, полученные из выражений, параметров или просто инварианты будут использоваться в качестве таковых в группировке. Они не будут иметь никакого эффекта, поскольку их значение одинаково для каждой строки.

Если вы группируете по позиции столбца или алиасу, то выражение соответствующее этой позиции (алиасу) будет скопировано из списка выборки SELECT. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

В дополнении к требуемым элементам, список группировки так же может содержать:

- Столбцы исходной таблицы, которые не включены в список выборки SELECT, или неагрегатные выражения, основанные на таких столбцах. Добавление таких столбцов может дополнительно разбить группы. Но так как эти столбцы не в списке выборки SELECT, вы не можете сказать, какому значению столбца соответствует значение агрегированной строки. Таким образом, если вы заинтересованы в этой информации, вы так же должны включить этот столбец или выражение в список выборки SELECT, что
возвращает вас к правилу "каждый не агрегированный столбец в списке выборки SELECT должен быть включён в список группировки `GROUP BY`";

• Выражения, которые не зависят от данных из основного набора, т.е. константы, контекстные переменные, некоррелированные подзапросы, возвращающие единственное значение и т.д. Это упоминается только для полноты картины, т.k. добавление этих элементов является абсолютно бессмысленным, поскольку они не повлияют на группировку вообще. "Безвредные, но бесполезные" элементы так же могут фигурировать в списке выбора SELECT без их копирования в список группировки GROUP BY.

Примеры

Когда в списке выбора SELECT содержатся только агрегатные столбцы, предложение GROUP BY необязательно:

```sql
SELECT COUNT(*), AVG(age)
FROM students
WHERE sex = 'M'
```

Этот запрос вернёт одну строку с указанием количества студентов мужского пола и их средний возраст. Добавление выражения, которое не зависит от строк таблицы STUDENTS, ничего не меняет:

```sql
SELECT COUNT(*), AVG(age), current_date
FROM students
WHERE sex = 'M'
```

Теперь строка результата будет иметь дополнительный столбец, отображающий текущую дату, но кроме этого, ничего фундаментального не изменилось. Группировка по-прежнему не требуется.

Тем не менее в обоих приведённых выше примерах это разрешено. Это совершенно справедливо и для запроса:

```sql
SELECT COUNT(*), AVG(age)
FROM students
WHERE sex = 'M'
GROUP BY class
```

и вернёт результат для каждого класса, в котором есть мальчики, перечисляя количество мальчиков и их средний возраст в этой конкретном классе. Если вы также оставите поле CURRENT_DATE, то это значение будет повторяться на каждой строке, что не интересно.

Этот запрос имеет существенный недостаток, хотя он даёт вам информацию о различных классах, но не говорит вам, какая строка к какому классу относится. Для того чтобы получить эту дополнительную часть информации, не агрегатный столбец CLASS должен быть добавлен в список выборки SELECT:
SELECT class, COUNT(*), AVG(age) 
FROM students 
WHERE sex = 'M' 
GROUP BY class

Теперь у нас есть полезный запрос. Обратите внимание, что добавление столбца CLASS делает предложение GROUP BY обязательным. Мы не можем удалять это предложение, так же мы не можем удалить столбец CLASS из списка столбцов.

Результат последнего запроса будет выглядеть примерно так:

<table>
<thead>
<tr>
<th>CLASS</th>
<th>COUNT</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A</td>
<td>12</td>
<td>13.5</td>
</tr>
<tr>
<td>2B</td>
<td>9</td>
<td>13.9</td>
</tr>
<tr>
<td>3A</td>
<td>11</td>
<td>14.6</td>
</tr>
<tr>
<td>3B</td>
<td>12</td>
<td>14.4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Заголовки “COUNT” и “AVG” не очень информативны. В простейшем случае вы можете обойти это, но лучше, если мы дадим им значимые имена с помощью псевдонимов:

SELECT class, 
COUNT(*) AS num_boys, 
AVG(age) AS boys_avg_age 
FROM students 
WHERE sex = 'M' 
GROUP BY class

Как вы помните из формального синтаксиса списка столбцов, ключевое слово AS не является обязательным.

Добавление большего не агрегированных (или точнее строчно зависимых) столбцов требуется добавления их в предложении GROUP BY тоже. Например, вы хотите видеть вышеуказанную информацию о девочках то же, и хотите видеть разницу между интернатами и студентами дневного отделения:

SELECT class, 
sex, 
boarding_type, 
COUNT(*) AS anumber, 
AVG(age) AS avg_age 
FROM students
GROUP BY class, sex, boarding_type

<table>
<thead>
<tr>
<th>CLASS</th>
<th>SEX</th>
<th>BOARDING_TYPE</th>
<th>ANUMBER</th>
<th>AVG_AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A</td>
<td>F</td>
<td>BOARDING</td>
<td>9</td>
<td>13.3</td>
</tr>
<tr>
<td>2A</td>
<td>F</td>
<td>DAY</td>
<td>6</td>
<td>13.5</td>
</tr>
<tr>
<td>2A</td>
<td>M</td>
<td>BOARDING</td>
<td>7</td>
<td>13.6</td>
</tr>
<tr>
<td>2A</td>
<td>M</td>
<td>DAY</td>
<td>5</td>
<td>13.4</td>
</tr>
<tr>
<td>2B</td>
<td>F</td>
<td>BOARDING</td>
<td>11</td>
<td>13.7</td>
</tr>
<tr>
<td>2B</td>
<td>F</td>
<td>DAY</td>
<td>5</td>
<td>13.7</td>
</tr>
<tr>
<td>2B</td>
<td>M</td>
<td>BOARDING</td>
<td>6</td>
<td>13.8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Каждая строка в результирующем наборе соответствует одной конкретной комбинации переменных CLASS, SEX и BOARDING_TYPE. Агрегированные результаты — количество и средний возраст — приведены для каждой из конкретизированной группы отдельно. В результате запроса вы не можете увидеть общённые результаты для мальчиков отдельно или для студентов дневного отделения отдельно. Таким образом, вы должны найти компромисс. Чем больше вы добавляете неагрегатных столбцов, тем больше вы конкретизируете группы, и тем больше вы упускаете общую картину из виду. Конечно, вы все ещё можете получить “большие” агрегаты, с помощью отдельных запросов.

HAVING

Так же, как и предложение WHERE ограничивает строки в наборе данных, теми которые удовлетворяют условию поиска, с той разницей, что предложение HAVING накладывает ограничения на агрегированные строки сгруппированного набора. Предложение HAVING не является обязательным и может быть использовано только в сочетании с предложением GROUP BY.

Условие(я) в предложении HAVING может ссылаться на:

- Любой агрегированный столбец в списке выбора SELECT. Это наиболее широко используемый случай;
- Любое агрегированное выражение, которое не находится в списке выбора SELECT, но разрешено в контексте запроса. Иногда это полезно;
- Любой столбец в списке GROUP BY. Однако более эффективно фильтровать не агрегированные данные на более ранней стадии в предложении WHERE;
- Любое выражение, значение которого не зависит от содержимого набора данных (например, константа или контекстная переменная). Это допустимо, но совершенно бессмысленно, потому что такое условие, не имеющее никакого отношения к самому набору данных, либо подавит весь набор, либо оставит его не тронутым.

Предложение HAVING не может содержать:
• Не агрегированные выражения столбца, которые не находятся в списке GROUP BY;
• Позицию столбца. Целое число в предложении HAVING – просто целое число;
• Псевдонимы столбца — даже если они появляются в предложении GROUP BY.

Примеры

Перестроим наши ранние примеры. Мы можем использовать предложение HAVING для исключения малых групп студентов:

```
SELECT
class,
COUNT(*) AS num_boys,
AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING COUNT(*) >= 5
```

Выберем только группы, которые имеют минимальный разброс по возрасту 1.2 года:

```
SELECT
class,
COUNT(*) AS num_boys,
AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING MAX(age) - MIN(age) > 1.2
```

Обратите внимание, что если вас действительно интересует эта информация, то неплохо бы включить в список выбора min(age) и max(age) или выражение max(age) – min(age).

Следующий запрос отбирает только учеников 3 класса:

```
SELECT
class,
COUNT(*) AS num_boys,
AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING class STARTING WITH '3'
```

Однако гораздо лучше переместить это условие в предложение WHERE:
class,
COUNT(*) AS num_boys,
AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M' AND class STARTING WITH '3'
GROUP BY class

6.1.7. WINDOW

Предложение WINDOW предназначено для задания именованных окон, которые используются оконными функциями. Поскольку выражение окна может быть довольно сложным, и использовать многократно, такая функциональность бывает полезной.

Синтаксис

<query spec> ::= 
SELECT
  [<first clause>] [<skip clause>]
  [<distinct clause>]
  <select list>
  <from clause>
  [<where clause>]
  [<group clause>]
  [<having clause>]
  [<named windows clause>]
  [<order clause>]
  [<rows clause>]
  [<offset clause>] [<limit clause>]
  [<plan clause>]

<named windows clause> ::= 
  WINDOW <window definition> [, <window definition>] ...

>window definition> ::= 
  window-name AS <window specification>

>window specification> ::= 
  (window-name [window partition] [window order] [window frame])

>window partition> ::= PARTITION BY <expr> [, <expr> ...]

>window order> ::= 
  ORDER BY <expr> [<direction>] [nulls placement]
  [, <expr> [<direction>] [nulls placement]] ...

<direction> ::= {ASC | DESC}

-nulls placement> ::= NULLS {FIRST | LAST}
Имя окна может быть использовано в предложении OVER для ссылки на определение окна, кроме того оно может быть использовано в качестве базового окна для другого именованного или встроенного (в предложении OVER) окна. Окна с рамкой (с предложениями RANGE и ROWS) не могут быть использованы в качестве базового окна (но могут быть использованы в предложении OVER window_name). Окно, которое использует ссылку на базовое окно, не может иметь предложение PARTITION BY и не может переопределять сортировку с помощью предложения ORDER BY.

Примеры

Example 227. Использование именованных окон

```
SELECT
    id,
    department,
    salary,
    COUNT(*) OVER w1,
    FIRST_VALUE(salary) OVER w2,
    LAST_VALUE(salary) OVER w2,
    SUM(salary) OVER (w2 ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s
FROM employee
WINDOW w1 AS (PARTITION BY department),
    w2 AS (w1 ORDER BY salary)
ORDER BY department, salary;
```

См. также:
Оконные (аналитические) функции.

6.1.8. PLAN

Предложение PLAN позволяет пользователю указать свой план выполнения запроса, который перекрывает тот план, который оптимизатор сгенерировал автоматически.
Синтаксис

PLAN <plan-expr>

<plan-expr> ::= 
    ( <plan-item> [, <plan-item> ...] )
    | <sorted-item>
    | <joined-item>
    | <merged-item>
    | <hash-item>

<sorted-item> ::= SORT (<plan-item>)

<joined-item> ::= JOIN (<plan-item>, <plan-item> [, <plan-item> ...])

<merged-item> ::= 
    [SORT] MERGE (<sorted-item>, <sorted-item> [, <sorted-item> ...])

[hash-item] ::= HASH (<plan-item>, <plan-item> [, <plan-item> ...])

<plan-item> ::= <basic-item> | <plan-expr>

<basic-item> ::= <relation> 
    { NATURAL
    | INDEX (<indexlist>)
    | ORDER index [INDEX (<indexlist>)]
    }

<relation> ::= table | view [table]

<indexlist> ::= index [, index ...]

Table 81. Параметры предложения PLAN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Имя таблицы или её алиас.</td>
</tr>
<tr>
<td>view</td>
<td>Имя представления.</td>
</tr>
<tr>
<td>index</td>
<td>Имя индекса.</td>
</tr>
</tbody>
</table>

Каждый раз, когда пользователь отправляет запрос ядру Firebird, оптимизатор вычисляет стратегию извлечения данных. Большинство клиентов Firebird имеют возможность отобразить пользователю план извлечения данных. В собственном инструменте isql это делается с помощью команды set plan on. Если вы хотите только увидеть план запроса без его выполнения, то вам необходимо ввести команду set planonly on, после чего будут извлекаться планы запросов без их выполнения. Для возврата isql в режим выполнения запросов введите команду set planonly off.

Более подробный план можно получить при включении расширенного плана. В isql это делается с помощью команды set explain on. Этот план
выводит более подробную информацию о методах доступа используемых оптимизатором, однако его нельзя включить в запрос. Описание расширенного плана выходит за рамки данного руководства.

В большинстве случаев, вы можете доверять тому, что Firebird выберет наиболее оптимальный план запроса. Однако если ваши запросы очень сложны и вам кажется, что они выполняются не эффективно, вам необходимо посмотреть план запроса и подумать можете ли вы улучшить его.

**Простые планы**

Простейшие планы состоят только из имени таблицы и следующим за ним метода извлечения. Например, для неотсортированной выборки из единственной таблицы без предложения `WHERE`:

```sql
SELECT * FROM students
PLAN (students NATURAL)
```

План в EXPLAIN форме:

```
Select Expression
  -> Table "STUDENTS" Full Scan
```

Если есть предложение `WHERE` вы можете указать индекс, который будет использоваться при нахождении совпадений:

```sql
SELECT *
FROM students
WHERE class = '3C'
PLAN (students INDEX (ix_stud_class))
```

План в EXPLAIN форме:

```
Select Expression
  -> Filter
    -> Table "STUDENTS" Access By ID
        -> Bitmap
            -> Index "IX_STUD_CLASS" Range Scan (full match)
```

Директива `INDEX` может использоваться также для условий соединения (которые будут обсуждаться чуть позже). Она содержит список индексов, разделённых запятыми.

Директива `ORDER` определяет индекс, который используется при сортировке набора данных, если присутствуют предложения `ORDER BY` или `GROUP BY`.
План в EXPLAIN форме:

```
Select Expression
  -> Table "STUDENTS" Access By ID
  -> Index "PK_STUDENTS" Full Scan
```

Инструкции ORDER и INDEX могут быть объединены:

```
SELECT *
FROM students
WHERE class >= '3'
PLAN (students ORDER pk_students INDEX (ix_stud_class))
ORDER BY id
```

План в EXPLAIN форме:

```
Select Expression
  -> Filter
    -> Table "STUDENTS" Access By ID
    -> Index "PK_STUDENTS" Full Scan
    -> Bitmap
      -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
```

В инструкциях ORDER и INDEX разрешено указывать один и тот же индекс:

```
SELECT *
FROM students
WHERE class >= '3'
PLAN (students ORDER ix_stud_class INDEX (ix_stud_class))
ORDER BY class
```

План в EXPLAIN форме:

```
Select Expression
  -> Filter
    -> Table "STUDENTS" Access By ID
    -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
    -> Bitmap
      -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
```
Для сортировки наборов данных, когда невозможно использовать индекс (или вы хотите подавить его использование), уберите инструкцию ORDER и предварите выражение плана инструкцией SORT:

```sql
SELECT *  
FROM students  
PLAN SORT (students NATURAL)  
ORDER BY name
```

План в EXPLAIN форме:

```
Select Expression  
  -> Sort (record length: 128, key length: 56)  
  -> Table "STUDENTS" Full Scan
```

Или когда индекс используется для поиска:

```sql
SELECT *  
FROM students  
WHERE class >= '3'  
PLAN SORT (students INDEX (ix_stud_class))  
ORDER BY name
```

План в EXPLAIN форме:

```
Select Expression  
  -> Sort (record length: 136, key length: 56)  
  -> Filter  
    -> Table "STUDENTS" Access By ID  
    -> Bitmap  
    -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
```

Обратите внимание, что инструкция SORT, в отличие от ORDER, находится за пределами скобок. Это отражает тот факт, что строки данных извлекаются неотсортированными и сортируются впоследствии.

При выборке из представления указывается само представление и участвующее в нем таблица. Например, если у вас есть представление FRESHMEN, которое выбирает только студентов первокурсников:

```sql
SELECT *  
FROM freshmen  
PLAN (freshmen students NATURAL)
```

План в EXPLAIN форме:
Или, например:

```
SELECT *
FROM freshmen
WHERE id > 10
PLAN SORT (freshmen students INDEX (pk_students))
ORDER BY name DESC
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 144, key length: 24)
  -> Filter
    -> Table "STUDENTS" as "FRESHMEN" Access By ID
    -> Bitmap
    -> Index "PK_STUDENTS" Range Scan (lower bound: 1/1)
```

Обратите внимание: если вы назначили псевдоним таблице или представлению, то в предложении PLAN необходимо использовать псевдоним, а не оригинальное имя.

**Составные планы**

Если вы делаете соединение, то вы можете указать индекс, который будет использоваться для сопоставления. Кроме того, вы должны использовать директиву JOIN для двух потоков в плане:

```
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
PLAN JOIN (s NATURAL, c INDEX (pk_classes))
```

План в EXPLAIN форме:

```
Select Expression
  -> Nested Loop Join (inner)
    -> Table "STUDENTS" as "S" Full Scan
    -> Filter
      -> Table "CLASSES" as "C" Access By ID
      -> Bitmap
      -> Index "PK.CLASSES" Unique Scan
```

То же самое соединение, отсортированное по индексированному столбцу:
```sql
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
PLAN JOIN (s ORDER pk_students, c INDEX (pk_classes))
ORDER BY s.id
```

План в EXPLAIN форме:

Select Expression
- Nested Loop Join (inner)
  - Table "STUDENTS" as "S" Access By ID
    - Index "PK_STUDENTS" Full Scan
  - Filter
    - Table "CLASSES" as "C" Access By ID
      - Bitmap
        - Index "PK_CLASSES" Unique Scan

И соединение, отсортированное не по индексированному столбцу:

```sql
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
PLAN SORT (JOIN (s NATURAL, c INDEX (pk_classes))))
ORDER BY s.name
```

План в EXPLAIN форме:

Select Expression
- Sort (record length: 152, key length: 12)
  - Nested Loop Join (inner)
    - Table "STUDENTS" as "S" Full Scan
    - Filter
      - Table "CLASSES" as "C" Access By ID
        - Bitmap
          - Index "PK_CLASSES" Unique Scan

Соединение с добавленным условием поиска:

```sql
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
WHERE s.class <= '2'
PLAN SORT (JOIN (s INDEX (fk_student_class), c INDEX (pk_classes)))
ORDER BY s.name
```
План в EXPLAIN форме:

Select Expression
  -> Sort (record length: 152, key length: 12)
     -> Nested Loop Join (inner)
        -> Filter
           -> Table "STUDENTS" as "S" Access By ID
               -> Bitmap
                  -> Index "FK_STUDENT_CLASS" Range Scan (lower bound: 1/1)
           -> Filter
              -> Table "CLASSES" as "C" Access By ID
                 -> Bitmap
                    -> Index "PK_CLASSES" Unique Scan

То же самое, но используется левое внешнее соединение:

```
SELECT s.id, s.name, s.class, c.mentor
FROM classes c
LEFT JOIN students s ON c.name = s.class
WHERE s.class <= '2'
PLAN SORT (JOIN (c NATURAL, s INDEX (fk_student_class)))
ORDER BY s.name
```

План в EXPLAIN форме:

Select Expression
  -> Sort (record length: 192, key length: 56)
     -> Filter
        -> Nested Loop Join (outer)
           -> Filter
              -> Table "CLASSES" as "C" Full Scan
           -> Filter
              -> Table "STUDENTS" as "S" Access By ID
                 -> Bitmap
                    -> Index "FK_STUDENT_CLASS" Range Scan (full match)

Если нет доступных индексов для условия соединения (или вы не хотите его использовать), то возможно соединение потоков с помощью метода HASH или MERGE.

Для соединения методом HASH в плане вместо директивы JOIN используется директива HASH. В этом случае меньший (ведомый) поток целиком вычитывается во внутренний буфер. В процессе чтения к каждому ключу связи применяется хеш-функция и пара {хеш, указатель в буфере} записывается в хеш-таблицу. После чего читается ведущий поток и его ключ связи апробируется в хеш-таблице.

```
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
```
План в EXPLAIN форме:

```
Select Expression
  -> Filter
  -> Hash Join (inner)
    -> Table "STUDENTS" as "S" Full Scan
    -> Record Buffer (record length: 145)
    -> Table "CLASSES" as "C" Full Scan
```

При выполнении соединения методом MERGE план должен сначала отсортировать оба потока по соединяемым столбцам и затем произвести слияние. Это достигается с помощью директив SORT (которую вы уже встречали) и MERGE используемую вместо JOIN.

```sql
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
PLAN MERGE (SORT (c NATURAL), SORT (s NATURAL))
```

Добавление предложения ORDER BY означает, что результат слияния также должен быть отсортирован:

```sql
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
PLAN SORT (MERGE (SORT (c NATURAL), SORT (s NATURAL)))
ORDER BY c.name, s.id
```

И наконец, мы добавляем условие поиска на двух индексированных столбцах таблицы STUDENTS:

```sql
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
WHERE s.id < 10 AND s.class <= '2'
PLAN SORT (MERGE (SORT (c NATURAL),
                   SORT (s INDEX (pk_students, fk_student_class))))
ORDER BY c.name, s.id
```

Как следует из формального определения синтаксиса, JOIN и MERGE могут объединять в плане более двух потоков. Кроме того, каждое выражение плана может использоваться в качестве элемента в охватывающем плане. Это означает, что планы некоторых сложных запросов могут иметь различные уровни вложенности.
Наконец, вместо `MERGE` вы можете писать `SORT MERGE`. Поскольку это не имеет абсолютно никакого значения и может создать путаницу с “настоящей” директивой `SORT` (которая действительно имеет значение), то вероятно лучше придерживаться простой директивы `MERGE`.

Помимо плана для основного запроса вы можете указать план для каждого подзапроса. Например, следующий запрос с указанием планов будет абсолютно правильным.

```sql
SELECT *
FROM COLOR
WHERE EXISTS(
    SELECT *
    FROM HORSE
    WHERE HORSE.CODE_COLOR = COLOR.CODE_COLOR
    PLAN (HORSE INDEX (FK_HORSE_COLOR))
)
PLAN(COLOR NATURAL)
```

### 6.1.9. UNION

Предложение `UNION` объединяет два и более набора данных, тем самым увеличивая общее количество строк, но не столбцов. Наборы данных, принимающие участие в `UNION`, должны иметь одинаковое количество столбцов. Однако столбцы в соответствующих позициях не обязаны иметь один и тот же тип данных, они могут быть абсолютно не связанными.

По умолчанию, объединение подавляет дубликаты строк. `UNION ALL` отображает все строки, включая дубликаты. Необязательное ключевое слово `DISTINCT` делает поведение по умолчанию явным.

**Синтаксис**

```sql
<union> ::=<individual-select>
UNION [DISTINCT | ALL]
<individual-select>
|
 [UNION [DISTINCT | ALL]
 <individual-select>
 ...]
[<union-wide-clauses>]

<individual-select> ::=SELECT
[FIRST m] [SKIP n]
[DISTINCT | ALL] <columns>
FROM <source> [[AS] alias]
[<joins>]
[WHERE <condition>]
[GROUP BY <grouping-list>
```
Отношения получают имена столбцов из первого запроса на выборку. Если вы хотите дать псевдонимы объединяемым столбцам, то делайте это для списка столбцов в самом верхнем запросе на выборку. Псевдонимы в других участвующих в объединении выборках разрешены, и могут быть даже полезными, но они не будут распространяться на уровне объединения.

Если объединение имеет предложение ORDER BY, то единственно возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC | DESC и/или NULLS {FIRST | LAST} директивами. Это также означает, что вы не можете упорядочить объединение ничем, что не является столбцом объединения. (Однако вы можете завернуть его в производную таблицу, которая даст вам все обычные параметры сортировки.)

Объединения позволены в подзапросах любого вида и могут самостоятельно содержать подзапросы. Они также могут содержать соединения (joins), и могут принимать участие в соединениях, если завёрнуты в производную таблицу.

Примеры

Этот запрос представляет информацию из различных музыкальных коллекций в одном наборе данных с помощью объединений:

```sql
SELECT id, title, artist, len, 'CD' AS medium
FROM cds
UNION
SELECT id, title, artist, len, 'LP'
FROM records
UNION
SELECT id, title, artist, len, 'MC'
FROM cassettes
ORDER BY 3, 2 -- artist, title
```

Если id, title, artist и length – единственные поля во всех участвующих таблицах, то запрос может быть записан так:
Уточнение “звёзд” необходимо здесь, потому что они не являются единственным элементом в списке столбцов. Заметьте, что псевдонимы “с” в первой и третьей выборке не кусают друг друга. Они не имеют контекста объединения, а лишь применяются к отдельным запросам на выборку.

Следующий запрос получает имена и телефонные номера переводчиков и корректоров. Те переводчики, которые также работают корректорами, будут отображены только один раз в результирующем наборе, если номера их телефонов одинаковые в обеих таблицах. Тот же результат может быть получен без ключевого слова DISTINCT. Если вместо ключевого слова DISTINCT, будет указано ключевое слово ALL, эти люди будут отображены дважды.

Пример использования UNION в подзапросе:

```sql
SELECT name, phone
FROM translators
UNION DISTINCT
SELECT name, telephone
FROM proofreaders
```

**6.1.10. ORDER BY**

Результат выборки данных при выполнении оператора SELECT по умолчанию никак не упорядочивается (хотя довольно часто происходит упорядочение в хронологическом порядке помещения строк в таблицу операторами INSERT). Предложение ORDER BY позволяет задать необходимый порядок при выборке данных.

**Синтаксис**

```sql
SELECT ... FROM ...
```
Чтобы упорядочить результаты запроса, используйте конструкцию `ORDER BY`:

```
ORDER BY <ordering-item> [, <ordering-item> ...]
```

где `<ordering-item>` может быть:

- `col-name` — полное имя столбца;
- `col-alias` — псевдоним столбца;
- `col-position` — порядковый номер столбца;
- `expression` — произвольное выражение;

После `<ordering-item>` может идти сопоставление порядка сортировки `collation-name` и указание иерархии сортировки `ASCENDING` или `DESCENDING`.

Таблица 82. Параметры предложения `ORDER BY`:

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>Полное имя столбца.</td>
</tr>
<tr>
<td>col-alias</td>
<td>Алиас (псевдоним) столбца.</td>
</tr>
<tr>
<td>col-position</td>
<td>Позиция столбца.</td>
</tr>
<tr>
<td>expression</td>
<td>Произвольное выражение.</td>
</tr>
<tr>
<td>collation-name</td>
<td>Имя сопоставления (порядка сортировки).</td>
</tr>
</tbody>
</table>

В предложении через запятую перечисляются столбцы, по которым нужно упорядочить результирующий набор данных. Можно задавать имя столбца, псевдоним, присвоенный столбцу в списке выбора при помощи ключевого слова `AS`, или порядковый номер столбца в списке выбора. В одном предложении можно для разных столбцов смешивать форму записи. Например, один столбец в списке сортировки может быть задан своим именем, а другой порядковым номером.

Если вы сортируете по позиции столбца или его алиасу, то выражение соответствующее этой позиции (алиасу) будет скопировано из списка выборки `SELECT`. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

В случае сортировки по номеру столбца для запроса вида `SELECT *` сервер раскрывает звёздочку (*) для определения сортируемых столбцов. Однако использование данной особенности в ваших запросах является “плохой практикой”.

**Направление сортировки**

Ключевое слово `ASCENDING` задаёт упорядочение по возрастанию значений. Допустимо сокращение `ASC`. Применяется по умолчанию.

Ключевое слово `DESCENDING` задаёт упорядочение по убыванию значений. Допустимо сокращение `DESC`.

В одном предложении упорядочение по одному столбцу может идти по возрастанию значений, а по другому — по убыванию.
Порядок сравнения

Ключевое слово COLLATE позволяет задать порядок сортировки строкового столбца, если нужен порядок, отличный от того, который был установлен для этого столбца (явно при описании столбца или по умолчанию, принятому для соответствующего набора символов).

Расположение NULL

Ключевое слово NULLS определяет, где в отсортированном наборе данных будут находиться значения NULL соответствующего столбца – в начале выборки (FIRST) или в конце (LAST). По умолчанию принимается NULLS FIRST.

Сортировка частей UNION

Части выборок SELECT, участвующих в объединении UNION, не могут быть отсортированы с использованием предложения ORDER BY. Однако вы можете достичь желаемого результата с использованием производных таблиц или общих табличных выражений. Предложение ORDER BY, записанное последним в объединении, будет применено ко всей выборке в целом, а не к последней его части. Для объединений единственно возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC | DESC и/или NULLS {FIRST | LAST} директивами.

Примеры

В описанном ниже запросе выборка будет отсортирована по возрастанию по столбцам RDB$CHARACTER_SET_ID, RDB$COLLATION_ID таблицы RDB$COLLATIONS:

`SELECT RDB$CHARACTER_SET_ID AS CHARSET_ID, RDB$COLLATION_ID AS COLL_ID, RDB$COLLATION_NAME AS NAME FROM RDB$COLLATIONS ORDER BY RDB$CHARACTER_SET_ID, RDB$COLLATION_ID`

То же самое, но сортировка производится по псевдонимам столбцов:

`SELECT RDB$CHARACTER_SET_ID AS CHARSET_ID, RDB$COLLATION_ID AS COLL_ID, RDB$COLLATION_NAME AS NAME FROM RDB$COLLATIONS ORDER BY CHARSET_ID, COLL_ID`

В следующем запросе производится сортировка, по номерам столбцов:

`SELECT RDB$CHARACTER_SET_ID AS CHARSET_ID, RDB$COLLATION_ID AS COLL_ID,`
Как было выше сказано, такая сортировка тоже допустима, но не рекомендуется:

```sql
SELECT *
FROM RDB$COLLATIONS
ORDER BY 3, 2
```

В данном запросе сортировка происходит по второму столбцу таблицы 800KS:

```sql
SELECT BOOKS.*, FILMS.DIRECTOR
FROM BOOKS, FILMS
ORDER BY 2
```

Обратите внимание на то, что выражения, результатом вычисления которых должны быть целые неотрицательные числа, будут интерпретироваться как номер столбца и вызовут исключение, если они не будут в диапазоне от 1 до числа столбцов.

```sql
SELECT X, Y, NOTE
FROM PAIRS
ORDER BY X+Y DESC
```

Примечания:

- Число, возвращаемое функцией или процедурой из UDF или хранимой процедуры, непредсказуемо, независимо от того, определена сортировка самим выражением или номером столбца;
- Только неотрицательные целые числа интерпретируются как номер столбца. Целое число, полученное однократным вычислением выражения или заменой параметра, запоминается как целочисленная постоянная величина, так как это значение одинаково для всех строк.

Сортировка по убыванию значений столбца PROCESS_TIME с размещением значений NULL в начале выборки:

```sql
SELECT *
FROM MSG
ORDER BY PROCESS_TIME DESC NULLS FIRST
```
Сортировка выборки полученной объединением выборок из двух запросов. Выборка сортируется по убыванию значений второго столбца с размещением NULL значений в конце списка и возрастанием значений первого столбца с размещением NULL значений в начале списка.

```
SELECT
  DOC_NUMBER, DOC_DATE
FROM PAYORDER
UNION ALL
SELECT
  DOC_NUMBER, DOC_DATE
FROM BUDGORDER
ORDER BY 2 DESC NULLS LAST, 1 ASC NULLS FIRST
```

### 6.1.11. ROWS

**Назначение**

Получение части строк из упорядоченного набора.

**Синтаксис**

```
SELECT <columns> FROM ...
[WHERE ...]
[ORDER BY ...]
ROWS m [TO n]
```

**Table 83. Параметры предложения ROWS**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>m, n</td>
<td>Любые целочисленные выражения.</td>
</tr>
</tbody>
</table>

Предложение ROWS было введено для совместимости с Interbase 6.5 и выше.

В отличие от FIRST и SKIP, выражение ROWS принимает все типы целочисленных (integer) выражений в качестве аргумента – без скобок! Конечно, скобки могут требоваться для правильных вычислений внутри выражения, и вложенный запрос также должен быть обернут в скобки.

- Нумерация записей в наборе данных начинается с 1.
- И FIRST/SKIP, и ROWS могут быть использованы без выражения ORDER BY, хотя это редко имеет смысл, за исключением случая, когда необходимо быстро взглянуть на данные таблицы – получаемые строки при этом будут чаще всего в случайном порядке. В этом случае запрос вроде `SELECT * FROM TABLE1 ROWS 20` вернёт 20 первых записей, а не целую таблицу (которая может очень большой).

Вызов `ROWS m` приведёт к возвращению первых `m` записей из набора данных.
Особенности при использовании **ROWS** с одним аргументом

- Если \( m \) больше общего числа записей в возвращаемом наборе данных, то будет возвращён весь набор данных;
- Если \( m = 0 \), то будет возвращён пустой набор данных;
- Если \( m < 0 \), выдаётся ошибка.

В случае указания **ROWS** \( m \) TO \( n \), то будут возвращены записи с \( m \) по \( n \) из набора данных.

Особенности при использовании **ROWS** с двумя аргументами

- Если \( m \) больше общего количества строк в наборе данных и \( n \geq m \), то будет возвращён пустой набор данных;
- Если число \( m \) не превышает общего количества строк в наборе данных, а \( n \) превышает, то выборка ограничивается строками, начиная с \( m \) до конца набора данных;
- Если \( m < 1 \) и \( n < 1 \), то оператор SELECT выдаст ошибку;
- Если \( n = m - 1 \), то будет возвращён пустой набор данных;
- Если \( n < m - 1 \), то оператор SELECT выдаст ошибку.

Замена **FIRST … SKIP**

В сущности, **ROWS** заменяет собой нестандартные выражения **FIRST** и **SKIP**, за исключением единственного случая, когда указывается только **SKIP**, т.е. когда возвращается весь набор данных за исключением пропуска указанного числа записей с начала.

Для того, что реализовать такое поведение с помощью **ROWS**, необходимо указать второй аргумент, заведомо больший, чем размер возвращаемого набора данных. Или запросить число записей в возвращаемом наборе через подзапрос.

Совместное использование **FIRST … SKIP** и **ROWS**

Нельзя использовать **ROWS** вместе с **FIRST/SKIP** в одном и том же операторе SELECT, но можно использовать разный синтаксис в разных подзапросах.

Использование **ROWS** в **UNION**

При использовании **ROWS** с выражением **UNION**, он будет применяться к объединённому набору данных, и должен быть помещён после последнего SELECT.

При необходимости ограничить возвращаемые наборы данных одного или нескольких операторов SELECT внутри **UNION**, можно воспользоваться следующими вариантами:

1. Использовать **FIRST/SKIP** в этих операторах SELECT. Необходимо помнить, что нельзя локально использовать выражение ORDER BY в SELECT внутри **UNION** – только глобально, ко всему суммарному набору данных;
2. Преобразовать SELECT в производные таблицы с выражениями **ROWS**.
Примеры

Ниже приведены примеры, ранее использованные для демонстрации FIRST/SKIP.

Следующий запрос вернёт первые 10 имен из таблицы PEOPLE (имена также будут отсортированы, см. ORDER BY).

```sql
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 1 TO 10
```

или его эквивалент

```sql
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 10
```

Следующий запрос вернёт все записи из таблицы PEOPLE, за исключением первых 10 имен:

```sql
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 11 TO (SELECT COUNT(*) FROM People)
```

А этот запрос вернёт последние 10 записей (обратите внимание на скобки):

```sql
SELECT id, name
FROM People
ORDER BY name ASC
ROWS (SELECT COUNT(*) - 9 FROM People) TO (SELECT COUNT(*) FROM People)
```

Этот запрос вернёт строки 81-100 из таблицы PEOPLE:

```sql
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 81 TO 100
```

См. также:

"FETCH, OFFSET", "FIRST, SKIP".
6.1.12. FETCH, OFFSET

Предложения FETCH и OFFSET являются SQL:2008 совместимым эквивалентом предложениям FIRST/SKIP и альтернативой предложению ROWS. Предложение OFFSET указывает, какое количество строк необходимо пропустить. Предложение FETCH указывает, какое количество строк необходимо получить.

Предложения OFFSET и FETCH могут применяться независимо уровня вложенности выражений запросов.

Синтаксис

```
SELECT <columns> FROM ...
[WHERE ...]
[ORDER BY ...]
[OFFSET <m> {ROW | ROWS}]
[FETCH {FIRST | NEXT} [ <n> ] { ROW | ROWS } ONLY]
```

Таблица 84. Параметры предложений OFFSET и FETCH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer-literal</td>
<td>Целочисленный литерал</td>
</tr>
<tr>
<td>query-parameter</td>
<td>Параметр запрос. ? в DSQL и :paramname в PSQL</td>
</tr>
</tbody>
</table>

- Firebird не поддерживает указание FETCH в процентах, определённое в стандарте.
- Firebird не поддерживает предложение FETCH с опцией WITH TIES, которая определена в стандарте.
- FIRST … SKIP и ROWS являются нестандартными альтернативами.
- Предложения OFFSET и/или FETCH не могут быть объединены с предложениями ROWS или FIRST/SKIP в одном выражении запроса.
- Выражения, ссылки на столбцы и т.д. недопустимы в любом из предложений.
- В отличие от предложения ROWS, предложения OFFSET и FETCH допустимы только в операторе SELECT.

Примеры использования OFFSET и FETCH

Следующий запрос возвращает все строки кроме первых 10, упорядоченных по столбцу COL1:

```
SELECT *
```
В этом примере возвращаются первые 10 строк, упорядоченных по столбцу COL1:

```
SELECT *
FROM T1
ORDER BY COL1
FETCH FIRST 10 ROWS ONLY
```

Использование предложений OFFSET и FETCH в производной таблице, результат которой ограничивается ещё раз во внешнем запросе:

```
SELECT *
FROM (  
    SELECT *
    FROM T1
    ORDER BY COL1 DESC
    OFFSET 1 ROW
    FETCH NEXT 10 ROWS ONLY
  ) a
ORDER BY a.COL1
FETCH FIRST ROW ONLY
```

См. также:
ROWS, "FIRST, SKIP".

**6.1.13. FOR UPDATE [OF]**

Синтаксис

```
SELECT ...  
FROM single_table  
[WHERE ...]
[FOR UPDATE [OF <column-names>]]
```

Предложение FOR UPDATE не делает то, что от него ожидается. В настоящее время единственный эффект от его использования заключается лишь в отключении упреждающей выборки в буфер.

Это, вероятно, изменится в будущем: план состоит в том, чтобы проверять курсоры, отмеченные как FOR UPDATE, действительно ли они обновляемые, и отклонять позиционированные обновления и удаления для курSORов,
оцененных как необновляемый.

Предложение `OF` не делает ничего вообще.

**6.1.14. WITH LOCK**

**Назначение**

Пессимистическая блокировка.

**Доступно**

DSQL, PSQL

**Синтаксис**

```
SELECT ...
FROM single_table
[WHERE ...]
[FOR UPDATE [OF <column-names>]]
WITH LOCK
```  

Опция `WITH LOCK`, обеспечивает возможность ограниченной явной пессимистической блокировки для осторожного использования в затронутых наборах строк:

a. крайне малой выборки (в идеале из одной строки) и
b. при контроле из приложения.

**Только для экспертов**

Пессимистическая блокировка редко требуется при работе с Firebird. Эту функцию можно использовать только хорошо понимая её.

Требуется хорошее знание различных уровней изоляции и других параметров транзакций прежде чем использовать явную блокировку в вашем приложении.

При успешном выполнении предложения `WITH LOCK` будут заблокированы выбранные строки данных и таким образом запрещён доступ на их изменение в рамках других транзакций до момента завершения вашей транзакции.

Предложение `WITH LOCK` доступно только для выборки данных (SELECT) из одной таблицы. Предложение `WITH LOCK` нельзя использовать:

- в подзапросах;
- в запросах с объединением нескольких таблиц (JOIN);
- с оператором `DISTINCT`, предложением `GROUP BY` и при использовании любых агрегатных функций;
- при работе с представлениями;
- при выборке данных из селективных хранимых процедур;
• при работе с внешними таблицами.

Сервер, в свою очередь, для каждой записи, подпадающей под явную блокировку, возвращает версию записи, которая является в настоящее время подтверждённой (актуальной), независимо от состояния базы данных, когда был выполнен оператор выборки данных, или исключение при попытке обновления заблокированной записи.

Ожидаемое поведение и сообщения о конфликте зависят от параметров транзакции, определённых в TPB (Transaction Parameters Block):

**Table 85. Влияние параметров TPB на явную блокировку**

<table>
<thead>
<tr>
<th>Режим TPB</th>
<th>Поведение</th>
</tr>
</thead>
<tbody>
<tr>
<td>isc_tpb_consistency</td>
<td>Явные блокировки переопределяются неявыми или явными блокировками табличного уровня и игнорируются.</td>
</tr>
<tr>
<td>isc_tpb_concurrency +</td>
<td>При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления.</td>
</tr>
<tr>
<td>isc_tpb_nowait</td>
<td></td>
</tr>
<tr>
<td>isc_tpb_concurrency +</td>
<td>При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления. Если в активной транзакции идёт редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировки записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Это означает, что при изменении версии записи и подтверждении транзакции с блокировкой возникает исключение конфликта обновления.</td>
</tr>
<tr>
<td>isc_tpb_wait</td>
<td></td>
</tr>
<tr>
<td>isc_tpb_read_committed +</td>
<td>Если есть активная транзакция, редактирующая запись (с использованием явной блокировки или нормальной оптимистической блокировки записи), то сразу же возникает исключение конфликта обновления.</td>
</tr>
<tr>
<td>isc_tpb_nowait</td>
<td></td>
</tr>
<tr>
<td>isc_tpb_read_committed +</td>
<td>Если в активной транзакции идёт редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировки записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи.</td>
</tr>
<tr>
<td>isc_tpb_wait</td>
<td></td>
</tr>
</tbody>
</table>

Использование предложения **FOR UPDATE**

Если предложение **FOR UPDATE** предшествует предложению **WITH LOCK**, то буферизация выборки не используется. Таким образом, блокировка применяется к каждой строке, одна за другой, по мере извлечения записей. Это делает возможным ситуацию, в которой...
успешная блокировка данных перестаёт работать при достижении в выборке строки, заблокированной другой транзакцией.

Кроме того, некоторые компоненты доступа позволяют установить размер буфера выборки и уменьшить его до 1 записи. Это позволяет вам заблокировать и редактировать строку до выборки и блокировки следующей или обрабатывать ошибки, не отменяя действий вашей транзакции.

Опциональное предложение “OF <column-names>” не делает ничего вообще.

See also
FOR UPDATE [OF]

Как сервер работает с WITH LOCK

Попытка редактирования записи с помощью оператора UPDATE, заблокированной другой транзакцией, приводит к вызову исключения конфликта обновления или ожиданию завершения блокирующей транзакции – в зависимости от режима TPB. Поведение сервера здесь такое же, как если бы эта запись уже была изменена блокирующей транзакцией.

Нет никаких специальных кодов gdscode, возвращаемых для конфликтов обновления, связанных с пессимистической блокировкой.

Сервер гарантирует, что все записи, возвращённые явным оператором блокировки, фактически заблокированы и действительно соответствуют условиям поиска, заданным в операторе WHERE, если эти условия не зависят ни от каких других таблиц, не имеется операторов соединения, подзапросов и т.п. Это также гарантирует то, что строки, не попадающие под условия поиска, не будут заблокированы. Это не даёт гарантии, что нет строк, которые попадают под условия поиска, и не заблокированы.

Такая ситуация может возникнуть, если в другой, параллельной транзакции подтверждаются изменения в процессе выполнения текущего оператора блокировки.

Сервер блокирует строки по мере их выборки. Это имеет важные последствия, если вы блокируете сразу несколько строк. Многие методы доступа к базам данных Firebird по умолчанию используют для выборки данных пакеты из нескольких сотен строк (так называемый "буфер выборки"). Большинство компонентов доступа к данным не выделяют строки, содержащиеся в последнем принятом пакете, и для которых произошёл конфликт обновления.

Предостережения при использовании WITH LOCK

• Откат неявной или явной точки сохранения отменяет блокировку записей, которые изменялись в рамках её действий, но ожидающие окончания блокировки транзакции при этом не уведомляются. Приложения не должны зависеть от такого поведения, поскольку в будущем оно может быть изменено;
• Хотя явные блокировки могут использоваться для предотвращения и/или обработки необычных ошибок конфликтов обновления, объем ошибок обновления (deadlock) вырастет, если вы тщательно не разработаете свою стратегию блокировки и не будете ей строго управлять;

• Большинство приложений не требуют явной блокировки записей. Основными целями явной блокировки являются: 1) предотвращение дорогостоящей обработки ошибок конфликта обновления в сильно загруженных приложениях и 2) для поддержания целостности объектов, отображаемых из реляционной базы данных в кластеризуемой среде. Если использование вами явной блокировки не подпадает под одну из этих двух категорий, то это является неправильным способом решения задач в Firebird;

• Явная блокировка — это расширенная функция; не злоупотребляйте её использованием! В то время как явная блокировка может быть очень важной для веб-сайтов, обрабатывающих тысячи параллельных пишущих транзакций или для систем типа ERP/CRM, работающих в крупных корпорациях, большинство прикладных программ не требуют её использования.

Примеры использования явной блокировки

Example 228. Блокировка одной записи

```sql
SELECT * 
FROM DOCUMENT 
WHERE DOCUMENT_ID=? WITH LOCK
```

Example 229. Блокировка нескольких записей с их последовательной обработкой курсором DSQL:

```sql
SELECT * 
FROM DOCUMENT 
WHERE PARENT_ID=? 
FOR UPDATE WITH LOCK
```

6.1.15. INTO

Назначение
Передача результатов SELECT в переменные.

Доступно в:
PSQL

Синтаксис:

```sql
SELECT [...] <column-list> 
FROM ... 
[...]```
В PSQL (хранимых процедурах, триггерах и др.) результаты выборки команды `SELECT` могут быть построчно загружены в локальные переменные (число, порядок и типы локальных переменных должны соответствовать полям `SELECT`). Часто такая загрузка — единственный способ что-то сделать с возвращаемыми значениями.

Простой оператор `SELECT` может быть использован в PSQL, только если он возвращает не более одной строки, то есть, если это запрос типа singleton (одиночка). Для запросов, возвращающих несколько строк, PSQL предлагает использовать оператор `FOR SELECT`.

В случае, когда запрос не возвращает данных (ноль строк), значения переменных в списке `INTO` не изменяется.

Также, PSQL поддерживает оператор `DECLARE CURSOR`, который связывает именованный курсор с определенной командой `SELECT` — и этот курсор впоследствии может быть использован для навигации по возвращаемому набору данных.

В PSQL выражение `INTO` должно появляться в самом конце команды `SELECT`.

**Обратите внимание.**
В PSQL двоеточие перед именами переменных является опциональным.

**Примеры**

В PSQL, можно присвоить значения `min_amt`, `avg_amt` и `max_amt` заранее объявленным переменным или выходным параметрам:

```sql
SELECT
    MIN(amount),
    AVG(CAST(amount AS float)),
    MAX(amount)
FROM orders
WHERE artno = 372218
INTO min_amt,
    avg_amt,
    max_amt;
```

В данном запросе `CAST` служит для корректного вычисления среднего значения. Если не привести значение к `float`, то среднее значение будет обрезано до ближайшего целого значения.

В триггере:

```sql
SELECT LIST(name, ' ', ')
```
FROM persons p
WHERE p.id IN (new.father, new.mother)
INTO new.parentnames;

6.1.16. Общие табличные выражения CTE ("WITH ... AS ... SELECT")

Общие табличные выражения (Common Table Expressions), сокращённо CTE, описаны как виртуальные таблицы или представления, определённые в преамбуле основного запроса, которые участвуют в основном запросе. Основной запрос может ссылаться на любое CTE из определённых в преамбуле, как и при выборке данных из обычных таблиц или представлений. CTE могут быть рекурсивными, т.е. ссылаяющимися сами на себя, но не могут быть вложенными.

Синтаксис

<cte-construct> ::= 
<cte-defs> <main-query>

<cte-defs> ::= WITH [RECURSIVE] <cte> [, <cte> ...]
<cte> ::= name [((column-list))] AS (<cte-stmt>)
<column-list> ::= column-alias [, column-alias ...]

Table 86. Параметры CTE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>cte-stmt</td>
<td>Любой оператор SELECT или UNION.</td>
</tr>
<tr>
<td>main-query</td>
<td>Основной оператор SELECT, который может ссылаться на любое CTE из найденных в преамбуле.</td>
</tr>
<tr>
<td>name</td>
<td>Аlias табличного выражения.</td>
</tr>
<tr>
<td>column-alias</td>
<td>Аlias столбца табличного выражения.</td>
</tr>
</tbody>
</table>

Примечания

- Определение CTE может содержать любой правильный оператор SELECT, если он не содержит преамбулы "WITH..." (операторы WITH не могут быть вложенными);
- CTE могут использовать друг друга, но ссылки не должны иметь циклов;
- CTE могут быть использованы в любой части главного запроса или другого табличного выражения и сколько угодно раз;
- Основной запрос может ссылаться на CTE несколько раз, но с разными алиасами;
- CTE могут быть использованы в операторах INSERT, UPDATE и DELETE как подзапросы;
- Если объявленное CTE не использовано, то будет выдано предупреждение "CTE cte is not used in query". В более ранних версиях вместо предупреждения выдавалась ошибка;
• CTE могут быть использованы и в PSQL в `FOR` циклах:

```sql
FOR
  WITH MY_RIVERS AS (
    SELECT *
    FROM RIVERS
    WHERE OWNER = 'me'
  )
  SELECT NAME,
        LENGTH
  FROM MY_RIVERS
  INTO :RNAME,
       :RLEN
DO
BEGIN...
END
```

Примеры

*Example 230. Запрос с использованием CTE*

```sql
WITH DEPT_YEAR_BUDGET AS ( 
  SELECT
    FISCAL_YEAR,
    DEPT_NO,
    SUM(PROJECTED_BUDGET) BUDGET
  FROM PROJ_DEPT_BUDGET
  GROUP BY FISCAL_YEAR, DEPT_NO
 )
SELECT
  D.DEPT_NO,
  D.DEPARTMENT,
  DYB_2008.BUDGET BUDGET_08,
  DYB_2009.BUDGET AS BUDGET_09
FROM DEPARTMENT D
  LEFT JOIN DEPT_YEAR_BUDGET DYB_2008
    ON D.DEPT_NO = DYB_2008.DEPT_NO AND
       DYB_2008.FISCAL_YEAR = 2008
  LEFT JOIN DEPT_YEAR_BUDGET DYB_2009
    ON D.DEPT_NO = DYB_2009.DEPT_NO AND
       DYB_2009.FISCAL_YEAR = 2009
WHERE EXISTS (SELECT *
              FROM PROJ_DEPT_BUDGET B
              WHERE D.DEPT_NO = B.DEPT_NO)
```
Рекурсивные CTE

Рекурсивное (ссылающееся само на себя) CTE это ОБЪЕДИНЕНИЕ, у которого должен быть, по крайней мере, один не рекурсивный элемент, к которому привязываются остальные элементы объединения. Не рекурсивный элемент помещается в CTE первым. Рекурсивные члены отделяются от не рекурсивных и друг от друга с помощью UNION ALL. Объединение не рекурсивных элементов может быть любого типа.

Рекурсивное CTE требует наличия ключевого слова RECURSIVE справа от WITH. Каждый рекурсивный член объединения может сослаться на себя только один раз и это должно быть сделано в предложении FROM.

Главным преимуществом рекурсивных CTE является то, что они используют гораздо меньше памяти и процессорного времени, чем эквивалентные рекурсивные хранимые процедуры.

Выполнение рекурсивного CTE

Выполнение рекурсивного CTE с точки зрения сервера Firebird можно описать следующим образом:

- Сервер начинает выполнение с не рекурсивного члена;
- Для каждой выбранной строки из нерекурсивного части выполняется каждый рекурсивный член один за другим, используя текущие значения из предыдущей итерации как параметры;
- Если во время выполнения экземпляр рекурсивного элемента не выдаёт строк, цикл выполнения переходит на предыдущий уровень и получает следующую строку от внешнего для него набора данных.

Примеры

*Example 231. Рекурсивное CTE*

```
WITH RECURSIVE DEPT_YEAR_BUDGET AS (  SELECT
    FISCAL_YEAR,
    DEPT_NO,
    SUM(PROJECTED_BUDGET) BUDGET
    FROM PROJ_DEPT_BUDGET
    GROUP BY FISCAL_YEAR, DEPT_NO
  ),
  DEPT_TREE AS (  SELECT
    DEPT_NO,
    HEAD_DEPT,
    DEPARTMENT,
    CAST('' AS VARCHAR(255)) AS INDENT
    FROM DEPARTMENT
    WHERE HEAD_DEPT IS NULL
  )
```
Следующий пример выводит родословную лошади. Основное отличие состоит в том, что рекурсия идёт сразу по двум веткам дерева родословной.

WITH RECURSIVE PEDIGREE (CODE_HORSE, CODE_FATHER, CODE_MOTHER, NAME, MARK, DEPTH) AS (SELECT HORSE.CODE_HORSE, HORSE.CODE_FATHER, HORSE.CODE_MOTHER, HORSE.NAME, CAST('' AS VARCHAR(80)), 0 FROM HORSE WHERE HORSE.CODE_HORSE = :CODE_HORSE UNION ALL SELECT HORSE.CODE_HORSE, HORSE.CODE_FATHER, HORSE.CODE_MOTHER, HORSE.NAME, 'F' || PEDIGREE.MARK, PEDIGREE.DEPTH + 1
FROM HORSE
JOIN PEDIGREE
ON HORSE.CODE_HORSE = PEDIGREE.CODE_FATHER
WHERE
- ограничение глубины рекурсии
PEDIGREE.DEPTH < :MAX_DEPTH
UNION ALL
SELECT
HORSE.CODE_HORSE,
HORSE.CODE_FATHER,
HORSE.CODE_MOTHER,
HORSE.NAME,
'M' || PEDIGREE.MARK,
PEDIGREE.DEPTH + 1
FROM HORSE
JOIN PEDIGREE
ON HORSE.CODE_HORSE = PEDIGREE.CODE_MOTHER
WHERE
- ограничение глубины рекурсии
PEDIGREE.DEPTH < :MAX_DEPTH
)
SELECT
CODE_HORSE,
NAME,
MARK,
DEPTH
FROM PEDIGREE

Примечания для рекурсивного CTE:

- В рекурсивных членах объединения не разрешается использовать агрегаты (DISTINCT, GROUP BY, HAVING) и агрегатные функции (SUM, COUNT, MAX и т.п.);
- Рекурсивная ссылка не может быть участником внешнего объединения OUTER JOIN;
- Максимальная глубина рекурсии составляет 1024;
- Рекурсивный член не может быть представлен в виде производной таблицы.

6.2. INSERT

Назначение
Вставка данных в таблицу.

Доступно в
DSQL, ESQL, PSQL

Синтаксис

```
INSERT INTO target [(<column_list>)]
```
### Table 87. Параметры оператора INSERT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Имя таблицы или представления, в которую происходит вставка новой записи или записей.</td>
</tr>
<tr>
<td>col-name</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
<tr>
<td>value_expression</td>
<td>Выражение, значение которого используется для вставки в таблицу или возврата в RETURNING.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context-variable</td>
<td>Контекстная переменная.</td>
</tr>
<tr>
<td>other-single-value-expr</td>
<td>Любое другое выражение, возвращающее единственное значение типа данных Firebird или NULL.</td>
</tr>
<tr>
<td>return_expression</td>
<td>Выражение, возвращаемое в предложении RETURNING.</td>
</tr>
<tr>
<td>collation</td>
<td>Существующее имя сортировки (только для символьных типов).</td>
</tr>
<tr>
<td>alias</td>
<td>Псевдоним для выражения, возвращаемого в предложении RETURNING.</td>
</tr>
<tr>
<td>varname</td>
<td>Имя PSQL переменной.</td>
</tr>
</tbody>
</table>
Оператор INSERT добавляет строки в таблицу или в одну, или более таблиц представления. Если значения столбцов указаны в разделе VALUES, то будет вставлена одна строка. Значения столбцов также могут быть получены из оператора SELECT, в этом случае может быть вставлено от нуля и более строк. В случае DEFAULT VALUES, значения можно не указывать, и вставлена будет одна строка.

• Один столбец не может быть указан более одного раза в списке столбцов;
• При возвращении значений столбцов вставленной записи с помощью INTO в контекстные переменные NEW.columnname в триггерах не обязательно использовать префикс двоеточия (“:”).

### 6.2.1. INSERT … VALUES

В списке VALUES должны быть указаны значения для всех столбцов в списке столбцов в том же порядке и совместимые по типу. Если список столбцов отсутствует, то значения должны быть указаны для каждого столбца таблицы или представления (исключая вычисляемые столбцы).

**Вводный синтаксис** даёт возможность определить набор символов для значений строковых констант (литералов). Вводный синтаксис работает только с литералами строк: он не может быть применён к строковым переменным, параметрам, ссылкам на столбцы или значения, выражениям.

**Example 232. Использование INSERT с предложением VALUES**

```sql
INSERT INTO cars (make, model, byyear)
VALUES ('Ford', 'T', 1908);

INSERT INTO cars
VALUES ('Ford', 'T', 1908, 'USA', 850);

-- обратите внимание на префикс '_' (introducer syntax)
INSERT INTO People
VALUES (_ISO8859_1 'Hans-Jörg Schäfer');
```

**Ключевое слово DEFAULT**

В списке VALUES вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

Если ключевое слово DEFAULT указано для столбца, определенного как GENERATED BY DEFAULT AS IDENTITY, то столбец получит следующее значение идентификации, так как будто этот
столбец не был указан в запросе вовсе.

Example 233. Использование ключевого слова DEFAULT в операторе INSERT

```sql
CREATE TABLE cars (  
    ID INTEGER GENERATED BY DEFAULT AS IDENTITY,  
    BYYEAR SMALLINT DEFAULT 1990 NOT NULL,  
    NAME VARCHAR(45),  
    CONSTRAINT pk_cars PRIMARY KEY (ID)  
);

-- в столбец BYYEAR попадёт значение 1990
INSERT INTO cars (byyear, name)  
VALUES (DEFAULT, 'Ford Focus');

-- в столбец id попадёт значение 2, как будто мы не указывали значение для id
INSERT INTO cars (id, byyear, name)  
VALUES (DEFAULT, 1996, 'Ford Mondeo');
```

6.2.2. INSERT … SELECT

В этом случае выходные столбцы оператора SELECT, должны предоставить значения для каждого целевого столбца в списке столбцов, в том же порядке и совместимого типа. Если список столбцов отсутствует, то значения должны быть предоставлены для каждого столбца таблицы или представления (исключая вычисляемые столбцы).

Example 234. Использование оператора в форме INSERT … SELECT

```sql
INSERT INTO cars (make, model, byyear)  
SELECT make, model, byyear  
FROM new_cars;

INSERT INTO cars  
SELECT *  
FROM new_cars;

INSERT INTO Members (number, name)  
SELECT number, name  
FROM NewMembers  
WHERE Accepted = 1  
UNION ALL  
SELECT number, name  
FROM SuspendedMembers  
WHERE Vindicated = 1

INSERT INTO numbers(num)  
WITH RECURSIVE r(n) AS (  
    SELECT 1 FROM rdb$database
    UNION  
    (SELECT num + 1 FROM numbers AS num)  
)
```
Конечно, имена столбцов в таблице источнике необязательно должны быть такими же, как и в таблице приёмнике.

Любой тип оператора SELECT разрешён, пока его выходные столбцы точно соответствуют столбцам вставки по числу и типу. Типы не должны быть точно такими же, но они должны быть совместимыми по присваиванию.

6.2.3. INSERT … DEFAULT VALUES

Предложение DEFAULT VALUES позволяет вставлять записи без указания значений вообще, ни непосредственно (в предложении VALUES), ни из оператора SELECT. Это возможно, только если для каждого NOT NULL поля и полей, на которые наложены другие ограничения, или имеются допустимые объявленные значения по умолчанию, или эти значения устанавливаются в BEFORE INSERT триггере.

Example 235. Использование DEFAULT VALUES в операторе INSERT

```sql
INSERT INTO journal
DEFAULT VALUES
RETURNING entry_id
```

6.2.4. Директива OVERRIDING

Значения столбцов идентификации (GENERATED BY DEFAULT AS IDENTITY) могут быть переопределены в операторах INSERT, UPDATE OR INSERT, MERGE. Для этого просто достаточно указать значение столбца в списке значений. Однако для столбцов определенных как GENERATED ALWAYS это недопустимо.

Директива OVERRIDING SYSTEM VALUE позволяет заменить сгенерированное системой значение на значение указанное пользователем. Директива OVERRIDING SYSTEM VALUE вызовет ошибку, если в таблице нет столбцов идентификации или если они определены как GENERATED BY DEFAULT AS IDENTITY.

Example 236. Использование директивы OVERRIDING SYSTEM VALUE в операторе INSERT

```sql
CREATE TABLE objects (  
id INT GENERATED ALWAYS AS IDENTITY,  
name CHAR(50));

-- будет вставлено значение с кодом 11  
INSERT INTO objects (id, name)
```
Директива OVERRIDE USER VALUE выполняет обратную задачу, т.е. заменяет значение указанное пользователем на значение сгенерированное системой, если столбец идентификации определён как GENERATED BY DEFAULT AS IDENTITY. Директива OVERRING USER VALUE вызовет ошибку, если в таблице нет столбцов идентификации или если они определены как GENERATED ALWAYS AS IDENTITY.

**Example 237. Использование директивы OVERRIDE USER VALUE в операторе INSERT**

```sql
CREATE TABLE objects ( 
  id INT GENERATED BY DEFAULT AS IDENTITY, 
  name CHAR(50));

-- значение 12 будет проигнорировано
INSERT INTO objects (id, name) 
OVERRIDING SYSTEM VALUE 
VALUES (12, 'Laptop');
```

См. также:
Столбцы идентификации (автоинкремент).

### 6.2.5. RETURNING

Оператор INSERT вставляющий только одну строку может включать необязательное предложение RETURNING для возврата значений из вставленной строки. Если предложение указано, то оно может содержать любые столбцы, указанные в операторе, или другие столбцы и выражения. Вместо списка столбцов вы можете указать звёздочку (*) для возврата всех значений столбцов таблицы. Возвращаемые значения содержат все изменения, произведённые в триггерах BEFORE.

В DSQL, оператор с RETURNING всегда возвращает только одну запись. Если указано предложение RETURNING, и обнаружено более одной вставленной записи, выдаётся сообщение об ошибке. Это поведение может быть изменено в последующих версиях Firebird.

**Example 238. Использование предложения RETURNING в операторе INSERT**

```sql
INSERT INTO Scholars (firstname, lastname, address, 
phone, email) 
VALUES ( 
  'Henry', 'Higgins', '27A Wimpole Street',
```

```
```


• Предложение RETURNING поддерживается только для INSERT … VALUES и одиночных INSERT … SELECT.

• В DSQL оператор с использованием RETURNING всегда возвращает ровно одну запись. Если запись не была вставлена на самом деле, то все поля в возвращаемой записи будут иметь значения NULL. Это поведение может быть изменено позже. В PSQL, если ни одна строка не вставлена, то ничего не возвращается и все целевые переменные сохраняют свои прежние значения.

6.2.6. Вставка столбцов BLOB

Вставка в столбцы BLOB только возможна при следующих обстоятельствах:

1. Клиентское приложение вставляет BLOB посредством Firebird API. В этом случае все зависит от приложения, и не рассматривается в этом руководстве;

2. Длина строкового литерала не может превышать 65,533 байт (64K - 3).

   Предел в символах вычисляется во время выполнения. Для мультибайтовых наборов символов он может отличаться. Например, для строки UTF8 (4 байта на символ) ограничение строкового литерала, вероятно, будет около (floor (65533/4)) = 16383 символов.

3. Если источником данных является столбец BLOB или выражение, возвращающее BLOB. Например, при использовании формы INSERT … SELECT или внутри PSQL кода, когда в качестве параметра подставляется переменная типа BLOB.

6.3. UPDATE

Назначение
Обновление данных в таблице.

Доступно в
DSQL, ESQL, PSQL

Синтаксис

```
UPDATE target [[AS] alias]
    SET col_name = <upd_value> [, col_name = <upd_value> ...]
    [WHERE {<search-conditions> | CURRENT OF cursorme}]
    [PLAN <plan_items>]
    [ORDER BY <sort_items>]
    [ROWS m [TO n]]
    [RETURNING <returning_list> [INTO <variables>]]
```

<upd_value> ::= <value_expression> | DEFAULT

<returning_list> ::= * | <output_column> [, <output_column]

<output_column> ::= 
    target.* | NEW.* | OLD.*
    | <return_expression> [COLLATE collation] [[AS] ret_alias]

<return_expression> ::= 
    <value_expression>
    | [target.]col_name
    | NEW.col_name
    | OLD.col_name

<value_expression> ::= 
    <literal>
    | <context-variable>
    | <other-single-value-expr>

<variables> ::= [:]varname [, [:]varname ...]

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Имя таблицы или представления, в которой происходит обновление записей.</td>
</tr>
<tr>
<td>alias</td>
<td>Псевдоним таблицы или представления.</td>
</tr>
<tr>
<td>col_name</td>
<td>Столбец таблицы или представления.</td>
</tr>
<tr>
<td>upd_value</td>
<td>Выражение для нового значения для столбца, который должен быть обновлен в таблице или представлении оператором.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context-variable</td>
<td>Контекстная переменная.</td>
</tr>
</tbody>
</table>
### Параметры

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>other-single-value-expr</td>
<td>Любое другое выражение, возвращающее единственноное значение типа данных Firebird или NULL.</td>
</tr>
<tr>
<td>search-conditions</td>
<td>Условие поиска, ограничивающее набор обновляемых строк.</td>
</tr>
<tr>
<td>cursorname</td>
<td>Имя курсора, по которому позиционируется обновляемая запись.</td>
</tr>
<tr>
<td>plan_items</td>
<td>Части плана запроса.</td>
</tr>
<tr>
<td>sort_items</td>
<td>Столбцы, перечисленные в предложении ORDER BY.</td>
</tr>
<tr>
<td>m, n</td>
<td>Целочисленные выражения для ограничения количества обновляемых строк.</td>
</tr>
<tr>
<td>return_expression</td>
<td>Выражение, возвращаемое в предложении RETURNING.</td>
</tr>
<tr>
<td>collation</td>
<td>Существующее имя сортировки (только для символьных типов).</td>
</tr>
<tr>
<td>ret_alias</td>
<td>Псевдоним для выражения, возвращаемого в предложении RETURNING.</td>
</tr>
<tr>
<td>varname</td>
<td>Имя PSQL переменной.</td>
</tr>
</tbody>
</table>

Оператор **UPDATE** изменяет значения столбцов в таблице, или в одной, или нескольких таблицах, лежащих в основе представления. Новые значения столбцов указываются в предложении **SET**. Затронутые строки могут быть ограничены предложениями **WHERE** и **ROWS**. Если нет ни **WHERE**, ни **ROWS**, все записи в таблице будут обновлены.

### 6.3.1. Использование псевдонима

Если вы назначаете псевдоним таблице или представлению, вы обязаны использовать псевдоним для уточнения столбцов таблицы.

#### Примеры

**Правильное использование:**

```sql
update Fruit set soort = 'pisang' where ...
update Fruit set Fruit.soort = 'pisang' where ...
update Fruit F set soort = 'pisang' where ...
update Fruit F set F.soort = 'pisang' where ...
```

**Неправильное использование:**

```sql
update Fruit F set Fruit.soort = 'pisang' where ...
```
6.3.2. SET

Изменяемые столбцы указываются в предложении SET. Столбцы и их значения перечисляются через запятую. Слева имя столбца, и справа значение или выражение.

Разрешено использовать имена столбцов в выражениях справа. При этом использоваться будет всегда старое значение столбца, даже если присваивание этому столбцу уже произошло ранее в перечислении SET. Один столбец может быть использован только один раз в конструкции SET.

Example 239. Использование оператора UPDATE

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

После выполнения оператора

```sql
update tset set a = 5, b = a
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Обратите внимание, что старые значения (1 и 2) используются для обновления столбца b, даже после того как столбцу a были назначено новое значение (5).

Ключевое слово DEFAULT

В предложении SET вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

Example 240. Использование ключевого слова DEFAULT в операторе UPDATE

```sql
CREATE TABLE cars (  
    ID INTEGER NOT NULL,  
    BYEAR SMALLINT DEFAULT 1990 NOT NULL,  
    NAME VARCHAR(45),  
    CONSTRAINT pk_cars PRIMARY KEY (ID)
)```

375
6.3.3. WHERE

Предложение WHERE ограничивает набор обновляемых записей заданным условием, или — в PSQL — текущей строкой именованного курсора, если указано предложение WHERE CURRENT OF.

Примечание: Предложение WHERE CURRENT OF используется только в PSQL, т.к. в DSQL нет оператора DSQL для создания курсора.

Строковые литералы могут предваряться именем набора символов, для того чтобы Firebird понимал, как интерпретировать данные.

**Example 241. Использование предложения WHERE в операторе UPDATE**

```sql
UPDATE addresses
SET city = 'Saint Petersburg', citycode = 'PET'
WHERE city = 'Leningrad';

UPDATE employees
SET salary = 2.5 * salary
WHERE title = 'CEO';

-- обратите внимание на префикс '_'
UPDATE People
SET name = _ISO8859_1 'Hans-Jörg Schäfer'
WHERE id = 53662;

UPDATE employee e
SET salary = salary * 1.05
WHERE EXISTS(
    SELECT *
    FROM employee_project ep
    WHERE e.emp_no = ep.emp_no);
```

6.3.4. PLAN

Предложение PLAN позволяет вручную указать план для оптимизатора.
**Example 242. Использование предложения PLAN в операторе UPDATE**

```
UPDATE company c SET c.company_name =
  ( SELECT k.contact_name
     FROM contact k
     WHERE k.id = c.contact_id
     PLAN (K INDEX (CONTACT_ID)))
WHERE c.company_name IS NULL OR c.company_name = ''
PLAN (C NATURAL)
```  

### 6.3.5. ORDER BY и ROWS

Предложение ORDER BY позволяет задать порядок обновления записей. Это может иметь значение в некоторых случаях.

Предложение ROWS имеет смысл только вместе с предложением ORDER BY. Однако его можно использовать отдельно.

При одном аргументе $m$, ROWS ограничивает update первыми $m$ записями.

Особенности:

- Если $m$ больше количества обрабатываемых записей в целевой таблице, то обновляется весь набор строк;
- Если $m = 0$, ни одна запись не обновляется;
- Если $m < 0$, выдаётся ошибка.

При двух аргументах $m$ и $n$, ROWS ограничивает update записей от $m$ до $n$ включительно. Оба аргумента – целочисленные, и начинаются с 1.

Особенности:

- Если $m$ больше количества записей в целевой таблице, ни одна запись не обновляется;
- Если $n$ больше количества записей в целевой таблице, то обновляются записи от $m$ до конца набора;
- Если $m < 1$ или $n < 1$, выдаётся ошибка;
- Если $n = m - 1$, ни одна запись не обновляется;
- Если $n < m - 1$, выдаётся ошибка.

**Example 243. Использование предложения ROWS в операторе UPDATE**

```
-- дать надбавку 20ти сотрудникам с наименьшей зарплатой
UPDATE employees
SET salary = salary + 50
ORDER BY salary ASC
```
6.3.6. RETURNING

Оператор UPDATE, обновляющий максимум одну строку, может включать RETURNING для возврата значений из обновляемой записи. В RETURNING могут включаться любые столбцы, необязательно только те, которые обновляются.

Возвращаемые значения содержат изменения, произведённые в триггерах BEFORE UPDATE, но не в триггерах AFTER UPDATE. Выражения OLD.fieldname и NEW.fieldname могут быть использованы в качестве имен столбцов. Если OLD. или NEW. не указано, возвращаются новые значения столбцов NEW.

Вместо списка столбцов вы можете указать звёздочку (*). В этом случае будут возвращены все значения столбцов таблицы. Звёздочку можно применять со спецификаторами NEW или OLD.

В DSQL оператор с RETURNING всегда возвращает только одну запись. Если записи не были обновлены оператором, то возвращаемые значения содержат NULL. Это поведение может быть изменено в последующих версиях Firebird.

INTO

Предложение INTO предназначено для передачи значений в локальные переменные. Оно доступно только в PSQL. Если записи не было обновлено, ничего не возвращается, и переменные, указанные в RETURNING, сохраняют свои прежние значения.

Example 244. Использование предложения RETURNING в операторе UPDATE

```
UPDATE Scholars
SET first_name = 'Hugh', last_name = 'Pickering'
WHERE first_name = 'Henry' AND last_name = 'Higgins'
RETURNING id, old.last_name, new.last_name;
```

Example 245. Использование * в предложении RETURNING в операторе UPDATE

```
UPDATE Scholars
SET first_name = 'Hugh', last_name = 'Pickering'
WHERE first_name = 'Henry' AND last_name = 'Higgins'
RETURNING old.*/;
```
6.3.7. Обновление столбцов BLOB

Обновление столбцов BLOB всегда полностью меняет их содержимое. Даже идентификатор BLOB (ID), который является ссылкой на данные BLOB и хранится в столбце, меняется. Столбцы типа BLOB могут быть изменены, если:

1. Клиентское приложение меняет BLOB посредством Firebird API. В этом случае все зависит от приложения, и не рассматривается в этом руководстве;
2. Длина строкового литерала не может превышать 65,533 байт (64K - 3).

Предел в символах вычисляется во время выполнения. Для мультибайтовых наборов символов он может отличаться. Например, для строки UTF8 (4 байта на символ) ограничение строкового литерала, вероятно, будет около (floor (65533/4)) = 16383 символов.

3. Если источником данных является столбец типа BLOB или выражение, возвращающее BLOB.

6.4. UPDATE OR INSERT

Назначение
Добавление новой или обновление существующей записи в таблице.

Доступно в
DSQL, PSQL

Синтаксис

```
UPDATE OR INSERT INTO target [(<column_list>)]
VALUES (<value_list>)
[MATCHING (<column_list>)]
[RETURNING <returning_list> [INTO <variables>]]
```

```
<column_list> ::= col_name [, col_name ...]

<value_list> ::= <ins_value> [, <ins_value> ...]

<ins_value> ::= <value_expression> | DEFAULT

<returning_list> ::= * | <output_column> [, <output_column>]

<output_column> ::= target.* | NEW.* | OLD.*
| <return_expression> [COLLATE collation] [[AS] alias]

<return_expression> ::=<value_expression>
| [target.]col_name
| NEW.col_name
```
Table 89. Параметры оператора UPDATE OR INSERT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Имя таблицы или представления, запись в которой будет обновлена или произойдет вставка новой записи.</td>
</tr>
<tr>
<td>col_name</td>
<td>Столбец таблицы или представления.</td>
</tr>
<tr>
<td>ins_value</td>
<td>Выражение, значение которого используется для вставки или обновления таблицы.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context-variable</td>
<td>Контекстная переменная.</td>
</tr>
<tr>
<td>other-single-value-expr</td>
<td>Любое другое выражение, возвращающее единственное значение типа данных Firebird или NULL.</td>
</tr>
<tr>
<td>return_expression</td>
<td>Выражение, возвращаемое в предложении RETURNING.</td>
</tr>
<tr>
<td>alias</td>
<td>Псевдоним для выражения, возвращаемого в предложении RETURNING.</td>
</tr>
<tr>
<td>varname</td>
<td>Имя PSQL переменной.</td>
</tr>
</tbody>
</table>

Оператор UPDATE OR INSERT вставляет или обновляет одну, или более существующих записей. Производимое действие зависит от значений столбцов в предложении MATCHING (или, если оно не указано, то от значений столбцов первичного ключа — PK). Если найдены записи, совпадающие с указанными значениями, то они обновляются. Если нет, то вставляется новая запись.

Совпадением считается полное совпадение значений столбцов MATCHING или PK. Совпадение проверяется с использованием IS NOT DISTINCT, поэтому NULL совпадает с NULL.

Ограничения

- Если у таблицы нет первичного ключа, то указание MATCHING считается обязательным;
- В списке MATCHING, так же как и в списке столбцов update/insert, каждый столбец может быть упомянут только один раз;
- Предложение INTO доступно только в PSQL.
6.4.1. Ключевое слово DEFAULT

В списке VALUES вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

Ограничение

Столбец для которого вместо значения использовано ключевое слово DEFAULT не может быть использован в предложении MATCHING.

Example 246. Использование ключевого слова DEFAULT в операторе UPDATE OR INSERT

```sql
CREATE TABLE cars (  ID INTEGER GENERATED BY DEFAULT AS IDENTITY,  BYYEAR SMALLINT DEFAULT 1990 NOT NULL,  NAME VARCHAR(45),  CONSTRAINT pk_cars PRIMARY KEY (ID) );

-- в столбец BYYEAR попадёт значение 1990
UPDATE OR INSERT INTO cars (byyear, name) VALUES (DEFAULT, 'Ford Focus')
MATCHING (name);
```

6.4.2. RETURNING

Предложение RETURNING может содержать любые столбцы, указанные в операторе, или другие столбцы и выражения. Возвращаемые значения содержат все изменения, произведённые в триггерах BEFORE, но не в триггерах AFTER. Выражения OLD.fieldname и NEW.fieldname могут быть использованы в качестве возвращаемых значений. Для обычных имен столбцов возвращаются новые значения.

Вместо списка столбцов вы можете указать звёздочку (*). В этом случае будут возвращены все значения столбцов таблицы. Звёздочку можно применять со спецификаторами NEW или OLD.

Example 247. Использование предложения RETURNING в операторе UPDATE OR INSERT

```sql
UPDATE OR INSERT INTO Cows (Name, Number, Location)
VALUES ('Suzy Creamcheese', 3278823, 'Green Pastures')
MATCHING (Number)
RETURNING rec_id
INTO :id;
```
6.5. DELETE

Назначение
Удаление данных из таблицы.

Доступно в
DSQL, ESQL, PSQL

Синтаксис:

DELETE
FROM target [[AS] alias]
[WHERE {<search-conditions> | CURRENT OF cursorname}]
[PLAN <plan_items>]
[ORDER BY <sort_items>]
[ROWS m [TO n]]
[RETURNING <returning_list> [INTO <variables>]]

<returning_list> ::= * | <output_column> [, <output_column>]

<output_column> ::= target.*
| <return_expression> [COLLATE collation] [[AS] ret_alias]

<return_expression> ::=<value_expression>
| [target.]col_name

[value_expression] ::=<literal>
| <context-variable>
| <other-single-value-expr>

<variables> ::=[:]varname [, [:]varname ...]

Table 90. Параметры оператора DELETE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Имя таблицы или представления, из которой удаляются записи.</td>
</tr>
<tr>
<td>alias</td>
<td>Псевдоним таблицы или представления.</td>
</tr>
<tr>
<td>col-name</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
<tr>
<td>search-conditions</td>
<td>Условие поиска, ограничивающее набор удаляемых записей.</td>
</tr>
<tr>
<td>cursorname</td>
<td>Имя курсора, по которому позиционируется удаляемая запись.</td>
</tr>
<tr>
<td>plan_items</td>
<td>Предложение плана.</td>
</tr>
<tr>
<td>sort_items</td>
<td>Предложение сортировки.</td>
</tr>
</tbody>
</table>
**Параметр** | **Описание**
--- | ---
m, n | Целочисленные выражения для ограничения количества удаляемых записей.
return_expression | Выражение, возвращаемое в предложении RETURNING.
literal | Литерал.
context-variable | Контекстная переменная.
other-single-value-expr | Любое другое выражение, возвращающее единственное значение типа данных Firebird или NULL.
collation | Существующее имя сортировки (только для символьных типов).
ret_alias | Псевдоним для выражения, возвращаемого в предложении RETURNING.
varname | Имя PSQL переменной.

Оператор `DELETE` удаляет строки из таблицы или из одной и более таблиц представления.

Если для таблицы указан псевдоним, то он должен использоваться для всех столбцов таблицы.

### 6.5.1. WHERE

Условие в предложении `WHERE` ограничивает набор удаляемых записей. Удаляются только те записи, которые удовлетворяют условию поиска, или только текущей записи именованного курсора.

Удаление с помощью `WHERE CURRENT OF` называется позиционированным удалением (positioned delete), потому что удаляется запись в текущей позиции. Удаление при помощи “WHERE условие” называется поисковым удалением (searched delete), поскольку Firebird ищет записи, соответствующие условию.

В чистом DSQL выражение `WHERE CURRENT OF` не имеет смисла, т.к. в DSQL нет оператора для создания курсора.

**Example 248. Использование предложения WHERE в операторе DELETE**

```
DELETE FROM People
WHERE first_name <> 'Boris' AND last_name <> 'Johnson';
```

```
DELETE FROM employee e
WHERE NOT EXISTS(
    SELECT *
    FROM employee_project ep
    WHERE e.emp_no = ep.emp_no);
```
6.5.2. PLAN

Предложение **PLAN** позволяет вручную указать план для оптимизатора.

*Example 249. Использование предложения **PLAN** в операторе **DELETE***

```sql
DELETE FROM Submissions
WHERE date_entered < '1-Jan-2002'
PLAN (Submissions INDEX ix_subm_date)
```

6.5.3. ORDER BY и ROWS

Предложение **ORDER BY** упорядочивает набор перед его удалением. Что может быть важно в некоторых случаях.

Предложение **ROWS** позволяет ограничить количество удаляемых строк. Имеет смысл только в комбинации с предложением **ORDER BY**, но допустимо и без него.

В качестве *m* и *n* могут выступать любые целочисленные выражения.

При одном аргументе *m*, удаляются первые *m* записей. Порядок записей без **ORDER BY** не определён (случайный).

Замечания:

- Если *m* больше общего числа записей в наборе, то весь набор удаляется;
- Если *m* = 0, то удаление не происходит;
- Если *m* < 0, то выдаётся сообщение об ошибке.

Если указаны аргументы *m* и *n*, удаление ограничено количеством записей от *m* до *n*, включительно. Нумерация записей начинается с 1.

Замечания по использованию двух аргументов:

- Если *m* больше общего числа строк в наборе, ни одна строка не удаляется;
- Если *m* > 0 и ≤ числа строк в наборе, а *n* вне этих значений, то удаляются строки от *m* до конца набора;
- Если *m* < 1 или *n* < 1, выдаётся сообщение об ошибке;
- Если *n* = *m* – 1, ни одна запись не удаляется;
• Если \( n < m - 1 \), выдаётся сообщение об ошибке.

**Example 250. Использование ORDER BY и ROWS в операторе DELETE**

Удаление самой старой покупки

```sql
DELETE FROM Purchases ORDER BY ByDate ROWS 1
```

Удаление заказов для 10 клиентов с самыми большими номерами

```sql
DELETE FROM Sales ORDER BY custno DESC ROWS 1 TO 10
```

Удаляет все записи из sales, поскольку не указано ROWS

```sql
DELETE FROM Sales ORDER BY custno DESC
```

Удаляет одну запись "с конца", т.е. от Z...

```sql
DELETE FROM popgroups ORDER BY name DESC ROWS 1
```

Удаляет пять самых старых групп

```sql
DELETE FROM popgroups ORDER BY formed ROWS 5
```

Сортировка (ORDER BY) не указана, поэтому будут удалены 8 обнаруженных записей, начиная с пятой.

```sql
DELETE FROM popgroups ROWS 5 TO 12
```

### 6.5.4. RETURNING

Оператор DELETE, удаляющий не более одной записи, может содержать конструкцию RETURNING для возвращения значений удаляемой записи. В RETURNING могут быть указаны любые столбцы, необязательно все, а также другие столбцы и выражения. Вместо списка столбцов может быть указана звёздочка (*), в этом случае будут возвращены все столбцы удалённой записи.

- В DSQL, оператор с RETURNING всегда возвращает только одну запись. Если записи не были удалены, то возвращаемые столбцы содержат NULL. Это поведение может быть изменено в последующих версиях Firebird;
- В PSQL, если строка не была удалена, ничего не возвращается, и целевые
 переменные сохраняют свои значения;
• Предложение INTO доступно только в PSQL.

**Example 251. Использование предложения RETURNING в операторе DELETE**

```sql
DELETE FROM Scholars
WHERE first_name = 'Henry' AND last_name = 'Higgins'
RETURNING last_name, full_name, id

DELETE FROM Dumbbells
ORDER BY iq DESC
ROWS 1
RETURNING last_name, iq
INTO :lname, :iq;

DELETE FROM TempSales ts
WHERE ts.id = tempid
RETURNING ts.qty
INTO new.qty;
```

### 6.6. **MERGE**

**Назначение**
Слияние записей источника в целевую таблицу (или обновляемое представление).

**Доступно в**
DSQL, PSQL

**Синтаксис**

```sql
MERGE INTO target [[AS] target_alias]
USING <source> [[AS] source_alias]
ON <join condition>
<merge when> [(<merge when> ...)]
[RETURNING <returning_list> [INTO <variable_list>]]

<source> ::= tablename | (<select_stmt>)

<merge when> ::=  
<merge when matched>  
| <merge when not matched>

<merge when matched> ::=  
WHEN MATCHED [ AND <condition> ]  
THEN { UPDATE SET <assignment_list> | DELETE }

<merge when not matched> ::=  
```

---

386
Chapter 6. Операторы обработки данных (DML)

Table 91. Параметры оператора MERGE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Целевая таблица или обновляемое представление.</td>
</tr>
<tr>
<td>source</td>
<td>Источник данных. Может быть таблицей, представлением, хранимой процедурой или производной таблицей.</td>
</tr>
<tr>
<td>target_alias</td>
<td>Псевдоним целевой таблицы или представления.</td>
</tr>
<tr>
<td>source_alias</td>
<td>Псевдоним источника.</td>
</tr>
<tr>
<td>join condition</td>
<td>Условие соединения целевой таблицы и источника.</td>
</tr>
<tr>
<td>condition</td>
<td>Дополнительные условия проверки в предложениях WHEN MATCHED или WHEN NOT MATCHED.</td>
</tr>
<tr>
<td>col_name</td>
<td>Столбец целевой таблицы или представления.</td>
</tr>
</tbody>
</table>
Оператор `MERGE` производит слияние записей источника в целевую таблицу (или обновляемое представление). Источником может быть таблица, представление, хранимой процедурой или производной таблицей. Каждая запись источника используется для обновления (предложение `UPDATE`) или удаления (предложение `DELETE`) одной или более записей цели, или вставки (предложение `INSERT`) записи в целевую таблицу, или ни для того, ни для другого. Условие обычно содержит сравнение столбцов в таблицах источника и цели.

В списке `VALUES` предложения `INSERT` и списке `SET` предложения `UPDATE` вместо значения столбца можно использовать ключевое слово `DEFAULT`. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение `NULL`.

Допускается указывать несколько предложений `WHEN MATCHED` и `WHEN NOT MATCHED`.

Предложения `WHEN` проверяются в указанном порядке. Если условие в предложении `WHEN` не выполняется, то мы пропускаем его и переходим к следующему предложению. Так будет происходить до тех пор, пока условие для одного из предложений `WHEN` не будет выполнено. В этом случае выполняется действие, связанное с предложением `WHEN`, и осуществляется переход на следующую строку источника. Для каждой записи источника выполняется только одно действие.

WHEN NOT MATCHED оценивается с точки зрения источника, т.е. таблицы или набора данных указанного в предложении `USING`. Так сделано потому, что если запись источника не имеет совпадения с записью цели, то выполняется `INSERT`. Разумеется, если запись цели не соответствует записи в источнике, то никакие действия не производятся.

На данный момент переменная `ROW_COUNT` возвращает значение 1, даже если было модифицировано или вставлено более 1 записи. См. [CORE-4400](#).

Если условие `WHEN MATCHED` присутствует, и несколько записей совпадают с записями в целевой таблице, то будет выдана ошибка.

До Firebird 4.0 `UPDATE` выполняется для всех совпадающих записей источника, и каждое последующее обновление перезапишет предыдущее. Это поведение не соответствует SQL стандарту.
**Example 252. Простые операторы MERGE**

```sql
MERGE INTO books b
USING purchases p
ON p.title = b.title AND p.booktype = 'bk'
WHEN MATCHED THEN
    UPDATE SET b.descr = b.descr || ' ' || p.descr
WHEN NOT MATCHED THEN
    INSERT (title, descr, bought)
    VALUES (p.title, p.descr, p.bought);

-- с использованием производной таблицы
MERGE INTO customers c
USING (SELECT * FROM customers_delta WHERE id > 10) cd
ON (c.id = cd.id)
WHEN MATCHED THEN
    UPDATE SET name = cd.name
WHEN NOT MATCHED THEN
    INSERT (id, name) VALUES (cd.id, cd.name);

-- совместно с рекурсивным CTE
MERGE INTO numbers
USING (WITH RECURSIVE r(n) AS (SELECT 1 FROM rdb$database
    UNION ALL
    SELECT n+1 FROM r WHERE n < 200)
    SELECT n FROM r)
) t
ON numbers.num = t.n
WHEN NOT MATCHED THEN
    INSERT(num) VALUES(t.n);

-- с использованием предложения DELETE
MERGE INTO SALARY_HISTORY
USING (SELECT EMP_NO
    FROM EMPLOYEE
    WHERE DEPT_NO = 120) EMP
ON SALARY_HISTORY.EMP_NO = EMP.EMP_NO
WHEN MATCHED THEN DELETE
```

**Example 253. Использование оператора MERGE с дополнительными условиями в предложениях WHEN [NOT] MATCHED**

В следующем примере происходит ежедневное обновление таблицы PRODUCT_INVENTORY на основе заказов, обработанных в таблице SALES_ORDER_LINE. Если количество заказов на
продукт таково, что уровень запасов продукта опускается до нуля или становится ещё ниже, то строка этого продукта удаляется из таблицы PRODUCT_INVENTORY.

MERGE INTO PRODUCT_INVENTORY AS TARGET
USING (SELECT SL.ID_PRODUCT, SUM(SL.QUANTITY) FROM SALES_ORDER_LINE SL JOIN SALES_ORDER S ON S.ID = SL.ID_SALES_ORDER WHERE S.BYDATE = CURRENT_DATE GROUP BY 1) AS SRC(ID_PRODUCT, QUANTITY)
ON TARGET.ID_PRODUCT = SRC.ID_PRODUCT
WHEN MATCHED AND TARGET.QUANTITY - SRC.QUANTITY <= 0 THEN DELETE
WHEN MATCHED THEN
UPDATE SET TARGET.QUANTITY = TARGET.QUANTITY - SRC.QUANTITY,
TARGET.BYDATE = CURRENT_DATE

См. также:
SELECT, INSERT, UPDATE, DELETE.

6.6.1. RETURNING

Оператор MERGE, затрагивающий не более одной строки, может содержать конструкцию RETURNING для возвращения значений добавленной, модифицируемой или удаляемой строки. В RETURNING могут быть указаны любые столбцы из целевой таблицы (обновляемого представления), необязательно все, а также другие столбцы и выражения.

Возвращаемые значения содержат изменения, произведённые в триггерах BEFORE.

Имена столбцов могут быть уточнены с помощью префиксов NEW и OLD для определения, какое именно значение столбца вы хотите получить до модификации или после.

Вместо списка столбцов может быть указана звёздочка (*), в этом случае будут возвращены все столбцы целевой таблицы. Префиксы NEW и OLD могут быть использованы совместно со звёздочкой.

Для предложений WHEN MATCHED UPDATE и MERGE WHEN NOT MATCHED неуточнённые имена столбцов или уточнённые именами таблиц или их псевдонимами понимаются как столбцы с префиксом NEW, для предложений MERGE WHEN MATCHED DELETE — с префиксом OLD.

Example 254. Использование оператора MERGE с предложением RETURNING

Немного модифицируем наш предыдущий пример, чтобы он затрагивал только одну строку, и добавим в него инструкцию RETURNING возвращающего старое и новое
количество товара и разницу между этими значениями.

MERGE INTO PRODUCT_INVENTORY AS TARGET
USING ( 
SELECT
     SL.ID_PRODUCT, 
     SUM(SL.QUANTITY)
FROM SALES_ORDER_LINE SL 
JOIN SALES_ORDER S ON S.ID = SL.ID SALES_ORDER
WHERE S.BYDATE = CURRENT_DATE 
     AND SL.ID_PRODUCT = :ID_PRODUCT
GROUP BY 1
) AS SRC(ID_PRODUCT, QUANTITY)
ON TARGET.ID_PRODUCT = SRC.ID_PRODUCT
WHEN MATCHED AND TARGET.QUANTITY - SRC.QUANTITY <= 0 THEN
DELETE
WHEN MATCHED THEN
UPDATE SET
     TARGET.QUANTITY = TARGET.QUANTITY - SRC.QUANTITY,
     TARGET.BYDATE = CURRENT_DATE
RETURNING OLD.QUANTITY, NEW.QUANTITY, SRC.QUANTITY
INTO :OLD_QUANTITY, :NEW_QUANTITY, :DIFF_QUANTITY

6.7. EXECUTE PROCEDURE

Назначение
Выполнение хранимой процедуры.

Доступно в
DSQL, ESQL, PSQL

Синтаксис

EXECUTE PROCEDURE procname
    [({ <inparam-list> | ( <inparam-list> ) })]
    [RETURNING_VALUES { <outvar-list> | ( <outvar-list> ) }]

<inparam-list> ::=  
    <inparam> [, <inparam> ...]

<outvar-list> ::=  
    <outvar> [, <outvar> ...]

<outvar> ::= [:]varname

Table 92. Параметры оператора EXECUTE PROCEDURE
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>procname</td>
<td>Имя хранимой процедуры.</td>
</tr>
<tr>
<td>inparam</td>
<td>Выражение совместимое по типу с входным параметром хранимой процедуры.</td>
</tr>
<tr>
<td>varname</td>
<td>PSQL переменная, в которую возвращается значение выходного параметра процедуры.</td>
</tr>
</tbody>
</table>

Оператор EXECUTE PROCEDURE выполняет хранимую процедуру, получая список из одного или нескольких входных параметров, если они определены, и возвращает односторочный набор значений, если он определён.

### 6.7.1. "Выполняемые" хранимые процедуры

Оператор EXECUTE PROCEDURE является наиболее часто используемым стилем вызова хранимой процедуры, которая написана для модификации некоторых данных. Их код не содержит оператора SUSPEND. Такие хранимые процедуры могут возвратить набор данных, состоящий не более чем из одной строки. Этот набор может быть передан в переменную другой (вызывающей) процедуры с помощью предложения RETURNING_VALUES. Клиентские интерфейсы, как правило, имеют обертку API, которые могут извлекать выходные значения в односторочный буфер при вызове процедуры через EXECUTE PROCEDURE в DSQL.

При вызове с помощью EXECUTE PROCEDURE процедур другого типа (селективных процедур) будет возвращена только первая запись из результирующего набора, несмотря на то, что эта процедура скорее всего должна возвращать многострочный результат. "Селективные" хранимые процедуры должны вызываться с помощью оператора SELECT, в этом случае они ведут себя как виртуальные таблицы.

- В PSQL и DSQL входными параметрами могут быть любые совместимые по типу выражения;
- Несмотря на то, что скобки для отделения списка передаваемых параметров необязательны после имени хранимой процедуры, желательно их использовать;
- Предложение RETURNING_VALUES доступно только в PSQL.

### Example 255. Использование оператора EXECUTE PROCEDURE в PSQL

```sql
EXECUTE PROCEDURE MakeFullName(:First_Name, :Middle_Name, :Last_Name)
RETURNING_VALUES :FullName;
```

В этом операторе использование двоеточия ("::<wbr/>") для входных и выходных параметров необязательно.

Разрешено использовать выражения в качестве параметров.

```sql
EXECUTE PROCEDURE MakeFullName
```
('Mr./Mrs. ' || First_Name, Middle_Name, upper(Last_Name))
RETURNING_VALUES FullName;

Example 256. Вызов оператора EXECUTE PROCEDURE в isql

EXECUTE PROCEDURE MakeFullName
  'J', 'Edgar', 'Hoover';

6.8. EXECUTE BLOCK

Назначение
Выполнение анонимного PSQL блока.

Доступно в
DSQL

Синтаксис

EXECUTE BLOCK [((<inparams>))]  
[RETURNS (<outparams>)]  
<psql-routine-body>

<inparams> ::= <param_decl> = ? [, <inparams> ]

<outparams> ::= <param_decl> [, <outparams>]

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<type> ::=  
  <non_array_datatype>
  | [TYPE OF] domain
  | TYPE OF COLUMN rel.col

<non_array_datatype> ::=  
  <scalar_datatype> | <blob_datatype>

<scalar_datatype> ::=  
  См. Синтаксис скалярных типов данных

<blob_datatype> ::=  
  См. Синтаксис типа данных BLOB

<psql-routine-body> ::=  
  См. Синтаксис тела модуля

Table 93. Параметры оператора EXECUTE BLOCK
Имя входного или выходного параметра процедуры. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры, а также её локальных переменных.

Тип данных SQL за исключением массивов.

Порядок сортировки.

Домен.

Имя таблицы или представления.

Имя столбца таблицы или представления.

Выполняет блок PSQL кода, так как будто это хранимая процедура, возможно с входными и выходными параметрами и локальными переменными. Это позволяет пользователю выполнять “на лету” PSQL в контексте DSQL.

Примеры:

Этот пример вводит цифры от 0 до 127 и соответствующие им ASCII символы в таблицу ASCITABLE:

```sql
EXECUTE BLOCK
AS
  DECLARE i INT = 0;
BEGIN
  WHILE (i < 128) DO
    BEGIN
      INSERT INTO AsciiTable VALUES (:i, ascii_char(:i));
      i = i + 1;
    END
  END
END
```

Следующий пример вычисляет среднее геометрическое двух чисел и возвращает его пользователю:

```sql
EXECUTE BLOCK (x DOUBLE PRECISION = ?, y DOUBLE PRECISION = ?)
RETURNS (gmean DOUBLE PRECISION)
AS
BEGIN
  gmean = sqrt(x*y);
  SUSPEND;
END
```
Поскольку этот блок имеет входные параметры, он должен быть предварительно подготовлен. После чего можно установить параметры и выполнить блок. Как это будет сделано, и вообще возможно ли это сделать, зависит от клиентского программного обеспечения. Смотрите примечания ниже.

Наш последний пример принимает два целочисленных значений, smallest и largest. Для всех чисел в диапазоне smallest..largest, блок выводит само число, его квадрат, куб и четвертую степень.

```sql
EXECUTE BLOCK (smallest INT = ?, largest INT = ?)
RETURNS (number INT, square BIGINT, cube BIGINT, fourth BIGINT)
AS BEGIN
  number = smallest;
  WHILE (number <= largest) DO BEGIN
    square = number * number;
    cube = number * square;
    fourth = number * cube;
    SUSPEND;
    number = number + 1;
  END
END
```

Опять же, как вы можете установить значения параметров, зависит от программного обеспечения клиента.

См. также:
Операторы языка PSQL.

### 6.8.1. Входные и выходные параметры

Выполнение блока без входных параметров должно быть возможным с любым клиентом Firebird, который позволяет пользователю вводить свои собственные DSQL операторы. Если есть входные параметры, все становится сложнее: эти параметры должны получить свои значения после подготовки оператора, но перед его выполнением. Это требует специальных возможностей, которыми располагает не каждое клиентское приложение (Например, isql такой возможности не предлагает).

Сервер принимает только вопросительные знаки ("?") в качестве заполнителей для входных значений, а не ":а", "MyParam" и т.д., или литературные значения. Клиентское программное обеспечение может поддерживать форму ":xxx", в этом случае будет произведена предварительная обработка запроса перед отправкой его на сервер.

Если блок имеет выходные параметры, вы должны использовать SUSPEND, иначе ничего не
будет возвращено.

Выходные данные всегда возвращаются в виде набора данных, так же как и в случае с оператором SELECT. Вы не можете использовать RETURNING VALUES или выполнить блок, вернув значения в некоторые переменные, используя INTO, даже если возвращается всего одна строка.

Для получения дополнительной информации о параметрах и объявлениях переменных, [TYPE OF domain, TYPE OF COLUMN] и т.д. обратитесь к главе DECLARE VARIABLE.

6.8.2. Терминатор оператора

Некоторые редакторы SQL-операторов — в частности утилита isql, которая идёт в комплекте с Firebird, и возможно некоторые сторонние редакторы — используют внутреннее соглашение, которое требует, чтобы все операторы были завершены с точкой с запятой.

Это создает конфликт с синтаксисом PSQL при кодировании в этих средах. Если вы не знакомы с этой проблемой и ее решением, пожалуйста, изучать детали в главе PSQL в разделе, озаглавленном Изменение терминатора в isql.
Chapter 7. Операторы процедурного SQL (PSQL)

Procedural SQL (PSQL)—процедурное расширение языка SQL. Это подмножество языка используется для написания хранимых процедур, хранимых функций, пакетов, триггеров и SQL блоков.

Это расширение содержит все основные конструкции классических языков программирования. Кроме того, в него входят немного модифицированные DML операторы (SELECT, INSERT, UPDATE, DELETE и др.).

7.1. Элементы PSQL

Процедурное расширение может содержать объявления локальных переменных и курсоров, операторы присваивания, условные операторы, операторы циклов, выброса пользовательского исключений, средства для обработки ошибок, отправки сообщений (событий) клиентским программам. Кроме того, в триггерах доступны специфичные контекстные переменные, такие как NEW и OLD.

В PSQL не допустимы операторы модификации метаданных (DDL операторы).

7.1.1. DML операторы с параметрами

В DML (SELECT, INSERT, UPDATE, DELETE и др.) операторах допустимы только именованные параметры. Если DML операторы содержат именованные параметры, то они должны быть предварительно объявлены как локальные переменные в операторе DECLARE [VARIABLE] заголовка модуля или доступны во входных или выходных параметрах PSQL модуля.

При использовании именованных параметров в DML операторах необходим префикс двоеточия “:”, однако в предложении INTO символ двоеточия не обязателен. Префикс двоеточия является необязательным в операторах специфичных для PSQL, таких, как операторы ветвления или присваивания. Префикс двоеточия не требуется также при вызове хранимой процедуры с помощью оператора EXECUTE PROCEDURE из другого PSQL модуля.

7.1.2. Транзакции

Хранимые процедуры и функции (в том числе содержащиеся в пакетах) выполняются в контексте той транзакции, в которой они были запущены. Триггеры выполняются в контексте транзакции, в которой выполнялся DML оператор, вызвавший запуск триггера. Для триггеров на событие базы данных запускается отдельная транзакция.

В PSQL не допустимы операторы старта и завершения транзакций, но существует возможность запуска оператора или блока операторов в автономной транзакции.
7.1.3. Структура модуля

В синтаксисе PSQL модулей можно выделить заголовок и тело. DDL операторы для их объявления являются сложными операторами, т.е. состоят из единственного оператора, который включает в себя блоки нескольких операторов. Такие операторы начинаются с глагола (CREATE, ALTER, DROP, RECREATE, CREATE OR ALTER) и завершаются последним оператором END тела модуля.

Заголовок модуля

Заголовок содержит имя модуля и описание локальных переменных. Для хранимых процедур и PSQL блоков заголовок может содержать описание входных и выходных параметров. Заголовок триггеров не может содержать входных и выходных параметров.

В заголовке триггера обязательно указывается событие (или комбинация событий), при котором тригер будет вызван автоматически.

Привилегии выполнения PSQL кода

PSQL код может выполняться в одном из следующих режимов:

- С привилегиями вызывающего пользователя (привилегии CURRENT_USER);
- С привилегиями определяющего пользователя (владельца объекта метаданных).

Привилегии выполнения PSQL модуля указываются в его заголовке в необязательное предложение SQL SECURITY. Если выбрана опция INVOKER, то PSQL модуль выполняются с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то PSQL модуль выполняется с привилегиями определяющего пользователя (владельца). Эти привилегии будут дополнены привилегиями выданные самому PSQL модулю с помощью оператора GRANT. По умолчанию процедуры, функции выполняются с привилегиями вызывающего пользователя, а триггеры наследуют привилегии безопасности указанные для таблицы.

Анонимные PSQL блоки (EXECUTE BLOCK) всегда выполняются с правами вызывающего пользователя.

Тело модуля

Тело модуля может быть написано на языке SQL или быть телом внешнего модуля.

Синтаксис тела модуля

\[
<	ext{routine-body}>::=
\begin{align*}
&<\text{psql-routine-spec}> \\
| &<\text{external-routine-spec}>
\end{align*}
\]

\[
<\text{psql-routine-spec}>::= \\
[<\text{rights-clause}>]<\text{psql-routine-body}>
\]

\[
<\text{rights-clause}>::= 
\]

398
SQL SECURITY {DEFINER | INVOKER}

<psql-routine-body> ::= 
  AS 
  [<declarations>]
  BEGIN 
  [<PSQL_statements>]
  END

<declarations> ::= 
  <declare-item> [<declare-item> ...]

<declare-item> ::= 
  <declare-var>; 
  | <declare-cursor>; 
  | <subroutine-declaration>; 
  | <subroutine-implemention>

$subroutine-declaration$ ::= <subfunc-decl> | <subproc-decl>

$subroutine-implemention$ ::= <subfunc-impl> | <subproc-impl>

$external-routine-spec$ ::= 
  <external-routine-reference>
  [AS <extbody>]

<external-routine-reference> ::= EXTERNAL NAME <extname> ENGINE <engine>

<extname> ::= 
  '<module-name>!<routine-name>[!<misc-info>]'

Table 94. Параметры тела модуля

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare-var</td>
<td>Объявление локальной переменной.</td>
</tr>
<tr>
<td>declare-cursor</td>
<td>Объявление именованного курсора.</td>
</tr>
<tr>
<td>subfunc-decl</td>
<td>Объявление подпрограммы – функции.</td>
</tr>
<tr>
<td>subproc-decl</td>
<td>Объявление подпрограммы – процедуры.</td>
</tr>
<tr>
<td>subfunc-impl</td>
<td>Реализация подпрограммы – функции.</td>
</tr>
<tr>
<td>subproc-impl</td>
<td>Реализация подпрограммы – процедуры.</td>
</tr>
<tr>
<td>extbody</td>
<td>Тело внешней процедуры. Строковый литерал который может использоваться UDR для различных целей.</td>
</tr>
<tr>
<td>module-name</td>
<td>Имя внешнего модуля, в котором расположена функция.</td>
</tr>
<tr>
<td>routine-name</td>
<td>Внутреннее имя функции внутри внешнего модуля.</td>
</tr>
</tbody>
</table>
Параметр | Описание
---|---
misc-info | Определяемая пользователем информация для передачи в функцию внешнего модуля.
engine | Имя движка для использования внешних функций. Обычно указывается имя UDR.

**Тело PSQL модуля**

Тело PSQL начинается с необязательного раздела, в котором объявляются переменные, курсоры и подпрограммы. Далее следует блок операторов, которые выполняются в логической последовательности как программа. Блок операторов — или составной оператор — заключен в ключевые слова BEGIN и END и выполняется как единый блок кода. Основной блок BEGIN … END может содержать любое количество других блоков BEGIN … END, как встроенных, так и последовательных. Максимальная вложенность блоков составляет 512 уровней. Все операторы, кроме BEGIN и END, заканчиваются точкой с запятой (";"). Никакой другой символ не может использоваться в качестве терминатора для операторов PSQL.

**Изменение терминатора в isql**

Здесь мы немного отвлечёмся для того, чтобы объяснить как переключить терминатор в утилите isql. Это необходимо, чтобы иметь возможность определять в ней PSQL модули, не конфликтуя с самим isql, который использует тот же самый символ, точку с запятой (,), как разделитель операторов.

**isql команда  SET TERM**

**Назначение**
Изменение символа(ов) терминатора, чтобы избежать конфликта с терминатором в PSQL операторах.

**Доступно в**
ISQL.

**Синтаксис**

```
SET TERM new_terminator old_terminator
```

**Table 95. Параметры оператора SET TERM**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_terminator</td>
<td>Новый терминатор.</td>
</tr>
<tr>
<td>old_terminator</td>
<td>Старый терминатор.</td>
</tr>
</tbody>
</table>

При написании триггеров и хранимых процедур в текстах скриптов, создающих требуемые программные объекты базы данных, во избежание двусмысленности относительно использования символа завершения операторов (по нормам SQL это...
точка с запятой) применяется оператор SET TERM, который, строго говоря, не является оператором SQL, а является командой интерактивного инструмента `isql`. При помощи этого оператора перед началом создания триггера или хранимой процедуры задаётся символ или строка символов, являющийся завершающим в конце текста триггера или хранимой процедуры. После описания текста соответствующего программного объекта при помощи того же оператора SET TERM значение терминатора возвращается к обычному варианту — точка с запятой.

Альтернативный терминатор может быть любой произвольной строкой символов за исключением точки с запятой, пробела и апострофа. Если вы используете буквенный символ, то он будет чувствителен к регистру.

*Example 257. Задание альтернативного терминатора*

```
SET TERM ^;

CREATE OR ALTER PROCEDURE SHIP_ORDER (PO_NUM CHAR(8))
AS
BEGIN
  /* Тело хранимой процедуры */
END^;

/* Другие хранимые процедуры и триггеры */

SET TERM ;^;

/* Другие операторы DDL */
```

**Тело внешнего модуля**

Тело внешнего модуля определяет механизм UDR, используемый для выполнения внешнего модуля, и дополнительно указывает имя вызываемой процедуры UDR (<extname>) и/или строку (<extbody>) с семантикой, специфичной для UDR.

Конфигурация внешних модулей и механизмов UDR не рассматривается далее в этом справочнике по языку. За подробностями обращайтесь к документации по конкретному движку UDR.

**7.2. Хранимые процедуры**

Хранимая процедура является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой процедуре могут обращаться хранимые процедуры (в том числе и сама к себе), триггеры и клиентские программы. Если хранимая процедура вызывает саму себя, то такая хранимая процедура называется рекурсивной.
7.2.1. Преимущества хранимых процедур

Хранимые процедуры имеют следующие преимущества:

<table>
<thead>
<tr>
<th>Модульность</th>
<th>Приложения, работающие с одной и той же базой данных, могут использовать одну и ту же хранимую процедуру, тем самым уменьшив размер кода приложения и устранив дублирование кода.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Упрощение поддержки приложений</td>
<td>При изменении хранимой процедуры, изменения отражаются сразу во всех приложениях, без необходимости их перекомпиляции.</td>
</tr>
<tr>
<td>Увеличение производительности</td>
<td>Поскольку хранимые процедуры выполняются на стороне сервера, а не клиента, то это уменьшает сетевой трафик, что повышает производительность.</td>
</tr>
</tbody>
</table>

7.2.2. Типы хранимых процедур

Существуют два вида хранимых процедур — выполняемые хранимые процедуры (executable stored procedures) и селективные процедуры (selectable stored procedures).

Выполняемые хранимые процедуры

Выполняемые хранимые процедуры, осуществляют обработку данных, находящихся в базе данных. Эти процедуры могут получать входные параметры и возвращать одиночный набор выходных (RETURNS) параметров. Такие процедуры выполняются с помощью оператора EXECUTE PROCEDURE. См. пример создания выполняемой хранимой процедуры в конце раздела CREATE PROCEDURE главы “Операторы DDL”.

Селективные хранимые процедуры

Селективные хранимые процедуры обычно осуществляют выборку данных из базы данных и возвращают при этом произвольное количество строк.

Такие процедуры позволяют получать довольно сложные наборы данных, которые зачастую невозможно или весьма затруднительно получить с помощью обычных DSQL SELECT запросов. Обычно такие процедуры выполняют циклический процесс извлечения данных, возможно преобразуя их, прежде чем заполнить выходные переменные (параметры) новыми данными на каждой итерации цикла. Оператор SUSPEND, обычно расположенный в конце каждой итерации, заполняет буфер и ожидает пока вызывающая сторона не выберет (fetch) строку.

Селективные процедуры могут иметь входные параметры и выходное множество, заданное в предложении RETURNS заголовка процедуры.

Обращение к селективной хранимой процедуре осуществляется при помощи оператора
7.2.3. Создание хранимой процедуры

Синтаксис создания выполняемых хранимых процедур и селективных процедур ничем не отличается. Разница заключается в логике программного кода.

Для получения информации о создание хранимых процедур см. CREATE PROCEDURE в главе “Операторы определения данных DDL”.

7.2.4. Изменение хранимой процедуры

В существующих хранимых процедурах можно изменять набор входных и выходных параметров и тело процедуры.

Для получения информации об изменении существующих хранимых процедур см. ALTER PROCEDURE, CREATE OR ALTER PROCEDURE, RECREATE PROCEDURE в главе “Операторы определения данных DDL”.

7.2.5. Удаление хранимой процедуры

Для получения информации об удалении хранимых процедур см. DROP PROCEDURE в главе “Операторы определения данных DDL”.

7.3. Хранимые функции

Хранимая функция является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой функции самой к себе такая хранимая функция называется рекурсивной.

В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Для возврата значения из хранимой функции используется оператор RETURN, который немедленно прекращает выполнение функции.

7.3.1. Создание хранимой функции

Для получения информации о создание хранимых функций см. CREATE FUNCTION в главе “Операторы определения данных DDL”.

7.3.2. Изменение хранимой функции

Для получения информации об изменении существующих хранимых функций см. ALTER FUNCTION, CREATE OR ALTER FUNCTION, RECREATE FUNCTION в главе “Операторы определения данных DDL”.
7.3.3. Удаление хранимой функции

Для получения информации об удалении хранимых функций см. DROP FUNCTION в главе “Операторы определения данных DDL”.

7.4. PSQL блоки

Для выполнения из декларативного SQL (DSQL) некоторых императивных действий используются анонимные (безымянные) PSQL блоки. Заголовок анонимного PSQL блока опционально может содержать входные и выходные параметры. Тело анонимного PSQL блока может содержать объявление локальных переменных, курсоров, подпрограмм и блок PSQL операторов.

Анонимный PSQL блок не определяется и сохраняется как объект метаданных, в отличие от хранимых процедур и триггеров. Он не может обращаться сам к себе.

Как и хранимые процедуры анонимные PSQL блоки могут использоваться для обработки данных или для осуществления выборки из базы данных.

Синтаксис (полный):

```
EXECUTE BLOCK
  [([<inparam> = ? [, <inparam> = ? ...]])
  [RETURNS (<outparam> [, <outparam> ...]])
  <psql-routine-body>
```

Тело анонимного PSQL блока не является объектом, он не может обращаться сам к себе.

См. Синтаксис тела модуля

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
<tr>
<td>outparam</td>
<td>Описание выходного параметра.</td>
</tr>
</tbody>
</table>

См. также:
EXECUTE BLOCK.

7.5. Пакеты

Пакет — группа процедур и функций, которая представляет собой единый объект базы данных.

Пакеты Firebird состоят из двух частей: заголовка (ключевое слово PACKAGE) и тела (ключевые слова PACKAGE BODY). Такое разделение очень сильно напоминает модули Delphi, заголовок соответствует интерфейсной части, а тело — части реализации.
7.5.1. Преимущества пакетов

Пакеты обладают следующими преимуществами:

Модульность

Блоки взаимозависимого кода выделены в логические модули, как это сделано в других языках программирования.

В программировании существует множество способов для группировки кода, например с помощью пространств имен (namespaces), модулей (units) и классов. Со стандартными процедурами и функциями базы данных это не возможно.

Упрощение отслеживания зависимостей

Пакеты упрощают механизм отслеживания зависимостей между набором связанных процедур, а также между этим набором и другими процедурами, как упакованными, так и неупакованными.

Каждый раз, когда упакованная подпрограмма определяет, что используется некоторый объект базы данных, информации о зависимости от этого объекта регистрируется в системных таблицах Firebird. После этого, для того чтобы удалить или изменить этот объект, вы сначала должны удалить, то что зависит от него. Поскольку зависимости от других объектов существуют только для тела пакета, это тело пакета может быть легко удалено, даже если какой-нибудь другой объект зависит от этого пакета. Когда тело удаляется, заголовок остаётся, что позволяет пересоздать это тело после того, как сделаны изменения связанные с удалённым объектом.

Упрощение управления разрешениями

Поскольку Firebird выполняет подпрограммы с полномочиями вызывающей стороны, то каждой вызывающей подпрограмме необходимо предоставить полномочия на использования ресурсов, если эти ресурсы не являются непосредственно доступными вызывающей стороне. Использование каждой подпрограммы требует предоставления привилегий на её выполнение для пользователей и/или ролей.

У упакованных подпрограмм нет отдельных привилегий. Привилегии действуют на пакет в целом. Привилегии, предоставленные пакетам, действительны для всех подпрограмм тела пакета, в том числе частных, и сохраняются для заголовка пакета.

Частные области видимости

Некоторые процедуры и функции могут быть частными (private), а именно их использование разрешено только внутри определения пакета.

Все языки программирования имеют понятие области видимости подпрограмм, которое невозможно без какой-либо формы группировки. Пакеты Firebird в этом отношении подобны модулям Delphi. Если подпрограмма не объявлена в заголовке пакета (interface), но реализована в теле (implementation), то такая подпрограмма становится частной (private). Частную подпрограмму возможно вызвать только из её пакета.
7.5.2. Создание пакета

Для получения информации о создании пакетов см. CREATE PACKAGE, CREATE PACKAGE BODY.

7.5.3. Модификация пакета

Для получения информации об изменении существующего заголовка или тела пакета см. ALTER PACKAGE, CREATE OR ALTER PACKAGE, RECREATE PACKAGE, RECREATE PACKAGE BODY.

7.5.4. Удаление пакета

Для получения информации об удалении пакета см. DROP PACKAGE, DROP PACKAGE BODY.

7.6. Триггеры

Триггер является программой, которая хранится в области метаданных базы данных и выполняется на стороне сервера. Напрямую обращение к триггеру невозможно. Он вызывается автоматически при наступлении одного или нескольких событий, относящихся к одной конкретной таблице (к представлению), или при наступлении одного из событий базы данных.

Триггер, вызываемый при наступлении события таблицы, связан с одной таблицей или представлением, с одним или более событиями для этой таблицы или представления (INSERT, UPDATE, DELETE) и ровно с одной фазой такого события (BEFORE или AFTER).

Триггер выполняется в той транзакции, в контексте которой выполнялась программа, вызвавшая соответствующее событие. Исключением являются триггеры, реагирующие на события базы данных. Для некоторых из них запускается транзакция по умолчанию.

7.6.1. Порядок срабатывания

Для каждой комбинации фаза-событие может быть определено более одного триггера. Порядок, в котором они выполняются, может быть указан явно с помощью дополнительного аргумента POSITION в определении триггера. Максимальная позиция равна 32767. Триггеры с меньшей позицией вызываются первыми.

Если предложение POSITION опущено или несколько триггеров с одинаковыми фазой и событием имеют одну и ту же позицию, то такие триггеры будут выполнятся в алфавитном порядке их имен.

7.6.2. DML триггеры

DML триггеры вызываются при изменении состояния данных DML операциями: редактирование, добавление или удаление строк. Они могут быть определены и для таблиц и для представлений.
Варианты триггеров

Существует шесть основных вариантов соотношения событие-фаза для таблиц (представления):

- до добавления новой строки (BEFORE INSERT)
- после добавления новой строки (AFTER INSERT)
- до изменения строки (BEFORE UPDATE)
- после изменения строки (AFTER UPDATE)
- до удаления строки (BEFORE DELETE)
- после удаления строки (AFTER DELETE)

Помимо базовых форм с единственной фазой и событием Firebird поддерживает также формы с одной фазой и множеством событий, например BEFORE INSERT OR UPDATE OR DELETE или AFTER UPDATE OR DELETE или любая другая комбинация на ваш выбор.

Триггеры с несколькими фазами, такие как BEFORE OR AFTER … не поддерживаются.

Контекстные переменные INSERTING, UPDATING и DELETING логического типа могут быть использованы в теле триггера для определения события, которое вызвало срабатывание триггера.

Контекстные переменные NEW и OLD

В DML триггерах Firebird обеспечивает доступ к множеству контекстных переменных NEW и OLD. Каждое множество является массивом всей строки: OLD.* — значение строки до изменения данных и NEW.* — требуемое ("новое") значение строки. Операторы могут ссылаться на них используя следующие формы NEW.columnname и OLD.columnname. columnname может быть любым столбцом определённым в таблице(представлении), а не только тем что был изменён.

Контекстные переменные NEW и OLD подчиняются следующим правилам:

- Во всех триггерах контекстные переменные OLD доступны только для чтения;
- В триггерах BEFORE UPDATE и BEFORE INSERT переменные NEW доступны для чтения и записи, за исключением COMPUTED BY столбцов;
- В INSERT триггерах ссылка на переменные OLD не допускается и вызовет исключение;
- В DELETE триггерах ссылка на переменные NEW не допускается и вызовет исключение;
- Во всех AFTER триггерах переменные NEW доступны только для чтения.

7.6.3. Триггеры на события базы данных

Триггер, связанный с событиями базы данных, может вызываться при следующих событиях:
После соединения с базой данных, или сброса сессионного окружения  
ON CONNECT  
Перед выполнением триггера автоматически запускается транзакция по умолчанию

До отсоединения от базы данных, или сбросом сессионного окружения  
ON DISCONNECT  
Перед выполнением триггера автоматически запускается транзакция по умолчанию

После старта транзакции  
ON TRANSACTION START  
Триgger выполняется в контексте текущей транзакции

Перед подтверждением транзакции  
ON TRANSACTION COMMIT  
Триgger выполняется в контексте текущей транзакции

Перед отменой транзакции  
ON TRANSACTION ROLLBACK  
Триgger выполняется в контексте текущей транзакции

Контекстная переменная RESETTING может использоваться в триггерах на события ON CONNECT и ON DISCONNECT для того, чтобы отличить сброс сеанса от подключения/отключения от базы данных.

7.6.4. DDL триггеры

DDL триггеры срабатывают на указанные события изменения метаданных в одной из фаз события. BEFORE триггеры запускаются до изменений в системных таблицах. AFTER триггеры запускаются после изменений в системных таблицах.

Переменные доступные в пространстве имён DDL_TRIGGER

Во время работы DDL триггера доступно пространство имён DDL_TRIGGER для использования в функции RDB$GET_CONTEXT. Его использование также допустимо в хранимых процедурах и функциях, вызванных DDL триггерами.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом, в случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имен DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Переменные доступные в пространстве имён DDL_TRIGGER:* EVENT_TYPE — тип события (CREATE, ALTER, DROP)

- OBJECT_TYPE — тип объекта (TABLE, VIEW и др.)
- DDL_EVENT — имя события (<ddl event item>),
  где <ddl event item> = EVENT_TYPE || ' ' || OBJECT_TYPE
- OBJECT_NAME — имя объекта метаданных
7.6.5. Создание триггера

Для получения информации о создании триггеров см. CREATE TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER в главе “Операторы определения данных DDL”.

7.6.6. Изменение триггера

Для получения информации об изменении триггеров см. ALTER TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER в главе “Операторы определения данных DDL”.

7.6.7. Удаление триггера

Для получения информации об удалении триггеров см. DROP TRIGGER в главе “Операторы определения данных DDL”.

7.7. Написание кода тела модуля

В этом разделе подробно рассматривается процедурные конструкции языка SQL и операторы доступные в теле хранимых процедур, триггеров и анонимных PSQL блоков.

### Маркер двоеточия (‘:’)

Маркер двоеточия (‘:’) используется в PSQL, чтобы пометить ссылку на переменную в DML операторе. В остальных случаях маркер двоеточия необязателен перед именами переменных.

Никогда не задавайте префикс двоеточия для контекстных переменных.

#### 7.7.1. Оператор присваивания

**Назначение**

Присваивание переменной значения.

**Доступно в**

PSQL

**Синтаксис**

```
varname = <value_expr>;
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>varname</td>
<td>Имя локальной переменной или параметра процедуры (функции).</td>
</tr>
</tbody>
</table>
### Параметр | Описание
--- | ---
value_expr | Выражение, константа или переменная совместимая по типу данных с varname.

PSQL использует символ равенства (‘=’) в качестве своего оператора присваивания. Оператор присваивания устанавливает переменной слева от оператора значение SQL выражения справа. Выражением может быть любое правильное выражение SQL. Оно может содержать литералы, имена внутренних переменных, арифметические, логические и строковые операции, обращения к встроенным функциям и к функциям, определённым пользователем.

**Example 258. Использование оператора присваивания**

```sql
CREATE PROCEDURE MYPROC (
    a INTEGER,
    b INTEGER,
    name VARCHAR (30)
)
RETURNS (
    c INTEGER,
    str VARCHAR(100))
AS
BEGIN
    -- присваиваем константу
    c = 0;
    str = ''; SUSPEND;
    -- присваиваем значения выражений
    c = a + b;
    str = name || CAST(b AS VARCHAR(10)); SUSPEND;
    -- присваиваем значение выражения
    -- построенного с использованием запроса
    c = (SELECT 1 FROM rdb$database);
    -- присваиваем значение из контекстной переменной
    str = CURRENT_USER;
    SUSPEND;
END
```

*См. также:* 
**DECLARE VARIABLE.**

### 7.7.2. DECLARE VARIABLE

**Назначение**
Объявление локальной переменной.
Доступно в PSQL

Синтаксис

```
DECLARE [VARIABLE] varname
   <type> [NOT NULL] [COLLATE collation]
   [{= | DEFAULT} <initvalue>] }

<type> ::= 
   <non_array_datatype>
 | [TYPE OF] domain
 | TYPE OF COLUMN rel.col

<non_array_datatype> ::= 
   <scalar_datatype> | <blob_datatype>

<scalar_datatype> ::= См. Синтаксис скалярных типов данных

<blob_datatype> ::= См. Синтаксис типа данных BLOB

<initvalue> ::= {<literal> | <context_var>}
```

**Table 98. Параметры оператора DECLARE VARIABLE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>varname</td>
<td>Имя локальной переменной.</td>
</tr>
<tr>
<td>literal</td>
<td>Литерал.</td>
</tr>
<tr>
<td>context_var</td>
<td>Любая контекстная переменная, тип которой совместим с типом локальной переменной.</td>
</tr>
<tr>
<td>non_array_datatype</td>
<td>Тип данных SQL кроме массивов.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
<tr>
<td>domain</td>
<td>Домен.</td>
</tr>
<tr>
<td>rel</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>col</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
</tbody>
</table>

Оператор DECLARE [VARIABLE] объявляет локальную переменную. Ключевое слово VARIABLE можно опустить. В одном операторе разрешено объявлять только одну переменную. В процедурах и триггерах можно объявить произвольное число локальных переменных, используя при этом каждый раз, новый оператор DECLARE VARIABLE.

Имя локальной переменной должно быть уникально среди имён локальных переменных, входных и выходных параметров процедуры внутри программного объекта.
Типы данных для переменных

В качестве типа данных локальной переменной может быть любой SQL тип, за исключением массивов.

В качестве типа переменной можно указать имя домена. В этом случае переменная будет наследовать все характеристики домена. Если перед названием домена дополнительно используется предложение TYPE OF, то используется только тип данных домена — не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Локальные переменные можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов еще набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Ограничение NOT NULL

Для локальных переменных можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Предложения CHARACTER SET и COLLATE

Если не указано иное, набор символов и последовательность сопоставления (сортировки) строковой переменной будут значениями по умолчанию для базы данных.

При необходимости можно включить предложение CHARACTER SET для обработки строковых данных, которые будут находиться в другом наборе символов.

Допустимая последовательность сопоставления (предложение COLLATE) также может быть включена с CHARACTER SET или без него.

Инициализация переменной

Локальной переменной можно устанавливать инициализирующее (начальное) значение. Это значение устанавливается с помощью предложения DEFAULT или оператора “=”. В качестве значения по умолчанию может быть использовано значение NULL, литерал и любая контекстная переменная совместимая по типу данных.

Обязательно используйте инициализацию начальным значением для любых переменных объявленных с ограничением NOT NULL, если они не получают значение по умолчанию иным способом.

Примеры объявления локальных переменных
Example 259. Различные способы объявления локальных переменных

```
CREATE OR ALTER PROCEDURE SOME_PROC
AS
  -- Объявление переменной типа INT
  DECLARE I INT;
  -- Объявление переменной типа INT не допускающей значение NULL
  DECLARE VARIABLE J INT NOT NULL;
  -- Объявление переменной типа INT со значением по умолчанию 0
  DECLARE VARIABLE K INT DEFAULT 0;
  -- Объявление переменной типа INT со значением по умолчанию 1
  DECLARE VARIABLE L INT = 1;
  -- Объявление переменной на основе домена COUNTRYNAME
  DECLARE FARM_COUNTRY COUNTRYNAME;
  -- Объявление переменной с типом равным типу домена COUNTRYNAME
  DECLARE FROM_COUNTRY TYPE OF COUNTRYNAME;
  -- Объявление переменной с типом столбца CAPITAL таблицы COUNTRY
  DECLARE CAPITAL TYPE OF COLUMN COUNTRY.CAPITAL;
BEGIN
  /* Операторы PSQL */
END
```

Cм. также:

Типы и подтипы данных, Пользовательские типы данных — домены, CREATE DOMAIN

### 7.7.3. DECLARE … CURSOR

**Назначение:**

Объявление курсора.

**Доступно в:**

PSQL

**Синтаксис**

```
DECLARE [VARIABLE] cursor_name
  [SCROLL | NO SCROLL]
  CURSOR FOR (<select_statement>);
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor_name</td>
<td>Имя курсора.</td>
</tr>
<tr>
<td>select_statement</td>
<td>Оператор SELECT.</td>
</tr>
</tbody>
</table>

Оператор DECLARE … CURSOR FOR объявляет именованный курсор, связывая его с набором данных, полученным в операторе SELECT, указанном в предложении CURSOR FOR. В
дальнейшем курсор может быть открыт, использоваться для обхода результирующего набора данных, и снова быть закрытым. Также поддерживаются позиционированные обновления и удаления при использовании WHERE CURRENT OF в операторах UPDATE и DELETE.

Имя курсора можно использовать в качестве ссылки на курсор, как на переменные типа запись. Текущая запись доступна через имя курсора, что делает необязательным предложение INTO в операторе FETCH.

Однонаправленные и прокручиваемые курсоры

Курсор может быть однонаправленными прокручиваемым. Необязательное предложение SCROLL делает курсор двунаправленным (прокручиваемым), предложение NO SCROLL — однонаправленным. По умолчанию курсоры являются однонаправленными.

Однонаправленные курсоры позволяют двигаться по набору данных только вперёд. Двунаправленные курсоры позволяют двигаться по набору данных не только вперёд, но и назад, а также на N позиций относительно текущего положения.

Прокручиваемые курсоры материализуются внутри как временный набор данных, таким образом, они потребляют дополнительные ресурсы памяти/диска, поэтому пользуйтесь ими только тогда, когда это действительно необходимо.

Особенности использования курсора

- Предложение FOR UPDATE разрешено использовать в операторе SELECT, но оно не требуется для успешного выполнения позиционированного обновления или удаления;
- Удостоверьтесь, что объявленные имена курсоров не совпадают, ни с какими именами, определёнными позже в предложениях AS CURSOR;
- Если курсор требуется только для прохода по результирующему набору данных, то практически всегда проще (и менее подвержено ошибкам) использовать оператор FOR SELECT с предложением AS CURSOR. Объявленные курсоры должны быть явно открыты, использованы для выборки данных и закрыты. Кроме того, вы должны проверить контекстную переменную ROW_COUNT после каждой выборки и выйти из цикла, если её значение ноль. Предложение FOR SELECT делает эту проверку автоматически. Однако объявленные курсоры дают большие возможности для контроля над последовательными событиями и позволяют управлять несколькими курсорами параллельно;
- Оператор SELECT может содержать параметры, например: "SELECT NAME || :SFX FROM NAMES WHERE NUMBER = :NUM". Каждый параметр должен быть заранее объявлен как переменная PSQL (это касается также входных и выходных параметров). При открытии курсора параметру присваивается текущее значение переменной;
- Если опция прокрутки опущена, то по умолчанию принимается NO SCROLL (т.е курсор открыт для движения только вперёд). Это означает, что могут быть использованы только команды FETCH [NEXT FROM]. Другие команды будут возвращать ошибки.

Если значение переменной PSQL, используемой в операторе SELECT,
изменяется во время выполнения цикла, то её новое значение может (но не всегда) использоваться при выборке следующих строк. Лучше избегать таких ситуаций. Если вам действительно требуется такое поведение, то необходимо тщательно протестировать код и убедиться, что вы точно знаете, как изменения переменной влияют на результаты выборки. Особо отметчу, что поведение может зависеть от плана запроса, в частности, от используемых индексов. В настоящее время нет строгих правил для таких ситуаций, но в новых версиях Firebird это может измениться.

Примеры использования именованного курсора

Example 260. Объявление именованного курсора

```sql
CREATE OR ALTER TRIGGER TBU_STOCK
BEFORE UPDATE ON STOCK
AS
  -- Объявление именованного курсора
  DECLARE C_COUNTRY CURSOR FOR (
    SELECT
      COUNTRY,
      CAPITAL
    FROM COUNTRY
  );
BEGIN
  /* Операторы PSQL */
END
```

Example 261. Объявление прокручиваемого курсора

```sql
EXECUTE BLOCK
RETURNS (N INT, RNAME CHAR(63))
AS
  -- Объявление прокручиваемого курсора
  DECLARE C SCROLL CURSOR FOR (
    SELECT
      ROW_NUMBER() OVER(ORDER BY RDB$RELATION_NAME) AS N,
      RDB$RELATION_NAME
    FROM RDB$RELATIONS
    ORDER BY RDB$RELATION_NAME);
BEGIN
  /* Операторы PSQL */
END
```

См. также:
OPEN, FETCH, CLOSE, FOR SELECT.

7.7.4. DECLARE PROCEDURE

Назначение
Объявление и реализация подпрограммы.

Доступно в PSQL

Синтаксис

\[
<\text{subproc-declaration}> ::= \\
\text{DECLARE PROCEDURE} \text{subprocname} [[(\text{input-parameters})]]  \\
[\text{RETURNS} (\text{output-parameters})];
\]

\[
<\text{subproc-implimentation}> ::= \\
\text{DECLARE PROCEDURE} \text{subprocname} [[(\text{input-parameters})]]  \\
[\text{RETURNS} (\text{output-parameters})] \\
<\text{psql-routine-body}>
\]

\[
\text{input-parameters} ::= \text{inparam} [, \text{inparam} ...]
\]

\[
\text{output-parameters} ::= \text{outparam} [, \text{outparam} ...]
\]

\[
<\text{psql-routine-body}> ::=
\text{См. Синтаксис тела модуля}
\]

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>subprocname</td>
<td>Имя подпрограммы.</td>
</tr>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
<tr>
<td>outparam</td>
<td>Описание выходного параметра.</td>
</tr>
</tbody>
</table>

Таблица 100. Параметры оператора DECLARE PROCEDURE

Оператор DECLARE PROCEDURE объявляет или реализует подпрограмму.

На подпрограмму накладываются следующие ограничения:

- Подпрограмма не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции, триггере и анонимном PSQL блоке);

- В настоящее время подпрограмма не имеет прямого доступа для использования переменных, курсоров из основного модуля. Это может быть разрешено в будущем.

Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя...
объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее.

Example 262. Использование подпроцедуры

```sql
SET TERM ^;
--
-- Подпроцедуры в EXECUTE BLOCK
--
EXECUTE BLOCK
RETURNS (name VARCHAR(63))
AS
  -- Подпроцедура, возвращающая список таблиц
  DECLARE PROCEDURE get_tables
  RETURNS(table_name VARCHAR(63))
  AS
  BEGIN
    FOR SELECT rdb$relation_name
    FROM rdb$relations
    WHERE rdb$view_blr IS NULL
    INTO table_name
    DO SUSPEND;
  END

  -- Подпроцедура, возвращающая список представлений
  DECLARE PROCEDURE get_views
  RETURNS(view_name VARCHAR(63))
  AS
  BEGIN
    FOR SELECT rdb$relation_name
    FROM rdb$relations
    WHERE rdb$view_blr IS NOT NULL
    INTO view_name
    DO SUSPEND;
  END

BEGIN
  FOR SELECT
```

Chapter 7. Операторы процедурного SQL (PSQL)
Example 263. Использование подпроцедур с предварительным объявлением

```sql
EXECUTE BLOCK RETURNS (o INTEGER)
AS
  -- Предварительное объявление P1.
  DECLARE PROCEDURE p1(i INTEGER = 1) RETURNS (o INTEGER);

  -- Предварительное объявление P2.
  DECLARE PROCEDURE p2(i INTEGER) RETURNS (o INTEGER);

  -- Реализация P1. Вы не должны переопределять значение параметра по умолчанию
  DECLARE PROCEDURE p1(i INTEGER) RETURNS (o INTEGER)
  AS
  BEGIN
    EXECUTE PROCEDURE p2(i) RETURNING_VALUES o;
  END

  DECLARE PROCEDURE p2(i INTEGER) RETURNS (o INTEGER)
  AS
  BEGIN
    o = i;
  END
  BEGIN
    EXECUTE PROCEDURE p1 RETURNING_VALUES o;
    SUSPEND;
  END!
```

См. также:

DECLARE FUNCTION, CREATE PROCEDURE.

7.7.5. DECLARE FUNCTION

Назначение

Объявление и реализация подфункции.
Доступно в
PSQL

Синтаксис

\[
<\text{subfunc-declaration}> ::= \\
  \text{DECLARE FUNCTION } \text{subfuncname}\ [(<\text{input-parameters}>)] \\
  \text{RETURNS } <\text{type}> \ [\text{COLLATE } \text{collation}] \ [\text{DETERMINISTIC}];
\]

\[
<\text{subfunc-implimentation}> ::= \\
  \text{DECLARE FUNCTION } \text{subfuncname}\ [(<\text{input-parameters}>)] \\
  \text{RETURNS } <\text{type}> \ [\text{COLLATE } \text{collation}] \ [\text{DETERMINISTIC}] \\
  <\text{psql-routine-body}>
\]

\[
<\text{input-parameters}> ::= <\text{inparam}> \ [, <\text{inparam}> ...]
\]

\[
<\text{output-parameters}> ::= <\text{outparam}> \ [, <\text{outparam}> ...]
\]

\[
<\text{psql-routine-body}> ::= \\
  \text{См. Синтаксис тела модуля}
\]

Таблица 101. Параметры оператора DECLARE FUNCTION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>subfuncname</td>
<td>Имя подфункции.</td>
</tr>
<tr>
<td>inparam</td>
<td>Описание входного параметра.</td>
</tr>
<tr>
<td>type</td>
<td>Тип выходного результата.</td>
</tr>
<tr>
<td>collation</td>
<td>Порядок сортировки.</td>
</tr>
</tbody>
</table>

Оператор DECLARE FUNCTION объявляет подфункцию.

На подфункцию накладываются следующие ограничения:

- Подпрограмма не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции, триггере и анонимном PSQL блоке);
- В настоящее время подпрограмма не имеет прямого доступа для использования переменных, курсоров из основного модуля. Это может быть разрешено в будущем.

Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее.
**Example 264. Использование подфункции**

```sql
--
-- Подфункция внутри хранимой функции
--
CREATE OR ALTER FUNCTION FUNC1 (n1 INTEGER, n2 INTEGER)
    RETURNS INTEGER
AS
    -- Подфункция
    DECLARE FUNCTION SUBFUNC (n1 INTEGER, n2 INTEGER)
        RETURNS INTEGER
    AS
    BEGIN
        RETURN n1 + n2;
    END
BEGIN
    RETURN SUBFUNC(n1, n2);
END
```

**Example 265. Использование рекурсивной подфункции**

```sql
EXECUTE BLOCK RETURNS (i INTEGER, o INTEGER)
AS
    -- Рекурсивная подпрограмма-функция без предварительного объявления.
    DECLARE FUNCTION fibonacci(n INTEGER) RETURNS INTEGER
    AS
    BEGIN
        IF (n = 0 OR n = 1) THEN
            RETURN n;
        ELSE
            RETURN fibonacci(n - 1) + fibonacci(n - 2);
        END
    BEGIN
    i = 0;

    WHILE (i < 10)
    DO
    BEGIN
        o = fibonacci(i);
        SUSPEND;
        i = i + 1;
    END
    END
```

См. также:
DECLARE PROCEDURE, CREATE FUNCTION.

7.7.6. BEGIN … END

Назначение
Обозначение составного оператора.

Доступно в
PSQL.

Синтаксис

BEGIN
  [<compound_statement> ...]
END

Операторные скобки BEGIN … END определяют составной оператор или блок операторов, который выполняется как одна единица кода. Каждый блок начинается оператором BEGIN и завершается оператором END. Блоки могут быть вложенными. Максимальная глубина ограничена 512 уровнями вложенности блоков. Составной оператор может быть пустым, что позволяет использовать его как заглушку, позволяющую избежать написания фиктивных операторов.

После операторов BEGIN и END точка с запятой не ставится. Однако утилита командной строки isql требует, чтобы после последнего оператора END в определении PSQL модуля следовал символ терминатора, установленного командой SET TERM. Терминатор не является частью синтаксиса PSQL.

Последний оператор END в триггере завершает работу триггера. Последний оператор END в хранимой процедуре работает в зависимости от типа процедуры:

• В селективной процедуре последний оператор END возвращает управление приложению и устанавливает значение SQLCODE равным 100, что означает, что больше нет строк для извлечения;
• В выполняемой процедуре последний оператор END возвращает управление и текущие значения выходных параметров, если таковые имеются, вызывающему приложению.

Примеры BEGIN … END

Пример процедуры из базы данных employee.fdb, демонстрирующий простое использование блоков BEGIN … END:

Example 266. Использование BEGIN … END

SET TERM ^;
CREATE OR ALTER PROCEDURE DEPT_BUDGET (DNO CHAR(3))
RETURNS (TOT DECIMAL(12,2))
AS
DECLARE VARIABLE SUMB DECIMAL(12,2);
DECLARE VARIABLE RDNO CHAR(3);
DECLARE VARIABLE CNT INTEGER;
BEGIN
TOT = 0;
SELECT
    BUDGET
FROM
    DEPARTMENT
WHERE DEPT_NO = :DNO
INTO :TOT;
SELECT
    COUNT(BUDGET)
FROM
    DEPARTMENT
WHERE HEAD_DEPT = :DNO
INTO :CNT;
IF (CNT = 0) THEN
    SUSPEND;
FOR
    SELECT
        DEPT_NO
    FROM
        DEPARTMENT
    WHERE HEAD_DEPT = :DNO
    INTO :RDNO
DO
    BEGIN
        EXECUTE PROCEDURE DEPT_BUDGET(:RDNO)
        RETURNING_VALUES :SUMB;
        TOT = TOT + SUMB;
    END
    SUSPEND;
END
SET TERM ;^
7.7.7. IF … THEN … ELSE

Назначение
Условный переход.

Доступно в
PSQL

Синтаксис

IF (<condition>)
    THEN <compound_statement>
    [ELSE <compound_statement>]

Table 102. Параметры оператора IF … THEN … ELSE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Логическое условие возвращающее TRUE, FALSE или UNKNOWN.</td>
</tr>
<tr>
<td>compound_statement</td>
<td>Составной оператор (оператор или блок операторов).</td>
</tr>
</tbody>
</table>

Оператор условного перехода IF используется для выполнения ветвления процесса обработки данных в PSQL. Если условие возвращает значение TRUE, то выполняется составной оператор или после ключевого слова THEN. Иначе (если условие возвращает FALSE или UNKNOWN) выполняется составной оператор после ключевого слова ELSE, если оно присутствует. Условие всегда заключается в круглые скобки.

Оператор ветвления

PSQL не обеспечивает более сложных переходов с несколькими ветвями, таких как CASE или SWITCH. Однако можно объединить операторы IF … THEN … ELSE в цепочку, см. Раздел примеров ниже. В качестве альтернативы, оператор CASE из DSQL доступен в PSQL и может удовлетворить по крайней мере некоторые варианты использования в виде switch:

```
CASE <test_expr>
    WHEN <expr> THEN <result>
    [WHEN <expr> THEN <result> ...]
    [ELSE <defaultresult>]
END

CASE
    WHEN <bool_expr> THEN <result>
    [WHEN <bool_expr> THEN <result> ...]
    [ELSE <defaultresult>]
END
```
Example 267. Использования CASE в PSQL.

```sql
... 
C = CASE
  WHEN A=2 THEN 1
  WHEN A=1 THEN 3
  ELSE 0
END;
...
```

Примеры If

Example 268. Использование оператора IF

Предположим, что переменные FIRST, LINE2 и LAST были объявлены ранее.

```sql
... 
IF (FIRST IS NOT NULL) THEN 
  LINE2 = FIRST || ' ' || LAST;
ELSE 
  LINE2 = LAST;
... 
```

Example 269. Объединение IF ... THEN ... ELSE в цепочку

Предположим, что переменные INT_VALUE и STRING_VALUE были объявлены ранее.

```sql
... 
IF (INT_VALUE = 1) THEN 
  STRING_VALUE = 'one';
ELSE IF (INT_VALUE = 2) THEN 
  STRING_VALUE = 'two';
ELSE IF (INT_VALUE = 3) THEN 
  STRING_VALUE = 'three';
ELSE 
  STRING_VALUE = 'too much';
... 
```

Этот пример можно заменить на функцию Простой CASE или DECODE.

См. также:

WHILE ... DO, CASE.
**7.7.8. WHILE … DO**

**Назначение**
Циклическое выполнение операторов.

**Доступно в**
PSQL

**Синтаксис**

```
[label:]
WHILE (<condition>) DO
<compound_statement>
```

**Table 103. Параметры оператора WHILE ... DO**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Логическое условие возвращающее TRUE, FALSE или UNKNOWN.</td>
</tr>
<tr>
<td>compound_statement</td>
<td>Составной оператор (оператор или блок операторов).</td>
</tr>
</tbody>
</table>

Оператор WHILE используется для организации циклов в PSQL. Составной оператор будет выполняться до тех пор, пока условие истинно (возвращает TRUE). Циклы могут быть вложенными, глубина вложения не ограничена.

**Примеры WHILE … DO**

**Example 270. Использование оператора WHILE ... DO**

Процедура расчёта суммы от 1 до I для демонстрации использования цикла:

```
CREATE PROCEDURE SUM_INT (I INTEGER)
RETURNS (S INTEGER)
AS
BEGIN
  s = 0;
  WHILE (i > 0) DO
    BEGIN
      s = s + i;
      i = i - 1;
    END
  END
END
```

При выполнении в *isql*:

```
EXECUTE PROCEDURE SUM_INT(4);
```

результат будет следующий
См. также:
FOR SELECT, FOR EXECUTE STATEMENT, LEAVE, CONTINUE.

7.7.9. BREAK

Назначение
Выход из цикла.

Синтаксис

```
<loop_stmt>
BEGIN
...
  BREAK;
...
END
```

Таблица 104. Параметры оператора BREAK

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_stmt</td>
<td>Оператор SELECT</td>
</tr>
<tr>
<td>condition</td>
<td>Логическое условие возвращающее TRUE, FALSE или UNKNOWN.</td>
</tr>
</tbody>
</table>

Оператор BREAK моментально прекращает работу внутреннего цикла операторов WHILE или FOR. Код продолжает выполняться с первого оператора после завершенного блока цикла.

Оператор BREAK похож на LEAVE, за исключением того, что не поддерживает метку перехода.

Этот оператор считается устаревшим. Начиная с Firebird 1.5 рекомендуется использовать SQL-99 совместимый оператор LEAVE.

См. также:
LEAVE, EXIT, CONTINUE.

7.7.10. LEAVE

Назначение
Выход из цикла.

Доступно в PSQL

Синтаксис

```sql
[label:]<loop_stmt>
BEGIN
...
LEAVE [label];
...
END

<loop_stmt> ::= 
 | FOR <select_stmt> INTO <var_list> DO 
 | FOR EXECUTE STATEMENT ... INTO <var_list> DO 
 | WHILE (<condition>) DO
```

Таблица 105. Параметры оператора LEAVE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Метка.</td>
</tr>
<tr>
<td>select_stmt</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>condition</td>
<td>Логическое условие возвращающее TRUE, FALSE или UNKNOWN.</td>
</tr>
</tbody>
</table>

Оператор LEAVE немедленно прекращает работу внутреннего цикла операторов WHILE или FOR. С использованием необязательного параметра label, LEAVE также может выйти и из внешнего цикла, то есть цикла помеченного меткой `. Код продолжает выполняться с первого оператора после завершенного блока цикла.

Примеры LEAVE

**Example 271. Использование оператора LEAVE**

В этом примере выход из цикла произойдёт при возникновении ошибки вставки в таблицу NUMBERS. Код продолжит своё выполнение с оператора C = 0.

```sql
...
WHILE (B < 10) DO
BEGIN
    INSERT INTO NUMBERS(B)
    VALUES (:B);
    B = B + 1;
    WHEN ANY DO
    BEGIN
        EXECUTE PROCEDURE LOG_ERROR (
```
Example 272. Использование оператора LEAVE с меткой

В этом примере оператор LEAVE LOOPA завершает внешний цикл, а LEAVE LOOPB — внутренний.

Обратите внимание: простого оператора LEAVE также было бы достаточно, чтобы завершить внутренний цикл.

... 
STMT1 = 'SELECT NAME FROM FARMS';
LOOPA:
FOR EXECUTE STATEMENT :STMT1 
INTO :FARM DO
BEGIN
STMT2 = 'SELECT NAME ' || 'FROM ANIMALS WHERE FARM = ''';
LOOPB:
FOR EXECUTE STATEMENT :STMT2 || :FARM || '''
INTO :ANIMAL DO
BEGIN
IF (ANIMAL = 'FLUFFY') THEN
LEAVE LOOPB;
ELSE IF (ANIMAL = FARM) THEN
LEAVE LOOPA;
ELSE
SUSPEND;
END
END
END
...

См. также:
BREAK, EXIT, CONTINUE.

7.7.11. CONTINUE

Назначение
Досрочное начало новой итерации цикла.

Доступно в
**PSQL**

**Синтаксис**

```psql
[label:] <loop_stmt>
BEGIN
...
  CONTINUE [label];
...
END
```

Атрибут <loop_stmt> ::= 
| FOR <select_stmt> INTO <var_list> DO |
| FOR EXECUTE STATEMENT ... INTO <var_list> DO |
| WHILE (<condition>) DO |

**Table 106. Параметры оператора CONTINUE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Метка.</td>
</tr>
<tr>
<td>select_stmt</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>condition</td>
<td>Логическое условие возвращающее TRUE, FALSE или UNKNOWN.</td>
</tr>
</tbody>
</table>

Оператор CONTINUE пропускает оставшуюся часть текущего блока цикла и запускает следующую итерацию текущего цикла WHILE или FOR. С использованием необязательного параметра `label`, CONTINUE также может начинать следующую итерацию для внешнего цикла, то есть есть цикла, помеченного меткой `label`.

**Примеры CONTINUE**

**Example 273. Использование оператора CONTINUE**

```psql
FOR 
  SELECT A, D FROM ATABLE INTO :achar, :ddate
DO BEGIN
  IF (ddate < current_data - 30) THEN
    CONTINUE;
  ELSE
    /* do stuff */
  END
...
```

**Cм. также:**

*LEAVE, BREAK.*
7.7.12. EXIT

Назначение
Завершение работы процедуры, функции или триггера.

Доступно в
PSQL

Синтаксис

EXIT;

Оператор EXIT, вызванный из любой точки выполняющегося PSQL модуля, переходит на последний оператор END, таким образом завершая выполнение программы.

Вызов EXIT в функции приведет к тому, что функция вернет NULL.

Примеры EXIT

Example 274. Использование оператора EXIT в селективной хранимой процедуре.

```sql
CREATE PROCEDURE GEN_100
RETURNS (I INTEGER)
AS
BEGIN
  I = 1;
  WHILE (I=1) DO
    BEGIN
      SUSPEND;
      IF (I=100) THEN
        EXIT;
      I = I + 1;
    END
  END
END
```

См. также:
LEAVE, BREAK, CONTINUE, SUSPEND.

7.7.13. SUSPEND

Назначение
Передача значений параметров в буфер и приостановка выполнения процедуры (PSQL блока) до тех пор, пока вызывающая сторона не получит результат.

Доступно в
Оператор SUSPEND передаёт значения выходных параметров в буфер и приостанавливает выполнение хранимой процедуры (PSQL блока). Выполнение остаётся приостановленным до тех пор, пока вызывающая сторона не получит содержимое буфера. Выполнение возобновляется с оператора, следующего непосредственно после оператора SUSPEND. Чаще всего это будет новой итерацией циклического процесса.

1. Оператор SUSPEND может встречаться только в хранимых процедурах или подпроцедурах, а также в анонимных блоках EXECUTE BLOCK.

2. Наличие ключевого слова SUSPEND определяет хранимую процедуру как выбираваемую (selectable) процедуру.

3. Приложения, использующие API интерфейсы, обычно делают выборку из хранимых процедур прозрачно.

4. Если выбираваемая (selectable) процедура выполняется с использованием EXECUTE PROCEDURE, она ведет себя как исполняемая процедура. Когда в такой хранимой процедуре выполняется инструкция SUSPEND, это то же самое, что выполнение инструкции EXIT, что приводит к немедленному завершению процедуры.

5. Оператор SUSPEND “нарушает” атомарность блока, внутри которого он находится. В случае возникновения ошибки в селективной процедуре, операторы, выполненные после последнего оператора SUSPEND, будут откачены. Операторы, выполненные до последнего оператора SUSPEND, не будут откачены, если не будет выполнен откат транзакции.

**Примеры SUSPEND**

*Example 275. Использование оператора SUSPEND в селективной хранимой процедуре.*

```sql
CREATE PROCEDURE GEN_100
RETURNS (I INTEGER)
AS
BEGIN
  I = 1;
  WHILE (I=1) DO
    BEGIN
      SUSPEND;
      IF (I=100) THEN EXIT;
    I = I + 1;
  END;
```

431
См. также: EXIT.

7.7.14. EXECUTE STATEMENT

Назначение
Выполнение динамически созданных SQL операторов.

Доступно в
PSQL

Синтаксис

\[
\begin{align*}
<execute_statement> & ::= \\
& \text{EXECUTE STATEMENT} <argument> \\
& \quad [\text{option} \; ...] \\
& \quad [\text{INTO} \; <\text{variables}>]
\end{align*}
\]

\[
<argument> ::= \\
\quad <\text{paramless_stmt}> \\
\quad | \; (<\text{paramless_stmt}> ) \\
\quad | \; ( <\text{stmt_with_params}> ) <\text{param_values}>)
\]

\[
<\text{param_values}> ::= <\text{named_values}> | <\text{positional_values}>
\]

\[
<\text{named_values}> ::= \\
\quad [\text{EXCESS}] \; \text{paramname} := <\text{value_expr}> \\
\quad [, \; [\text{EXCESS}] \; \text{paramname} := <\text{value_expr}> \; ...]
\]

\[
<\text{positional_values}> ::= <\text{value_expr}> [, <\text{value_expr}> \; ...]
\]

\[
<\text{option}> ::= \\
\quad \text{WITH} \; \{\text{AUTONOMOUS} \; | \; \text{COMMON}\} \; \text{TRANSACTION} \\
\quad | \; \text{WITH} \; \text{CALLER PRIVILEGES} \\
\quad | \; \text{AS} \; \text{USER} \; \text{user} \\
\quad | \; \text{PASSWORD} \; \text{password} \\
\quad | \; \text{ROLE} \; \text{role} \\
\quad | \; \text{ON} \; \text{EXTERNAL} \; [\text{DATA SOURCE}] \; <\text{connect_string}>
\]

\[
<\text{connection_string}> ::= \\
\quad \text{См.} \; <\text{filespec} \; \text{в Синтаксис CREATE DATABASE} \; !!
\]

\[
<\text{variables}> ::= [:] \; \text{name} \; [, [:] \; \text{name} \; ...]
\]

Table 107. Параметры оператора EXECUTE STATEMENT
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>paramless_stmt</td>
<td>Строчный литерал или переменная, содержащая не параметризованный SQL запрос.</td>
</tr>
<tr>
<td>stmt_with_params</td>
<td>Строчный литерал или переменная, содержащая параметризованный SQL запрос.</td>
</tr>
<tr>
<td>paramname</td>
<td>Имя параметра SQL запроса.</td>
</tr>
<tr>
<td>value_expr</td>
<td>Выражение для получения значения параметра запроса.</td>
</tr>
<tr>
<td>user</td>
<td>Имя пользователя. Может быть строкой, CURRENT_USER или переменной.</td>
</tr>
<tr>
<td>password</td>
<td>Пароль. Может быть строкой или переменной.</td>
</tr>
<tr>
<td>role</td>
<td>Роль. Может быть строкой, CURRENT_ROLE или переменной.</td>
</tr>
<tr>
<td>connection_string</td>
<td>Строка соединения с удалённой БД Может быть строкой или переменной.</td>
</tr>
<tr>
<td>varname</td>
<td>Переменная.</td>
</tr>
</tbody>
</table>

Оператор `EXECUTE STATEMENT` принимает строковый параметр и выполняет его, как будто это оператор DSQL. Если оператор возвращает данные, то с помощью предложения INTO их можно передать в локальные переменные.

**Параметризованные операторы**

В DSQL операторе можно использовать параметры. Параметры могут быть именованными и позиционными (безымянными). Значение должно быть присвоено каждому параметру.

**Особенности использования параметризованных операторов**

1. Одновременное использование именованных и позиционных параметров в одном запросе запрещено;
2. Если у оператора есть параметры, они должны быть помещены в круглые скобки при вызове `EXECUTE STATEMENT`, независимо от вида их представления: непосредственно в виде строки, как имя переменной или как выражение;
3. Именованным параметрам должно предшествовать двоеточие (`:`) в самом операторе, но не при присвоении значения параметру;
4. Передача значений безымянным параметрам должна происходить в том же порядке, в каком они встречаются в тексте запроса;
5. Присвоение значений параметров должно осуществляться при помощи специального оператора “:=”, аналогичного оператору присваивания языка Pascal;
6. Каждый именованный параметр может использоваться в операторе несколько раз, но только один раз при присвоении значения;
7. Для позиционных параметров число подставляемых значений должно точно равняться числу параметров (вопросительных знаков) в операторе;
8. Необязательное ключевое слово `EXCESS` обозначает, что данный именованный параметр
необязательно должен упоминаться в тексте запроса. Обратите внимание, что все не EXCESS параметры должны присутствовать в запросе.

Примеры EXECUTE STATEMENT с параметрами

Example 276. Использование EXECUTE STATEMENT с именованными параметрами:

```sql
...  
DECLARE license_num VARCHAR(15);
DECLARE connect_string VARCHAR(100);
DECLARE stmt VARCHAR (100) =
   'SELECT license
    FROM cars
    WHERE driver = :driver AND location = :loc';
BEGIN
...
SELECT connstr
FROM databases
WHERE cust_id = :id
INTO connect_string;
...
FOR
    SELECT id
    FROM drivers
    INTO current_driver
DO
BEGIN
    FOR
        SELECT location
        FROM driver_locations
        WHERE driver_id = :current_driver
        INTO current_location
    DO
    BEGIN
...
EXECUTE STATEMENT (stmt)
(driver := current_driver,
loc := current_location)
ON EXTERNAL connect_string
INTO license_num;
...
```

Example 277. Использование EXECUTE STATEMENT с позиционными параметрами:

```sql
DECLARE license_num VARCHAR (15);
DECLARE connect_string VARCHAR (100);
DECLARE stmt VARCHAR (100) =
   'SELECT license
```

434
FROM cars
WHERE driver = ? AND location = ?'
BEGIN
...
SELECT connstr
FROM databases
WHERE cust_id = :id
INTO connect_string;
...
FOR SELECT id
FROM drivers
INTO current_driver
DO
BEGIN
FOR
SELECT location
FROM driver_locations
WHERE driver_id = :current_driver
INTO current_location
DO
BEGIN
...
EXECUTE STATEMENT (stmt)
(current_driver, current_location)
ON EXTERNAL connect_string
INTO license_num;
...

Example 278. Использование EXECUTE STATEMENT с избыточными (EXCESS) параметрами:

CREATE PROCEDURE P_EXCESS (A_ID INT, A_TRAN INT = NULL, A_CONN INT = NULL)
RETURNS (ID INT, TRAN INT, CONN INT)
AS
DECLARE S VARCHAR(255);
DECLARE W VARCHAR(255) = '';
BEGIN
S = 'SELECT * FROM TTT WHERE ID = :ID';

IF (A_TRAN IS NOT NULL)
THEN W = W || ' AND TRAN = :a';

IF (A_CONN IS NOT NULL)
THEN W = W || ' AND CONN = :b';

IF (W <> '')
THEN S = S || W;

-- could raise error if TRAN or CONN is null
-- FOR EXECUTE STATEMENT (:S) (a := :A_TRAN, b := A_CONN, id := A_ID)
WITH {AUTONOMOUS | COMMON} TRANSACTION

По умолчанию оператор выполняется в контексте текущей транзакции. При использовании предложения WITH AUTONOMOUS TRANSACTION запускается новая транзакция с такими же параметрами, как и текущая. Она будет подтверждена, если оператор выполнен без ошибок и отменена (откачена) в противном случае. С предложением WITH COMMON TRANSACTION по возможности используется текущая транзакция.

Если оператор должен работать в отдельном соединении, то используется уже запущенная в этом соединении транзакция (если таковая транзакция имеется). В противном случае стартует новая транзакция с параметрами текущей транзакции. Любые новые транзакции, запущенные в режиме "COMMON", подтверждаются или откатываются вместе с текущей транзакцией.

WITH CALLER PRIVILEGES

По умолчанию операторы SQL выполняются с правами текущего пользователя. Спецификация WITH CALLER PRIVILEGES добавляет к ним привилегии для вызова хранимой процедуры или триггера, так же как если бы оператор выполнялся непосредственно подпрограммой. WITH CALLER PRIVILEGES не имеет никакого эффекта, если также присутствует предложение ON EXTERNAL.

ON EXTERNAL [DATA SOURCE]

С предложением ON EXTERNAL DATA SOURCE оператор выполняется в отдельном соединении с той же или другой базой данных, возможно даже на другом сервере. Если строка подключения имеет значение NULL или '' (пустая строка), предложение ON EXTERNAL считается отсутствующим и оператор выполняется для текущей базы данных. Стока подключения подробно описана в операторе CREATE DATABASE см. Создание БД на удаленном сервере.

При выполнении оператора в отдельном соединении используется пул соединений и пул транзакций.

Пул внешних подключений (External connection pool)

Чтобы избежать задержек при частом использовании внешних соединений, подсистема внешних источников данных (EDS) использует пул внешних подключений. Пул сохраняет неиспользуемые внешние соединения в течении некоторого времени, что позволяет избегать затрат на подключение/отключение для часто используемых строк подключения.

Как работает пул соединений:
• каждое внешнее соединение связывается с пулом при создании;
• пул имеет два списка: неиспользуемых соединений и активных соединений;
• когда соединение становится неиспользуемым (т. е. у него нет активных запросов и нет активных транзакций), то оно сбрасывается и помещается в список ожидающих (при успешном завершении сброса) или закрывается (если при сбросе произошла ошибка). Соединение сбрасывается при помощи инструкции ALTER SESSION RESET. Сброс считается успешным, если не возникла ошибка.

Если внешний источник данных не поддерживает оператор ALTER SESSION RESET, то это не считается ошибкой, и такое соединение будет помещено в пул.

• если пул достиг максимального размера, то самое старое бездействующее соединение закрывается;
• когда Firebird просит создать новое внешнее соединение, то пул сначала ищет кандидата в списке простаивающих соединений. Поиск основан на 4 параметрах: ---
  ◦ строка подключения;
  ◦ имя пользователя;
  ◦ пароль;
  ◦ роль.
Поиск чувствителен к регистру;
• если подходящее соединение найдено, то проверяется живое ли оно;
• если соединение не прошло проверку, то оно удаляется и поиск повторяется (ошибка не возвращается пользователю);
• найденное (и живое) соединение перемещается из списка простаивающих соединений в список активных соединений и возвращается вызывающему;
• если имеется несколько подходящих соединений, то будет выбрано наиболее часто используемое;
• если нет подходящего соединения, то создаётся новое и помещается в список активных соединений;
• когда время жизни простаивающего соединения истекло, то оно удаляется из пула и закрывается.

Основные характеристики:
• отсутствие “вечных” внешних соединений;
• ограниченное количество неактивных (простаивающих) внешних соединений в пule;
• поддерживает быстрый поиск среди соединений (по 4 параметрам указаным выше);
• пул является общим для всех внешних баз данных;
• пул является общим для всех локальных соединений, обрабатываемых данным
процессом Firebird.

Параметры пула внешних соединений:

- время жизни соединения: временной интервал с момента последнего использования соединения, после истечения которого он будет принудительно закрыт. Параметр ExtConnPoolLifeTime в firebird.conf. По умолчанию равен 7200 секунд;
- размер пула: максимально допустимое количество незанятых соединений в пуле. Параметр ExtConnPoolSize в firebird.conf. По умолчанию равен 0, т.е. пул внешних соединений отключен.

Пулом внешних соединений, а также его параметрами можно управлять с помощью специальных операторов. Подробнее см. ALTER EXTERNAL CONNECTIONS POOL.

Состояние пула внешних подключений можно запросить с использованием контекстных переменных в пространстве имен SYSTEM:

Table 108. Переменные пространства имен SYSTEM для контроля пула внешних соединений

<table>
<thead>
<tr>
<th>Переменная</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT_CONN_POOL_SIZE</td>
<td>Размер пула.</td>
</tr>
<tr>
<td>EXT_CONN_POOL_LIFE_TIME</td>
<td>Время жизни неактивных соединений.</td>
</tr>
<tr>
<td>EXT_CONN_POOL_IDLE_COUNT</td>
<td>Текущее количество неактивных соединений в пуле.</td>
</tr>
<tr>
<td>EXT_CONN_POOL_ACTIVE_COUNT</td>
<td>Текущее количество активных соединений в пуле.</td>
</tr>
</tbody>
</table>

Особенности внешних подключений

1. Внешние соединения используют по умолчанию предложение WITH COMMON TRANSACTION и остаются открытыми до закрытия текущей транзакции. Они могут быть снова использованы при последующих вызовах оператора EXECUTE STATEMENT, но только если строка подключения точно такая же. Если включен пул внешних соединений, то вместо закрытия соединения, такие соединения будут попадать в список неактивных (простаивающих) соединений;

2. Внешние соединения, созданные с использованием предложения WITH AUTONOMOUS TRANSACTION, закрываются после выполнения оператора или попадают в список неактивных соединений пула (если он включен);

3. Операторы WITH AUTONOMOUS TRANSACTION могут использовать соединения, которые ранее были открыты операторами WITH COMMON TRANSACTION. В этом случае использованное соединение остаётся открытым и после выполнения оператора, т.к. у этого соединения есть, по крайней мере, одна не закрытая транзакция. Если включен пул внешних соединений, то вместо закрытия соединения, такие соединения будут попадать в список неактивных (простаивающих) соединений;

4. Если локальная транзакция запущена в режиме изолированности READ COMMITTED READ CONSISTENCY и внешний источник данных не поддерживает данный режим
изолированности, то внешняя транзакция будет запущена в режиме изолированности SNAP SHOT (CONCURRENCY).

Особенности пула транзакций (Transaction pooling)

1. При использовании предложения WITH COMMON TRANSACTION транзакции будут снова использованы как можно дольше. Они будут подтверждаться или откатываться вместе с текущей транзакцией;

2. При использовании предложения WITH AUTONOMOUS TRANSACTION всегда запускается новая транзакция. Она будет подтверждена или отменена сразу же после выполнения оператора;

Особенности обработки исключений

При использовании предложения ON EXTERNAL дополнительное соединение всегда делается через так называемого внешнего провайдера, даже если это соединение к текущей базе данных. Одним из последствий этого является то, что вы не можете обработать исключение привычными способами. Каждое исключение, вызванное оператором, возвращает eds_connection или eds_statement ошибки. Для обработки исключений в коде PSQL вы должны использовать WHEN GDSCODE eds_connection, WHEN GDSCODE eds_statement или WHEN ANY.

Если предложение ON EXTERNAL не используется, то исключения перехватываются в обычном порядке, даже если это дополнительное соединение с текущей базой данных.

Другие замечания

• Набор символов, используемый для внешнего соединения, совпадает с используемым набором для текущего соединения.

• Двухфазные транзакции не поддерживаются.

AS USER, PASSWORD и ROLE

Необязательные предложения AS USER, PASSWORD и ROLE позволяют указывать от имени какого пользователя, и с какой ролью будет выполняться SQL оператор. То, как авторизуется пользователь и открыто ли отдельное соединение, зависит от присутствия и значений параметров ON EXTERNAL [DATA SOURCE], AS USER, PASSWORD и ROLE.

• При использовании предложения ON EXTERNAL открывается новое соединение и:
  ◦ Если присутствует, по крайней мере, один из параметров AS USER, PASSWORD и ROLE, то будет предпринята попытка нативной аутентификации с указанными значениями параметров (в зависимости от строки соединения — локально или удалённо). Для недостающих параметров не используются никаких значений по умолчанию;
  ◦ Если все три параметра отсутствуют, и строка подключения не содержит имени сервера (или IP адреса), то новое соединение устанавливается к локальному серверу с пользователем и ролью текущего соединения. Термин 'локальный' означает 'компьютер, где установлен сервер Firebird'. Это совсем не обязательно компьютер клиента;
Если все три параметра отсутствуют, но строка подключения содержит имя сервера (или IP адреса), то будет предпринята попытка доверенной (trusted) авторизации к удалённому серверу. Если авторизация прошла, то удалённая операционная система назначит пользователю имя — обычно это учётная запись, под которой работает сервер Firebird.

• Если предложение ON EXTERNAL отсутствует:
  ◦ Если присутствует, по крайней мере, один из параметров AS USER, PASSWORD и ROLE, то будет открыто соединение к текущей базе данных с указанными значениями параметров. Для недостающих параметров не используются никаких значений по умолчанию;
  ◦ Если все три параметра отсутствуют, то оператор выполняется в текущем соединении.

Если значение параметра NULL или '', то весь параметр считается отсутствующим. Кроме того, если параметр считается отсутствующим, то AS USER принимает значение CURRENT_USER, а ROLE — CURRENT_ROLE. Сравнение при авторизации сделано чувствительным к регистру: в большинстве случаев это означает, что имена пользователя и роли должны быть написаны в верхнем регистре.

Предосторожения
1. Не существует способа проверить синтаксис выполняемого SQL оператора;
2. Нет никаких проверок зависимостей для обнаружения удалённых столбцов в таблице или самой таблицы;
3. Выполнение оператора с помощью оператора EXECUTE STATEMENT значительно медленнее, чем при непосредственном выполнении;
4. Возвращаемые значения строго проверяются на тип данных во избежание непредсказуемых исключений преобразования типа. Например, строка '1234' преобразуется в целое число 1234, а строка 'abc' вызовет ошибку преобразования.

В целом эта функция должна использоваться очень осторожно, а вышеупомянутые факторы всегда должны приниматься во внимание. Если такого же результата можно достичь с использованием PSQL и/или DSQL, то это всегда предпочтительнее.

См. также: FOR EXECUTE STATEMENT.

7.7.15. FOR SELECT

Назначение
Цикл по строкам результата выполнения оператора SELECT.

Доступно в PSQL
Синтаксис

[label:] FOR <select_stmt> [INTO <variables>] [AS CURSOR cursorname] DO <compound_statement>

<variables> ::= [:{endsb}varname [, [:{endsb}varname ...]

Table 109. Параметры оператора FOR SELECT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Необязательная метка для LEAVE и/или CONTINUE. Должна следовать правилам для идентификаторов.</td>
</tr>
<tr>
<td>select_stmt</td>
<td>Оператор SELECT.</td>
</tr>
<tr>
<td>cursorname</td>
<td>Имя курсора. Должно быть уникальным среди имен переменных и курсоров PSQL модуля.</td>
</tr>
<tr>
<td>varname</td>
<td>Имя локальной переменной или входного/выходного параметра.</td>
</tr>
<tr>
<td>compound_statement</td>
<td>Составной оператор (оператор или блок операторов).</td>
</tr>
</tbody>
</table>

Оператор FOR SELECT выбирает очередную строку из таблицы (представления, селективной хранимой процедуры), после чего выполняется составной оператор. В каждой итерации цикла значения полей текущей строки копируются в локальные переменные. Добавление предложения AS CURSOR делает возможным позиционное удаление и обновление данных. Операторы FOR SELECT могут быть вложенными.

Оператор FOR SELECT может содержать именованные параметры, которые должны быть предварительно объявлены в операторе DECLARE VARIABLE, или во входных (выходных) параметрах процедуры (PSQL блока).

Оператор FOR SELECT должен содержать предложение INTO, которое располагается в конце этого оператора, или предложение AS CURSOR. На каждой итерации цикла в список переменных указанных в предложении INTO копируются значения полей текущей строки запроса. Цикл повторяется, пока не будут прочитаны все строки. После этого происходит выход из цикла. Цикл также может быть завершён до прочтения всех строк при использовании оператора LEAVE.

Необъявленный курсор

Необязательное предложение AS CURSOR создаёт именованный курсор, на который можно ссылаться (с использованием предложения WHERE CURRENT OF) внутри составного оператора следующего после предложения DO, для того чтобы удалить или модифицировать текущую строку.

Разрешается использовать имя курсора как переменную типа запись (аналогично OLD и NEW в триггерах), что позволяет получить доступ к столбцам результирующего набора (т.е.
Использование предложения AS CURSOR делает предложение INTO необязательным. 

Правила для курсорных переменных:

• Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;
• К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;
• Переменные курсора доступны только для чтения;
• В операторе FOR SELECT без предложения AS CURSOR необходимо использовать предложение INTO. Если указано предложение AS CURSOR, предложение INTO не требуется, но разрешено;
• Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор UPDATE (с предложением WHERE CURRENT OF) обновит также и значения полей в переменной курсора для последующих чтений. Выполнение оператора DELETE (с предложением WHERE CURRENT OF) установит NULL для значений полей переменной курсора для последующих чтений.

• Над курсором, объявлённым с помощью предложения AS CURSOR нельзя выполнить операторы OPEN, FETCH и CLOSE;
• Убедитесь, что имя курсора, определённое здесь, не совпадает ни с какими именами, созданными ранее оператором DECLARE VARIABLE;
• Предложение FOR UPDATE, разрешённое для использования в операторе SELECT, не является обязательным для успешного выполнения позиционного обновления или удаления.

Примеры использования FOR SELECT

Example 279. Использование оператора FOR SELECT

```sql
CREATE PROCEDURE SHOWNUMS
RETURNS (  
    AA INTEGER,  
    BB INTEGER,  
    SM INTEGER,  
    DF INTEGER)
AS
BEGIN
    FOR SELECT DISTINCT A, B
    FROM NUMBERS
    ORDER BY A, B
    INTO AA, BB
    DO
    BEGIN
        SM = AA + BB;
```
Example 280. Вложенный FOR SELECT

```sql
CREATE PROCEDURE RELFIELDS
RETURNS (  
    RELATION CHAR(32),  
    POS INTEGER,  
    FIELD CHAR(32))
AS
BEGIN
    FOR SELECT RDB$RELATION_NAME
        FROM RDB$RELATIONS
        ORDER BY 1
        INTO :RELATION
    DO
        BEGIN
            FOR SELECT
                RDB$FIELD_POSITION + 1,
                RDB$FIELD_NAME
            FROM RDB$RELATION_FIELDS
            WHERE
                RDB$RELATION_NAME = :RELATION
            ORDER BY RDB$FIELD_POSITION
            INTO :POS, :FIELD
            DO
                BEGIN
                    IF (POS = 2) THEN
                        RELATION = ' "';
                        -- Для исключения повтора имён таблиц и представлений
                        SUSPEND;
                END
            END
        END
    END
END
```

Example 281. Использование предложения AS CURSOR для позиционного удаления записи

```sql
CREATE PROCEDURE DELTOWN (  
    TOWNTODELETE VARCHAR(24))
RETURNS (  
    TOWN VARCHAR(24),  
    POP INTEGER)
AS
BEGIN
```

Chapter 7. Операторы процедурного SQL (PSQL)
Example 282. Использование неявно объявленного курсора как курсорной переменной

```sql
EXECUTE BLOCK
RETURNS ( o CHAR(63))
AS
BEGIN
  FOR
    SELECT
      rdb$relation_name AS name
    FROM
      rdb$relations AS CURSOR c
  DO
    BEGIN
      o = c.name;
      SUSPEND;
    END
  END
```

Example 283. Разрешение неоднозначностей курсорной переменной внутри запросов

```sql
EXECUTE BLOCK
RETURNS ( o1 CHAR(63),
            o2 CHAR(63))
AS
BEGIN
  FOR
    SELECT
      rdb$relation_name
    FROM
      rdb$relations
```
WHERE
    rdb$relation_name = 'RDB$RELATIONS' AS CURSOR c
DO
BEGIN
FOR
    SELECT
        -- с префиксом разрешается как курсор
        c.rdb$relation_name x1,
        -- без префикса как псевдоним таблицы rdb$relations
        c.rdb$relation_name x2
FROM
    rdb$relations c
WHERE
    rdb$relation_name = 'RDB$DATABASE' AS CURSOR d
DO
BEGIN
    o1 = d.x1;
    o2 = d.x2;
    SUSPEND;
END
END
END

См. также:
SELECT, DECLARE ...CURSOR, OPEN, CLOSE, FETCH.

7.7.16. FOR EXECUTE STATEMENT

Назначение
Выполнение динамически созданных SQL операторов с возвратом нескольких строк данных.

Доступно в
PSQL

Синтаксис

[[label:]
FOR <execute_statement> DO <compound_statement>

Table 110. Параметры оператора FOR EXECUTE STATEMENT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Необязательная метка для LEAVE и/или CONTINUE. Должна соответствовать правилам для идентификаторов.</td>
</tr>
<tr>
<td>execute_statement</td>
<td>Оператор EXECUTE STATEMENT.</td>
</tr>
<tr>
<td>compound_statement</td>
<td>Составной оператор (оператор или блок операторов).</td>
</tr>
</tbody>
</table>
Оператор FOR EXECUTE STATEMENT используется (по аналогии с конструкцией FOR SELECT) для операторов SELECT или EXECUTE BLOCK, возвращающих более одной строки.

Примеры `FOR EXECUTE STATEMENT`

*Example 284. Использование оператора FOR EXECUTE STATEMENT*

```sql
CREATE PROCEDURE DynamicSampleThree (  
    Q_FIELD_NAME VARCHAR(100),  
    Q_TABLE_NAME VARCHAR(100)  
) RETURNS(  
    LINE VARCHAR(32000)  
) AS  
    DECLARE VARIABLE P_ONE_LINE VARCHAR(100);  
BEGIN  
    LINE = '';  
    FOR EXECUTE STATEMENT  
        'SELECT T1."' || :Q_FIELD_NAME || ' FROM " || :Q_TABLE_NAME || ' T1'  
    INTO :P_ONE_LINE  
    DO  
        IF (:P_ONE_LINE IS NOT NULL) THEN  
            LINE = LINE || :P_ONE_LINE || ' ';  
        SUSPEND;  
    END  
END
```

См. также: EXECUTE STATEMENT.

7.7.17. OPEN

Назначение
Открытие курсора.

Доступно в PSQL

Синтаксис

```
OPEN cursor_name;
```

*Table 111. Параметры оператора OPEN*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor_name</td>
<td>Имя курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE ⋯ CURSOR.</td>
</tr>
</tbody>
</table>
Оператор OPEN открывает ранее объявленный курсор, выполняет объявленный в нем оператор SELECT и получает записи из результирующего набора данных. Оператор OPEN применим только к курсорам, объявленным в операторе DECLARE ... CURSOR.

Если в операторе SELECT курсора имеются параметры, то они должны быть объявлены как локальные переменные или входные (выходные) параметры до того как объявлен курсор. При открытии курсора параметру присваивается текущее значение переменной.

Примеры OPEN

Example 285. Использование оператора OPEN

```
SET TERM ^;

CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (  
    RNAME CHAR(31)  
)  
AS  
    DECLARE C CURSOR FOR (  
        SELECT RDB$RELATION_NAME  
        FROM RDB$RELATIONS);  
BEGIN  
    OPEN C;  
    WHILE (1 = 1) DO  
        BEGIN  
            FETCH C INTO :RNAME;  
            IF (ROW_COUNT = 0) THEN  
                LEAVE;  
                SUSPEND;  
            END  
            CLOSE C;  
        END  
    END ;^  
SET TERM ;^  
```

Example 286. Использование оператора OPEN с параметрами

Данный пример возвращает набор скриптов для создания представлений с использованием блока PSQL с именованными курсорами.

```
EXECUTE BLOCK
RETURNS (  
    SCRIPT BLOB SUB_TYPE TEXT)  
AS  
    DECLARE VARIABLE FIELDS VARCHAR(8191);  
```
DECLARE VARIABLE FIELD_NAME TYPE OF RDB$FIELD_NAME;
DECLARE VARIABLE RELATION RDB$RELATION_NAME;
DECLARE VARIABLE SOURCE TYPE OF COLUMN RDB$RELATIONS.RDB$VIEW_SOURCE;
-- именованный курсор
DECLARE VARIABLE CUR_R CURSOR FOR (SELECT RDB$RELATION_NAME, RDB$VIEW_SOURCE FROM RDB$RELATIONS WHERE RDB$VIEW_SOURCE IS NOT NULL);
-- Именованный курсор
DECLARE CUR_F CURSOR FOR (SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS WHERE -- Важно! Переменная должна быть объявлена ранее RDB$RELATION_NAME = :RELATION);
BEGIN
OPEN CUR_R;
WHILE (1 = 1) DO
BEGIN
FETCH CUR_R INTO :RELATION, :SOURCE;
IF (ROW_COUNT = 0) THEN
LEAVE;

FIELDS = NULL;
-- Курсор CUR_F использует -- значение переменной RELATION инициализированной ранее
OPEN CUR_F;
WHILE (1 = 1) DO
BEGIN
FETCH CUR_F INTO :FIELD_NAME;
IF (ROW_COUNT = 0) THEN
LEAVE;
IF (FIELDS IS NULL) THEN
FIELDS = TRIM(FIELD_NAME);
ELSE
FIELDS = FIELDS || ', ' || TRIM(FIELD_NAME);
END
CLOSE CUR_F;

SCRIPT = 'CREATE VIEW ' || RELATION;
IF (FIELDS IS NOT NULL) THEN
SCRIPT = SCRIPT || '(' || FIELDS || ')';
7.7.18. FETCH

Назначение

Чтение записи из набора данных, связанного с курсором.

Доступно в

PSQL

Синтаксис

FETCH [<fetch_scroll> FROM] cursor_name
[INTO [:]varname [, [:]varname ...]];

<fetch_scroll> ::= NEXT | PRIOR | FIRST | LAST
| RELATIVE n
| ABSOLUTE n

Table 112. Параметры оператора FETCH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor_name</td>
<td>Имя курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE ... CURSOR.</td>
</tr>
<tr>
<td>var_name</td>
<td>PSQL переменная.</td>
</tr>
<tr>
<td>n</td>
<td>Целое число.</td>
</tr>
</tbody>
</table>

Оператор FETCH выбирает следующую строку данных из результирующего набора данных курсора и присваивает значения столбцов в переменные PSQL. Оператор FETCH применим только к курсорам, объявленным в операторе DECLARE ... CURSOR.

Оператор FETCH может указывать в каком направлении и на сколько записей продвинется позиция курсора. Предложение NEXT допустимо использовать как с прокручиваемыми, там и не прокручиваемыми курсорами.

Остальные предложения допустимо использовать только с прокручиваемыми курсорами.
Опции прокручиваемого курсора

**NEXT**

перемещает указатель курсора на 1 запись вперёд. Это действие по умолчанию.

**PRIOR**

перемещает указатель курсора на 1 запись назад.

**FIRST**

перемещает указатель курсора на первую запись.

**LAST**

перемещает указатель курсора на последнюю запись.

**ABSOLUTE n**

перемещает указатель курсора на указанную запись; \( n \) — целочисленное выражение, где 1 обозначает первую строку. Для отрицательных значений абсолютная позиция берется с конца набора результатов, поэтому -1 указывает последнюю строку, -2 - предпоследнюю строку и т. д. Нулевое значение (0) будет располагаться перед первой строкой.

**RELATIVE n**

перемещает курсор на \( n \) строк из текущей позиции; положительные числа перемещают указатель вперёд, а отрицательные числа — назад; использование нуля (0) не приведет к перемещению курсора, а ROW_COUNT будет установлено в ноль, поскольку новая строка не была выбрана.

Необязательное предложение INTO помещает данные из текущей строки курсора в PSQL переменные.

Разрешается использовать имя курсора как переменную типа запись (аналогично OLD и NEW в триггерах), что позволяет получить доступ к столбцам результирующего набора (т.е. `cursor_name . columnname`).

Правила использования курсорных переменных

- Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;

- К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;

- Переменные курсора доступны только для чтения;

- Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор UPDATE (с предложением WHERE CURRENT OF) обновит также и значения полей переменной курсора для последующих чтений. Выполнение оператора DELETE (с предложением WHERE CURRENT OF) установит NULL для значений полей переменной курсора для последующих чтений.

Для проверки того, что записи набора данных исчерпаны, используется контекстная переменная ROW_COUNT, которая возвращает количество строк выбранных оператором. Если
произошло чтение очередной записи из набора данных, то ROW_COUNT равняется единице, иначе нулю.

Примеры FETCH

Example 287. Использования оператора FETCH

```
SET TERM ^;

CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (RNAME CHAR(63))
AS

DECLARE C CURSOR FOR (SELECT RDB$RELATION_NAME FROM RDB$RELATIONS);
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
    BEGIN
      FETCH C INTO :RNAME;
      IF (ROW_COUNT = 0) THEN
        LEAVE;
        SUSPEND;
      END
    CLOSE C;
  END
END ^

SET TERM ;^
```

Example 288. Использования оператора FETCH со вложенными курсорами

```
EXECUTE BLOCK
RETURNS (SCRIPT BLOB SUB_TYPE TEXT)
AS

DECLARE VARIABLE FIELDS VARCHAR(8191);
DECLARE VARIABLE FIELD_NAME TYPE OF RDB$FIELD_NAME;
DECLARE VARIABLE RELATION RDB$RELATION_NAME;
DECLARE VARIABLE SRC TYPE OF COLUMN RDB$RELATIONS.RDB$VIEW_SOURCE;
-- Объявление именованного курсора
DECLARE VARIABLE CUR_R CURSOR FOR (SELECT RDB$RELATION_NAME, RDB$VIEW_SOURCE
FROM RDB$RELATIONS
WHERE RDB$VIEW_SOURCE IS NOT NULL);
```
-- Объявление именованного курсора, в котором
-- используется локальная переменная
DECLARE CUR_F   CURSOR FOR (  
    SELECT RDB\$FIELD_NAME  
    FROM RDB\$RELATION_FIELDS  
    WHERE  
      -- Важно переменная должна быть объявлена ранее  
      RDB\$RELATION_NAME = :RELATION);  
BEGIN
    OPEN CUR_R;
    WHILE (1 = 1) DO
      BEGIN
        FETCH CUR_R
        INTO :RELATION, :SRC;
        IF (ROW_COUNT = 0) THEN
          LEAVE;
          FIELDS = NULL;
          -- Курсор CUR_F будет использовать значение
          -- переменной \$RELATION инициализированной выше
          OPEN CUR_F;
          WHILE (1 = 1) DO
            BEGIN
              FETCH CUR_F
              INTO :FIELD_NAME;
              IF (ROW_COUNT = 0) THEN
                LEAVE;
                IF (FIELDS IS NULL) THEN
                  FIELDS = TRIM(FIELD_NAME);
                ELSE
                  FIELDS = FIELDS || ', ' || TRIM(FIELD_NAME);
                END
            END
          CLOSE CUR_F;
          SCRIPT = 'CREATE VIEW ' || RELATION;
          IF (FIELDS IS NOT NULL) THEN
            SCRIPT = SCRIPT || '(' || FIELDS || ')';
          SCRIPT = SCRIPT || ' AS ' || ASCII_CHAR(13);
          SCRIPT = SCRIPT || SRC;
        SUSPEND;
      END
      CLOSE CUR_R;
    END
END
Пример использования оператора FETCH с прокручиваемым курсором

```sql
EXECUTE BLOCK RETURNS (  
    N INT,  
    RNAME CHAR(63)) AS  
DECLARE C SCROLL CURSOR FOR (  
    SELECT  
        ROW_NUMBER() OVER(ORDER BY RDB$RELATION_NAME) AS N,  
        RDB$RELATION_NAME  
    FROM RDB$RELATIONS  
    ORDER BY RDB$RELATION_NAME);  
BEGIN  
    OPEN C;  
    -- перемещаемся на первую запись (N=1)  
    FETCH FIRST FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;  
    -- перемещаемся на 1 запись вперёд (N=2)  
    FETCH NEXT FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;  
    -- перемещаемся на пятую запись (N=5)  
    FETCH ABSOLUTE 5 FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;  
    -- перемещаемся на 1 запись назад (N=4)  
    FETCH PRIOR FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;  
    -- перемещаемся на 3 записи вперёд (N=7)  
    FETCH RELATIVE 3 FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;  
    -- перемещаемся на 5 записей назад (N=2)  
    FETCH RELATIVE -5 FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;  
    -- перемещаемся на первую запись (N=1)  
    FETCH FIRST FROM C;  
    RNAME = C.RDB$RELATION_NAME;  
    N = C.N;  
    SUSPEND;
```
FETCH LAST FROM C;
RNAME = C.RDB$RELATION_NAME;
N = C.N;
SUSPEND;
CLOSE C;
END

См. также:
OPEN, CLOSE, DECLARE ... CURSOR.

7.7.19. CLOSE

Назначение
Закрытие курсора.

Доступно в
PSQL

Синтаксис

CLOSE cursor_name;

Table 113. Параметры оператора CLOSE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor_name</td>
<td>Имя открытого курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE ... CURSOR.</td>
</tr>
</tbody>
</table>

Оператор CLOSE закрывает открытый курсор. Любые все ещё открытые курсоры будут автоматически закрыты после выполнения кода триггера, хранимой процедуры, функции или анонимного PSQL блока, в пределах кода которого он был открыт. Оператор CLOSE применим только к курсорам, объявленным в операторе DECLARE ... CURSOR.

Примеры CLOSE
См. примеры в операторе FETCH.

См. также:
FETCH, OPEN, DECLARE ... CURSOR.

7.7.20. IN AUTONOMOUS TRANSACTION

Назначение
Выполнение составного оператора в автономной транзакции.

Доступно в
Инструкция `IN AUTONOMOUS TRANSACTION DO <compound_statement>`

Table 114. Параметры оператора `IN AUTONOMOUS TRANSACTION`

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>compound_statement</td>
<td>Составной оператор (оператор или блок операторов).</td>
</tr>
</tbody>
</table>

Оператор `IN AUTONOMOUS TRANSACTION` позволяет выполнить составной оператор в автономной транзакции. Код, работающий в автономной транзакции, будет подтверждаться сразу же после успешного завершения независимо от состояния родительской транзакции. Это бывает нужно, когда определённые действия не должны быть отменены, даже в случае возникновения ошибки в родительской транзакции.

Автономная транзакция имеет тот же уровень изоляции, что и родительская транзакция. Любое исключение, вызванное или появившееся в блоке кода автономной транзакции, приведёт к откату автономной транзакции и отмене всех внесённых изменений. Если код будет выполнен успешно, то автономная транзакция будет подтверждена.

Примеры `IN AUTONOMOUS TRANSACTION`

Example 290. Использование автономных транзакций

Данный пример демонстрирует использование автономной транзакции в триггере на событие подключения к базе данных для регистрации всех попыток соединения, в том числе и неудачных.

```sql
CREATE TRIGGER TR_CONNECT ON CONNECT
AS
BEGIN
  -- Все попытки соединения с БД сохраняем в журнал
  IN AUTONOMOUS TRANSACTION DO
    INSERT INTO LOG(MSG)
    VALUES ('USER ' || CURRENT_USER || ' CONNECTS.');
  IF (CURRENT_USER IN (SELECT
                         USERNAME
                        FROM
                        BLOCKED_USERS)) THEN
    BEGIN
      -- Сохраняем в журнал, что попытка соединения
      -- с БД оказалась неудачной
      -- и отправляем сообщение о событии
      IN AUTONOMOUS TRANSACTION DO
        BEGIN
          INSERT INTO LOG(MSG)
          VALUES ('USER ' || CURRENT_USER || ' REFUSED.');
        END
    END
END
```
См. также:
Управление транзакциями.

7.7.21. POST_EVENT

Назначение
Посылка события (сообщения) клиентским приложениям.

Доступно в
PSQL

Синтаксис

```
POST_EVENT event_name;
```

Таблица 115. Параметры оператора POST_EVENT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_name</td>
<td>Имя события, ограничено 127 байтами.</td>
</tr>
</tbody>
</table>

Оператор POST_EVENT сообщает о событии менеджеру событий, который сохраняет его в таблице событий. При подтверждении транзакции менеджер событий информирует приложения, ожидающие это событие.

Имя события это своего рода код или короткое сообщение, выбор за вами, т.к. это просто строка длинной до 127 байт.

В качестве имени события может быть использован строковый литерал, переменная или любое правильное SQL выражение.

Примеры POST_EVENT

`Example 291. Оповещение приложения о вставке записи в таблицу SALES`

```
SET TERM ^;
CREATE TRIGGER POST_NEW_ORDER FOR SALES
ACTIVE AFTER INSERT POSITION 0
AS
BEGIN
  POST_EVENT 'new_order';
```

7.7.22. RETURN

Назначение
Возврат значения из хранимой функции

Доступно в
PSQL

Синтаксис

```
RETURN value;
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Выражение для возврата значения из функции; Может быть любым выражением, совместимым с типом возвращаемого значения функции.</td>
</tr>
</tbody>
</table>

Оператор RETURN завершает выполнение функции и возвращает значение выражения value.

RETURN может использоваться только в PSQL функциях (хранимых и локальных функциях).

Примеры RETURN

См. Примеры CREATE FUNCTION

7.7.23. Обработка ошибок

В Firebird существуют PSQL операторы для обработки ошибок и исключений в модулях. Существует множество встроенных исключений, которые возникают в случае возникновения стандартных ошибок при работе с DML и DDL операторами.

Системные исключения

Исключение представляет собой сообщение, которое генерируется, когда возникает ошибка.

Все обрабатываемые Firebird исключения имеют заранее определённые числовые (символьные) значение для контекстных переменных и связанные с ними тексты сообщений. Сообщения об ошибке написаны по умолчанию на английском языке. Существуют и локализованные сборки СУБД, в которых сообщения об ошибках переведены на другие языки.

Полный список системных исключений вы можете найти в приложении "Обработка ошибок, коды и сообщения":

---

457
• Коды ошибок SQLSTATE и их описание
• "Коды ошибок GDSCODE их описание, и SQLCODE"

Пользовательские исключения

Пользовательские исключения могут быть объявлены в базе данных как постоянные объекты и вызваны из PSQL кода для сообщения об ошибке при нарушении некоторых бизнес правил. Текст пользовательского исключения ограничен 1021 байтом. Подробности см. CREATE EXCEPTION.

В коде PSQL исключения обрабатываются при помощи оператора WHEN. Если исключение будет обработано в вашем коде, то вы обеспечите исправление или обход ошибки и позволите продолжить выполнение, то клиенту не возвращается никакого сообщения об исключении.

Исключение приводит к прекращению выполнения в блоке. Вместо того чтобы передать выполнение на конечный оператор END, теперь процедура отыскивает уровни во вложенных блоках, начиная с блока где была вызвана ошибка, и переходит на внешние блоки, чтобы найти код обработчика, который “знает” о таком исключении. Она отыскивает первый оператор WHEN, который может обработать эту ошибку.

EXCEPTION

Назначение

Возбуждение пользовательского исключения или повторный вызов исключения.

Доступно в

PSQL

Синтаксис

```
EXCEPTION [ exception_name [ custom_message | USING (<value_list>)] ]
<value_list> ::= <val> [, <val> ...]
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception_name</td>
<td>Имя исключения.</td>
</tr>
<tr>
<td>custom_message</td>
<td>Альтернативный текст сообщения, выдаваемый при возникновении исключения. Максимальная длина текстового сообщения составляет 1021 байт.</td>
</tr>
<tr>
<td>val</td>
<td>Значения, которыми заменяются слоты в тексте сообщения исключения.</td>
</tr>
</tbody>
</table>
Оператор EXCEPTION возбуждает пользовательское исключение с указанным именем. При возбуждении исключения можно также указать альтернативный текст сообщения, который заменит текст сообщения заданным при создании исключения.

Текст сообщения исключения может содержать слоты для параметров, которые заполняются при возбуждении исключения. Для передачи значений параметров в исключение используется предложение USING. Параметры рассматриваются слева направо. Каждый параметр передаётся в оператор возбуждающий исключение как “N-ый”, N начинается с 1:

- Если N-ый параметр не передан, его слот не заменяется;
- Если передано значение NULL, слот будет заменён на строку “*** null ***”;
- Если количество передаваемых параметров будет больше, чем содержится в сообщении исключения, то лишние будут пропущены;
- Максимальный номер параметра равен 9;
- Общая длина сообщения, включая значения параметров, ограничена 1053 байтами.

Статус вектор генерируется, используя комбинацию кодов isc_except, <exception number>, isc_formatted_exception, <formatted exception message>, <exception parameters>.

Поскольку используется новый код ошибки (isc_formatted_exception), клиент должен быть версии 3.0 или по крайней мере использовать firebird.msg от версии 3.0 для того чтобы правильно преобразовать статус вектор в строку.

Если в тексте сообщения, встретится номер слота параметра больше 9, то второй и последующий символ будут восприняты как литералы. Например, @10 будет воспринято как @1 после которого следует литерал ‘0’.

CREATE EXCEPTION ex1
'something wrong in @1 @2 @3 @4 @5 @6 @7 @8 @9 @10 @11';

EXECUTE BLOCK AS
BEGIN
EXCEPTION ex1 USING ('a','b','c','d','e','f','g','h','i');
END;

Statement failed, SQLSTATE = HY000
exception 1
-EX1
-something wrong in a b c d e f g h i a0 a1

Исключение может быть обработано в операторе WHEN ... DO. Если пользовательское
исключение не было обработано в триггере или в хранимой процедуре, то действия, выполненные внутри этой хранимой процедуры (триггера) отменяются, а вызвавшая программа получает текст, заданный при создании исключения или альтернативный текст сообщения.

В блоке обработки исключений (и только в нем), вы можете повторно вызвать пойманное исключение или ошибку, вызывая оператор EXCEPTION без параметров. Вне блока с исключением такой вызов не имеет никакого эффекта.

Пользовательские исключения хранятся в системной таблице RDB$EXCEPTIONS.

Примеры EXCEPTION

Example 292. Вызов исключения

```sql
CREATE OR ALTER PROCEDURE SHIP_ORDER (PO_NUM CHAR(8))
AS
DECLARE VARIABLE ord_stat CHAR(7);
DECLARE VARIABLE hold_stat CHAR(1);
DECLARE VARIABLE cust_no INTEGER;
DECLARE VARIABLE any_po CHAR(8);
BEGIN
    SELECT
        s.order_status,
        c.on_hold,
        c.cust_no
    FROM
        sales s, customer c
    WHERE
        po_number = :po_num AND
        s.cust_no = c.cust_no
    INTO :
        ord_stat,
        hold_stat,
        cust_no;

/* Этот заказ уже отправлен на поставку. */
    IF (ord_stat = 'shipped') THEN
        EXCEPTION order_already_shipped;
/* Другие операторы */
END
```

Example 293. Вызов исключения с заменой исходного сообщения альтернативным

```sql
CREATE OR ALTER PROCEDURE SHIP_ORDER (PO_NUM CHAR(8))
AS
```
DECLARE VARIABLE ord_stat CHAR(7);
DECLARE VARIABLE hold_stat CHAR(1);
DECLARE VARIABLE cust_no INTEGER;
DECLARE VARIABLE any_po CHAR(8);
BEGIN
SELECT
    s.order_status,
    c.on_hold,
    c.cust_no
FROM
    sales s, customer c
WHERE
    po_number = :po_num AND
    s.cust_no = c.cust_no
INTO
    :ord_stat,
    :hold_stat,
    :cust_no;

/* Этот заказ уже отправлен на поставку. */
IF (ord_stat = 'shipped') THEN
    EXCEPTION order_already_shipped 'Order status is '' | ord_stat | ''';
/* Другие операторы */
END

Example 294. Использование параметризованного исключения

CREATE EXCEPTION EX_BAD_SP_NAME
    'Name of procedures must start with ''@1'' : ''@2''';
...
CREATE TRIGGER TRG_SP_CREATE BEFORE CREATE PROCEDURE
AS
    DECLARE SP_NAME VARCHAR(255);
BEGIN
    SP_NAME = RDB$GET_CONTEXT('DDL_TRIGGER', 'OBJECT_NAME');
    IF (SP_NAME NOT STARTING 'SP_') THEN
        EXCEPTION EX_BAD_SP_NAME USING ('SP_', SP_NAME);
    END

См. также:
CREATE EXCEPTION, WHEN ... DO.

WHEN ... DO

Назначение
Обработка ошибок.

Доступно в
Оператор `WHEN … DO` используется для обработки ошибочных ситуаций и пользовательских исключений. Оператор перехватывает все ошибки и пользовательские исключения, перечисленные после ключевого слова `WHEN`. Если после ключевого слова `WHEN` указано ключевое слово `ANY`, то оператор перехватывает любые ошибки и пользовательские исключения, даже если они уже были обработаны в вышестоящем `WHEN` блоке.

Оператор `WHEN … DO` должен находиться в самом конце блока операторов перед оператором `END`.

После ключевого слова `DO` следует составной оператор, в котором можно произвести обработку ошибки или исключения. Составной оператор — это оператор или блок операторов, заключённый в операторные скобки `BEGIN` и `END`. В этом операторе доступны контекстные переменные `GDSCODE, SQLCODE, SQLSTATE`. Для получения имени активного пользовательского исключения или текста интерпретированного сообщения об ошибке вы можете воспользоваться системной функцией `RDB$ERROR`. В этом же блоке доступен оператор повторного вызова ошибки или исключительной ситуации `EXCEPTION` (без параметров).

После предложения `WHEN GDSCODE` вы должны использовать символьные имена — такие, как `grant_obj_notfound` и т.д. Но в составном операторе, после ключевого слова `DO` доступна контекстная переменная `GDSCODE`, которая содержит целое число. Для сравнения его с определённой ошибкой вы должны использовать числовое значение, например, 335544551 для `grant_obj_notfound`. 
Оператор WHEN ... DO вызывается только в том случае, если произошло одно из указанных в его условии событий. В случае выполнения оператора (даже если в нем фактически не было выполнено никаких действий) ошибка или пользовательское исключение не прерывает и не отменяет действий триггера или хранимой процедуры, где был выдан этот оператор, работа продолжается, как если бы никаких исключительных ситуаций не было. Однако в этом случае будет отменено действие DML оператора (SELECT, INSERT, UPDATE, DELETE, MERGE), который вызвал ошибку и все ниже находящиеся операторы в том же блоке операторов не будут выполнены.

Если ошибка вызвана не одним из DML операторов (SELECT, INSERT, UPDATE, DELETE, MERGE), то будет отменен не только оператор вызвавший ошибку, а весь блок операторов. Кроме того, действия в операторе WHEN ... DO так же будут откачены. Это относится также и к оператору выполнения хранимой процедуры EXECUTE PROCEDURE. Подробнее смотри в CORE-4483.

Область действия оператора WHEN ... DO

Оператор перехватывает ошибки и исключения в текущем блоке операторов. Он также перехватывает подобные ситуации во вложенных блоках, если эти ситуации не были в них обработаны.

Оператор WHEN ... DO видит все изменения, произведённые до оператора вызвавшего ошибку. Однако если вы попытаетесь запротоколировать их в автономной транзакции, то эти изменения будут не доступны, поскольку на момент старта автономной транзакции, транзакция, в которой произошли эти изменения, не подтверждена.

Примеры использования WHEN...DO

Example 295. Замена стандартной ошибки своей.

```
CREATE EXCEPTION COUNTRY_EXIST ' ';
SET TERM ^;
CREATE PROCEDURE ADD_COUNTRY ( 
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) ) 
AS BEGIN
    INSERT INTO country (country, currency) VALUES (:ACountryName, :ACurrency);
    WHEN SQLCODE -803 DO 
        EXCEPTION COUNTRY_EXIST 'Такая страна уже добавлена!';
END^ 
SET TERM ^;
```
Example 296. Registration of the error in the journal and its repetition in the WHEN block.

```sql
CREATE PROCEDURE ADD_COUNTRY (  
    ACountryName COUNTRYNAME,  
    ACurrency VARCHAR(10) )  
AS  
BEGIN  
    INSERT INTO country (country,  
        currency)  
    VALUES (:ACountryName,  
        :ACurrency);  
    WHEN ANY DO  
    BEGIN  
        -- Write the error to the journal  
        IN AUTONOMOUS TRANSACTION DO  
        INSERT INTO ERROR_LOG (PSQL_MODULE,  
            ERROR_TEXT,  
            EXCEPTION_NAME,  
            GDS_CODE,  
            SQL_CODE,  
            SQL_STATE)  
        VALUES ('ADD_COUNTRY',  
            RDB$ERROR(MESSAGE), -- Error message  
            RDB$ERROR(EXCEPTION), -- Name of the user-defined exception  
            GDSCODE,  
            SQLCODE,  
            SQLSTATE)  
        );  
        -- Repeat the error  
        EXCEPTION;  
    END  
END  
```

Example 297. Handling multiple errors in one WHEN ... DO block.

```sql
...  
WHEN GDSCODE GRANT_OBJ_NOTFOUND,  
    GDSCODE GRANT_FLD_NOTFOUND,  
    GDSCODE GRANT_NOPRIV,  
    GDSCODE GRANT_NOPRIV_ON_BASE  
DO  
BEGIN  
    EXECUTE PROCEDURE LOG_GRANT_ERROR(GDSCODE);  
    EXIT;  
END  
...  
```
Example 298. Перехват ошибок по коду SQLSTATE.

```sql
EXECUTE BLOCK AS
  DECLARE VARIABLE I INT;
BEGIN
  BEGIN
    I = 1 / 0;
    WHEN SQLSTATE '22003' DO
      EXCEPTION E_CUSTOM_EXCEPTION
        'Numeric value out of range.';
    WHEN SQLSTATE '22012' DO
      EXCEPTION E_CUSTOM_EXCEPTION 'Division by zero.';
    WHEN SQLSTATE '23000' DO
      EXCEPTION E_CUSTOM_EXCEPTION
        'Integrity constraint violation.';
  END
END
```

См. также:
Оператор EXCEPTION, Коды ошибок SQLSTATE и их описание, Коды ошибок GDSCODE и SQLCODE и их описание.
Chapter 8. Встроенные скалярные функции

8.1. Функции для работы с контекстными переменными

8.1.1. RDB$GET_CONTEXT()

Доступно в
DSQL, PSQL

Синтаксис

```
RDB$GET_CONTEXT('<namespace>', 'varname')
```

Таблица 119. Параметры функции RDB$GET_CONTEXT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Пространство имён.</td>
</tr>
<tr>
<td>varname</td>
<td>Имя переменной. Зависит от регистра. Максимальная длина 80 байт.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARCHAR(255) CHARACTER SET NONE

Функция RDB$GET_CONTEXT возвращает значение контекстной переменной из одного из пространства имён.

В настоящий момент существуют следующие пространства имён:

- **SYSTEM** — предоставляет доступ к системным контекстным переменным. Эти переменные доступны только для чтения;
- **USER_SESSION** — предоставляет доступ к пользовательским контекстным переменным, заданным через функцию RDB$SET_CONTEXT. Переменные существуют в течение подключения;
- **USER_TRANSACTION** — предоставляет доступ к пользовательским контекстным переменным, заданным через функцию RDB$SET_CONTEXT. Переменные существуют в течение транзакции;
- **DDL_TRIGGER** — предоставляет доступ к системным контекстным переменным, доступным только во время выполнения DDL триггера. Эти переменные доступны только для
Пространства имён USER_SESSION и USER_TRANSACTION — изначально пусты и пользователь сам создаёт переменные и наполняет их при помощи функции RDB$SET_CONTEXT.

Для предотвращения DoS атак, существует ограничение на 1000 переменных в одном "пространстве имён".

Если запрашиваемая функцией переменная существует в указанном пространстве имён, то будет возвращено её значение в виде строки VARCHAR(255) CHARACTER SET NONE. При обращении к несуществующей переменной в пространстве SYSTEM возникает ошибка, если такое происходит с пространствами имён USER_SESSION или USER_TRANSACTION — функция возвращает NULL.

Пространство имён SYSTEM

Переменные пространства имён SYSTEM

CLIENT_ADDRESS
Адрес клиента. Для TCP – IP адрес, для XNET – локальный ID процесса. Для остальных случаев NULL.

CLIENT_HOST
Имя хоста сетевого протокола удаленного клиента. Значение возвращается для всех поддерживаляемых протоколов.

CLIENT_OS_USER
Имя пользователя операционной системы на клиентском компьютере.

CLIENT_PID
PID процесса на клиентском компьютере.

CLIENT_PROCESS
Полный путь к клиентскому приложению, подключающемуся к базе данных.

CLIENT_VERSION
Версия клиентской библиотеки (fbclient), используемой клиентским приложением.

CURRENT_ROLE
Глобальная переменная CURRENT_ROLE.

CURRENT_USER
Глобальная переменная CURRENT_USER.

DB_NAME
Каноническое имя текущей базы данных. Это либо имя псевдонима (если соединение с помощью имени файлов запрещено DatabaseAccess = NONE) или, в противном случае, полностью расширенное имя файла базы данных.

Chapter 8. Встроенные скалярные функции
**DB_FILE_ID**
Уникальный идентификатор текущей базы данных на уровне файловой системы.

**DB_GUID**
GUID базы данных.

**EFFECTIVE_USER**
Эффективный пользователь в текущий момент. Указывает пользователя с привилегиями которого в текущий момент времени выполняется процедура, функция или триггер.

**ENGINE_VERSION**
Версия сервера Firebird.

**EXT_CONN_POOL_SIZE**
Размер пула внешних соединений.

**EXT_CONN_POOL_LIFETIME**
Время жизни неактивных соединений в пуле внешних соединений.

**EXT_CONN_POOL_IDLE_COUNT**
Текущее количество неактивных соединений в пуле внешних соединений.

**EXT_CONN_POOL_ACTIVE_COUNT**
Текущее количество активных соединений в пуле внешних соединений.

**GLOBAL_CN**
Последнее значение текущего глобального счётчика Commit Number

**ISOLATION_LEVEL**
Уровень изоляции текущей транзакции — CURRENT_TRANSACTION. Значения: 'READ_Committed', 'SNAPSHOT' или 'CONSISTENCY'.

**LOCK_TIMEOUT**
Время ожидания транзакцией высвобождения ресурса при блокировке, в секундах.

**NETWORK_PROTOCOL**
Протокол, используемый для соединения с базой данных. Возможные значения: 'TCPv4', 'TCPv6', 'WNET', 'XNET', NULL.

**READ_ONLY**
Отображает, является ли транзакция, транзакцией только для чтения. 'FALSE' для Read-Write транзакций 'TRUE' для Read Only.

**REPLICA_MODE**
Режим репликации: пустая строка или NULL — первичная база данных, 'READ-ONLY' — реплика в режиме только чтение, 'READ-WRITE' — реплика в режиме чтение и запись.
REPLICATION_SEQUENCE
Текущее значение последовательности репликации (номер последнего сегмента, записанного в журнал репликации).

SESSION_ID
Глобальная переменная CURRENT_CONNECTION.

SESSION_IDLE_TIMEOUT
Содержит текущее значение тайм-аута простого соединения в секундах, который был установлен на уровне соединения, или ноль, если тайм-аут не был установлен.

SESSION_TIMEZONE
Текущий часовой пояс, установленный в текущей сессии.

SNAPSHOT_NUMBER
Номер моментального снимка базы данных: уровня транзакции (для транзакции SNAPSHOT или CONSISTENCY) или уровня запроса (для транзакции READ COMMITTED READ CONSISTENCY). NULL, если моментальный снимок не существует.

STATEMENT_TIMEOUT
Содержит текущее значение тайм-аута выполнения оператора в миллисекундах, который был установлен на уровне подключения, или ноль, если тайм-аут не был установлен.

TRANSACTION_ID
Глобальная переменная CURRENT_TRANSACTION.

WIRE_COMPRESSED
Используется ли сжатие сетевого трафика. Если используется сжатие сетевого трафика возвращает 'TRUE', если не используется — 'FALSE'. Для встроенных соединений — возвращает NULL.

WIRE_ENCRYPTED
Используется ли шифрование сетевого трафика. Если используется шифрование сетевого трафика возвращает 'TRUE', если не используется — 'FALSE'. Для встроенных соединений — возвращает NULL.

WIRE_CRYPT_PLUGIN
Если используется шифрование сетевого трафика, то возвращает имя текущего плагина шифрования, в противном случае NULL.

**Пространство имён DDL_TRIGGER**

Использование пространства имён DDL_TRIGGER допустимо, только во время работы DDL тритгер. Его использование также допустимо в хранимых процедурах и функциях, вызванных тритгерами DDL.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триттега, значения,
относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триттера значения выталкиваются. Таким образом в случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL тритгер, и этот тритгер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имён DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL тритгер в стеке вызовов.

Переменные пространства имён DDL_TRIGGER

EVENT_TYPE
тип события (CREATE, ALTER, DROP).

OBJECT_TYPE
тип объекта (TABLE, VIEW и др.).

DDL_EVENT
(<ddl event item>), где <ddl_event_item> это EVENT_TYPE || ' ' || OBJECT_TYPE

OBJECT_NAME
имя объекта метаданных.

OLD_OBJECT_NAME
имя объекта метаданных до переименования.

NEW_OBJECT_NAME
имя объекта метаданных после переименования.

SQL_TEXT
текст SQL запроса.

Ещё раз обратите внимание на то, что пространства имен и имен переменных регистрочувствительны, должны быть не пустыми строками, и заключены в кавычки!

Примеры

Example 299. Использование функции RDB$GET_CONTEXT

NEW.USERADR = RDB$GET_CONTEXT ('SYSTEM', 'CLIENT_ADDRESS');

См. также:
RDB$SET_CONTEXT.

8.1.2. RDB$SET_CONTEXT()

Доступно в
DSQL, PSQL
Синтаксис

RDB$SET_CONTEXT('<namespace>', 'varname', {<value> | NULL})

<namespace> ::= USER_SESSION | USER_TRANSACTION

Table 120. Параметры функции RDB$SET_CONTEXT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Пространство имён.</td>
</tr>
<tr>
<td>varname</td>
<td>Имя переменной. Зависит от регистра. Максимальная длина 80 байт.</td>
</tr>
<tr>
<td>value</td>
<td>Данные любого типа при условии, что их можно привести к типу VARCHAR(255) CHARACTER SET NONE.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

INTEGER

Функция RDB$SET_CONTEXT создаёт, устанавливает значение или обнуляет переменную в одном из используемых пользователем пространстве имён: USER_SESSION или USER_TRANSACTION.

Функция возвращает 1, если переменная уже существовала до вызова и 0, если не существовала. Для удаления переменной надо установить её значение в NULL. Если данное пространство имён не существует, то функция вернёт ошибку. Пространство имён и имя переменной зависят от регистра, должны быть не пустые строки, и заключены в кавычки.

- Пространство имён SYSTEM доступно только для чтения;
- Максимальное число переменных в рамках одного соединения (для пространства USER_SESSION) или одной транзакции (для пространства USER_TRANSACTION) равно 1000;
- Все переменные в пространстве имён USER_TRANSACTION сохраняются при ROLLBACK RETAIN или ROLLBACK TO SAVEPOINT, независимо от того, в какой точке во время выполнения транзакции они были установлены.

Example 300. Использование функции RDB$SET_CONTEXT

```
SELECT RDB$SET_CONTEXT ('USER_SESSION', 'DEBUGL', 3) 
FROM RDB$DATABASE;

-- в PSQL доступен такой синтаксис
RDB$SET_CONTEXT('USER_SESSION', 'RECORDSFLOWN', RECCOUNTER);

SELECT RDB$SETCONTEXT ('USER_TRANSACTION', 'SAVEPOINTS', 'YES')
FROM RDB$DATABASE;
```
Example 301. Использование функций для работы с контекстными переменными

```sql
SET TERM ^;
CREATE PROCEDURE set_context(User_ID VARCHAR(40),
    Trn_ID INT) AS
    BEGIN
        RDB$SET_CONTEXT('USER_TRANSACTION', 'Trn_ID', Trn_ID);
        RDB$SET_CONTEXT('USER_TRANSACTION', 'User_ID', User_ID);
    END^;
SET TERM ;^;

CREATE TABLE journal (jrn_id INTEGER NOT NULL PRIMARY KEY,
    jrn_lastuser VARCHAR(40),
    jrn_lastaddr VARCHAR(255),
    jrn_lasttran INTEGER);

SET TERM ^;
CREATE TRIGGER UI_JOURNAL FOR JOURNAL BEFORE INSERT OR UPDATE
AS BEGIN
    new.jrn_lastuser = RDB$GET_CONTEXT('USER_TRANSACTION', 'User_ID');
    new.jrn_lastaddr = RDB$GET_CONTEXT('SYSTEM', 'CLIENT_ADDRESS');
    new.jrn_lasttran = RDB$GET_CONTEXT('USER_TRANSACTION', 'Trn_ID');
END^;
SET TERM ;^;

EXECUTE PROCEDURE set_context('skidder', 1);

INSERT INTO journal(jrn_id) VALUES(0);

COMMIT;
```

См. также:
RDB$GET_CONTEXT.

8.2. Математические функции

8.2.1. ABS()

Доступно в
DSQL, PSQL
### Table 121. Параметры функции ABS

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа</td>
</tr>
</tbody>
</table>

*Тип возвращаемого результата:* тот же что и входной аргумент.

Функция ABS возвращает абсолютное значение (модуль) аргумента.

### 8.2.2. ACOS()

*Доступно в* DSQL, PSQL

#### Синтаксис

ACOS (number)

*Тип возвращаемого результата:* DOUBLE PRECISION

Функция ACOS возвращает арккосинус (в радианах) аргумента.

В случае если аргумент функции вне границы диапазона [-1, 1], то функция вернёт неопределённое значение NaN.

*См. также:* COS().

### 8.2.3. ACOSH()

*Доступно в* DSQL, PSQL

#### Синтаксис

ACOSH (number)
### 8.2.4. ASIN()

Доступно в
DSQL, PSQL

**Синтаксис**

```
ASIN (number)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа в диапазоне [-1; 1].</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
DOUBLE PRECISION

Функция ASIN возвращает арксинус (в радианах) аргумента.

В случае если аргумент функции вне границы диапазона [-1, 1], то функция вернёт неопределённое значение NaN.

**См. также:**
SIN().

### 8.2.5. ASINH()

Доступно в
DSQL, PSQL

**Синтаксис**

```
ASIN (number)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа в диапазоне [-1; +∞].</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
DOUBLE PRECISION

Функция ASIN возвращает гиперболический арккосинус (в радианах) аргумента.

**См. также:**
COSH().

---

**Table 123. Параметры функции ACOSH**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа в диапазоне [1; +∞].</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
DOUBLE PRECISION

Функция ACOSH возвращает гиперболический арккосинус (в радианах) аргумента.

**См. также:**
COSH().
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
DOUBLE PRECISION

Функция ASINH возвращает гиперболический арксинус (в радианах) аргумента.

**См. также:**
SINH().

**8.2.6. ATAN()**

Доступно в
DSQL, PSQL

**Синтаксис**

```
ATAN (number)
```

**Table 126. Параметры функции ATAN**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

**Typ возвращаемого результата:**
DOUBLE PRECISION

Функция ATAN возвращает арктангенс аргумента.

Функция возвращает угол в радианах в диапазоне \([-\pi/2; \pi/2]\).

**См. также:**
ATAN2(), TAN().

**8.2.7. ATAN2()**

Доступно в
DSQL, PSQL

**Синтаксис**

```
ATAN2 (y, x)
```

**Table 127. Параметры функции ATAN2**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>у</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>
### Table 128. Параметры функции ATANH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

**Typ возвращаемого результата:**

DOUBLE PRECISION

Функция ATAN2 возвращает угол как отношение синуса к косинусу, аргументы, у которых задаются этими двумя параметрами, а знаки синуса и косинуса соответствуют знакам параметров. Это позволяет получать результаты по всей окружности, включая углы -π/2 и π/2.

Особенности использования:

- Результат — угол в диапазоне [-π, π] радиан;
- Если x отрицательный, то при нулевом значении y результат равен π, а при значении 0 равен -π;
- Если и y и x равны 0, то результат бессмыслен.

- Полностью эквивалентное описание этой функции следующее: ATAN2 (y, x) является углом между положительной осью X и линией от начала координат до точки (x, y). Это также делает очевидным, что значение ATAN2 (0, 0) не определено;
- Если x больше, чем 0, ATAN2 (y, x) совпадает с ATAN (y/x);
- Если известны и синус, и косинус угла, то ATAN2 (sin, cos) возвращает угол.

См. также:

ATAN(), SIN(), COS().

### 8.2.8. ATANH()

Доступно в

DNL, PSQL

Синтаксис

```
ATANH (number)
```

**Typ возвращаемого результата:**

DOUBLE PRECISION
Функция ATANH возвращает гиперболический арктангенс (в радианах) аргумента.

*См. также:* TANH().

### 8.2.9. CEIL(), CEILING()

**Доступно в:** DSQL, PSQL

**Синтаксис**

```plaintext
CEIL[ING] (number)
```

*Таблица 129. Параметры функции CEIL[ING]*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

*Тип возвращаемого результата:* BIGINT, INT128, DECFLOAT или DOUBLE PRECISION в зависимости от типа аргумента.

Функция CEIL возвращает наименьшее целое число, большее или равное аргументу.

*См. также:* FLOOR(), TRUNC().

### 8.2.10. COS()

**Доступно в:** DSQL, PSQL

**Синтаксис**

```plaintext
COS (angle)
```

*Таблица 130. Параметры функции COS*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>Угол, выраженный в радианах.</td>
</tr>
</tbody>
</table>

*Тип возвращаемого результата:* DOUBLE PRECISION

Функция COS возвращает косинус угла. Аргумент должен быть задан в радианах.

Любой NOT NULL результат находится в диапазоне [-1, 1].

*См. также:*
8.2.11. COSH()

Доступно в
DSQL, PSQL

Синтаксис

COSH (number)

Таблица 131. Параметры функции COSH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция COSH возвращает гиперболический косинус аргумента.

Любой NOT NULL результат находится в диапазоне [1, +∞].

См. также:
ACOSH().

8.2.12. COT()

Доступно в
DSQL, PSQL

Синтаксис

COT (angle)

Таблица 132. Параметры функции COT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>Угол, выраженный в радианах.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция COT возвращает котангенс угла. Аргумент должен быть задан в радианах.

См. также:
TAN().
8.2.13. EXP()

Доступно в
DSQL, PSQL

Синтаксис

\[
\text{EXP (number)}
\]

Таблица 133. Параметры функции EXP

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция EXP возвращает значение натуральной экспоненты, \( e^{\text{number}} \)

См. также:
LN().

8.2.14. FLOOR()

Доступно в
DSQL, PSQL

Синтаксис

\[
\text{FLOOR (number)}
\]

Таблица 134. Параметры функции FLOOR

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
BIGINT, INT128, DECFLOAT или DOUBLE PRECISION в зависимости от типа аргумента.

Функция FLOOR возвращает целое число, меньшее или равное аргументу.

См. также:
CEIL(), CEILING(), TRUNC().

8.2.15. LN()

Доступно в
DSQL, PSQL
Синтаксис

\[ \text{LN (number)} \]

**Таблица 135. Параметры функции LN**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

*Тип возвращаемого результата:*  
DOUBLE PRECISION

Функция LN возвращает натуральный логарифм аргумента.

В случае если передан отрицательный или нулевой аргумент функция вернёт ошибку.

*См. также:*  
\[ \text{EXP()} \]

### 8.2.16. LOG()

**Доступно в:**  
DSQL, PSQL

Синтаксис

\[ \text{LOG (x, y)} \]

**Таблица 136. Параметры функции LOG**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Основание. Выражение числового типа.</td>
</tr>
<tr>
<td>y</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

*Тип возвращаемого результата:*  
DOUBLE PRECISION

Функция LOG возвращает логарифм y (второй аргумент) по основанию x (первый аргумент).

**Особенности использования:**

- Если один из аргументов меньше или равен 0, то возникает ошибка;
- Если оба аргумента равны 1, то результатом функции будет NaN (Not-a-Number — не число);
- Если \( x = 1 \) и \( y < 1 \), то результатом функции будет \(-\infty\);
- Если \( x = 1 \) и \( y > 1 \), то результатом функции будет \(+\infty\).
**8.2.17. LOG10()**

**Доступно в**
DSQL, PSQL

**Синтаксис**

\[
\text{LOG10} (\text{number})
\]

**Table 137. Параметры функции LOG10**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
DOUBLE PRECISION

Функция LOG10 возвращает десятичный логарифм аргумента.

Если входной аргумент отрицательный или равен 0, возникает ошибка.

**8.2.18. MOD()**

**Доступно в**
DSQL, PSQL

**Синтаксис**

\[
\text{MOD} (a, b)
\]

**Table 138. Параметры функции MOD**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Выражение числового типа.</td>
</tr>
<tr>
<td>b</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
INTEGER, BIGINT или INT128 в зависимости от типов аргументов.

Функция MOD возвращает остаток от целочисленного деления.

Вещественные числа округляются до выполнения деления. Например, результатом “mod(7.5, 2.5)” будет 2 (“mod(8, 3)”), а не 0.

**8.2.19. PI()**

**Доступно в**
8.2.20. POWER()

Доступно в
DSQL, PSQL

Синтаксис

```
POWER (x, y)
```

**Table 139. Параметры функции POWER**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Выражение числового типа.</td>
</tr>
<tr>
<td>y</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

**Typ возвращаемого результата:**
DOUBLE PRECISION

Функция POWER возвращает результат возведения числа x в степень y то есть \(x^y\).

ℹ️ Если x меньше нуля, возникает ошибка.

8.2.21. RAND()

Доступно в
DSQL, PSQL

Синтаксис

```
RAND ()
```

**Typ возвращаемого результата:**
DOUBLE PRECISION

Функция RAND возвращает псевдослучайное число в интервале от 0 до 1.
8.2.22. ROUND()

Доступно в
DSQL, PSQL

Синтаксис

\[
\text{ROUND (number [, scale])}
\]

Table 140. Параметры функции ROUND

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
<tr>
<td>scale</td>
<td>Масштаб — целое число, определяющее число десятичных разрядов, к которым должен быть проведено округление, т.e.</td>
</tr>
<tr>
<td>• 2 для округления к самому близкому кратному 0.01 числу</td>
<td></td>
</tr>
<tr>
<td>• 1 для округления к самому близкому кратному 0.1 числу</td>
<td></td>
</tr>
<tr>
<td>• 0 для округления к самому близкому целому числу</td>
<td></td>
</tr>
<tr>
<td>• -1 для округления к самому близкому кратному 10 числу</td>
<td></td>
</tr>
<tr>
<td>• -2 для округления к самому близкому кратному 100 числу</td>
<td></td>
</tr>
</tbody>
</table>

По умолчанию 0.

Тип возвращаемого результата
масштабируемое целое (INTEGER, BIGINT или INT128) или DECFLOAT, или DOUBLE PRECISION в зависимости от типа number.

Функция ROUND округляет число до ближайшего целого числа. Если дробная часть равна 0.5, то округление до ближайшего большего целого числа для положительных чисел и до ближайшего меньшего для отрицательных чисел. С дополнительным опциональным параметром scale число может быть округлено до одной из степеней числа 10 (десятки, сотни, десятые части, сотые части и т.д.) вместо просто целого числа.

Если используется параметр scale, то результат имеет такой же масштаб, как и первый параметр number.

Примеры ROUND

Example 302. Использование функции ROUND

\[
\text{ROUND(123.654, 1)} -- \text{Результат: 123.700 (а не 123.7)}
\]
\[
\text{ROUND(8341.7, -3)} -- \text{Результат: 8000.0 (а не 8000)}
\]
\[
\text{ROUND(45.1212, 0)} -- \text{Результат: 45.0000 (а не 45)}
\]
\[
\text{ROUND(45.1212)} -- \text{Результат: 45}
\]
8.2.23. SIGN()

Доступно в
DSQL, PSQL

Синтаксис

SIGN (number)

Таблица 141. Параметры функции SIGN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
SMALLINT

Функция SIGN возвращает знак входного параметра.

-1 — число меньше нуля
0 — число равно нулю
1 — число больше нуля

8.2.24. SIN()

Доступно в
DSQL, PSQL

Синтаксис

SIN (angle)

Таблица 142. Параметры функции SIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>Угол, выраженный в радианах.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция SIN возвращает синус угла. Аргумент должен быть задан в радианах.

Любой NOT NULL результат находится в диапазоне [-1, 1].

См. также:
8.2.25. SINH()

Доступно в
DSQL, PSQL

Синтаксис

\[
\text{SINH (number)}
\]

Table 143. Параметры функции SINH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция SINH возвращает гиперболический синус аргумента.

См. также:
ASINH().

8.2.26. SQRT()

Доступно в
DSQL, PSQL

Синтаксис

\[
\text{SQRT (number)}
\]

Table 144. Параметры функции SQRT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция SQRT возвращает квадратный корень аргумента.

8.2.27. TAN()

Доступно в
DSQL, PSQL
Синтаксис

TAN (angle)

Таблица 145. Параметры функции TAN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>Угол, выраженный в радианах.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция TAN возвращает тангенс угла. Аргумент должен быть задан в радианах.

См. также:
ATAN(), ATAN2().

8.2.28. TANH()

Доступно в
DSQL, PSQL

Синтаксис

TANH (number)

Таблица 146. Параметры функции TANH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
DOUBLE PRECISION

Функция TANH возвращает гиперболический тангенс аргумента.

Любой NOT NULL результат находится в диапазоне [-1, 1].

См. также:
ATANH().

8.2.29. TRUNC()

Доступно в
DSQL, PSQL
Синтаксис

TRUNC (number [, scale])

Table 147. Параметры функции TRUNC

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Выражение числового типа.</td>
</tr>
<tr>
<td>scale</td>
<td>Масштаб — целое число, определяющее число десятичных разрядов, к которым должен быть проведено усечение, т.е.</td>
</tr>
</tbody>
</table>

- 2 для усечения к самому близкому кратному 0.01 числу
- 1 для усечения к самому близкому кратному 0.1 числу
- 0 для усечения к самому близкому целому числу
- -1 для усечения к самому близкому кратному 10 числу
- -2 для усечения к самому близкому кратному 100 числу

По умолчанию 0.

Тип возвращаемого результата

масштабируемое целое (INTEGER, BIGINT или INT128) или DECFLOAT, или DOUBLE PRECISION в зависимости от типа number.

Функция TRUNC усекает число до ближайшего целого числа. С дополнительным опциональным параметром scale число может быть усечено до одной из степеней числа 10 (десятки, сотни, десятые части, сотые части и т.д.) вместо просто целого числа.

Если используется параметр scale, то результат имеет такой же масштаб, как и первый параметр number.

Функция всегда увеличивает отрицательные числа, поскольку она обрезает дробную часть.

Example 303. Использование функции TRUNC

<table>
<thead>
<tr>
<th>TRUNC(789.2225, 2)</th>
<th>-- Результат: 789.2200 (а не 789.22)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUNC(345.4, -2)</td>
<td>-- Результат: 300.0 (а не 300)</td>
</tr>
<tr>
<td>TRUNC(-163.41, 0)</td>
<td>-- Результат: -163.00 (а не -163)</td>
</tr>
<tr>
<td>TRUNC(-163.41)</td>
<td>-- Результат: -163</td>
</tr>
</tbody>
</table>

См. также:

ROUND(), CEIL(), CEILING(), FLOOR().
### 8.3. Функции для работы со строками

#### 8.3.1. ASCII_CHAR()

**Доступно в**
DSQL, PSQL

**Синтаксис**

```
ASCII_CHAR (code)
```

**Таблица 148. Параметры функции ASCII_CHAR**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Целое число в диапазоне от 0 до 255.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
CHAR(1) CHARACTER SET NONE.

Функция ASCII_CHAR возвращает ASCII символ соответствующий номеру, переданному в качестве аргумента.

**См. также:**
ASCII_VAL().

#### 8.3.2. ASCII_VAL()

**Доступно в**
DSQL, PSQL

**Синтаксис**

```
ASCII_VAL (ch)
```

**Таблица 149. Параметры функции ASCII_VAL**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>Строка типа данных [VAR]CHAR или текстовый BLOB максимального размера 32767 байт.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
SMALLINT

Функция ASCII_VAL возвращает ASCII код символа, переданного в качестве аргумента.

Особенности использования:

- Если строка содержит более одного символа, то возвращается код первого символа
строки;
• Если строка пустая, возвращается ноль;
• Если аргумент NULL, то возвращаемое значение также NULL.

См. также:
ASCII_CHAR().

8.3.3. BASE64_DECODE()

Доступно в
DSQL, PSQL

Синтаксис

```
BASE64_DECODE (base64_data)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64_data</td>
<td>Данные в кодировке Base64, дополненные знаком = до длины кратной 4</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
BLOB или VARBINARY

BASE64_DECODE декодирует строку с данными закодированными алгоритмом base64 и возвращает декодированное значение как VARBINARY или BLOB в зависимости от входного аргумента.

Если длина типа `base64_data` не кратна 4, то во время подготовки возникает ошибка. Если длина значения `base64_data` не кратна 4, то во время выполнения возникает ошибка.

Когда входной аргумент не является BLOB, то длина результирующего типа вычисляется как `type_length * 3/4`, где `type_length` — максимальная длина в байтах типа входного аргумента.

Примеры BASE64_DECODE

```sql
Example 304. Использование BASE64_DECODE

```
```sql
select cast(base64_decode('VGVzdCBiYXNlNjQ=' as varchar(12))
   from rdb$database;
```

CAST

```
==========
Test base64
```
См. также:
BASE64_ENCODE().

8.3.4. BASE64_ENCODE()

Доступно в
DSQL, PSQL

Синтаксис

BASE64_ENCODE (binary_data)

Таблица 151. Параметры функции BASE64_ENCODE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary_data</td>
<td>Двоичные данные для кодирования</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

VARCHAR CHARACTER SET ASCII или BLOB SUB_TYPE TEXT CHARACTER SET ASCII

Функция BASE64_ENCODE кодирует binary_data с помощью алгоритма base64 и возвращает закодированное значение как VARCHAR CHARACTER SET ASCII или BLOB SUB_TYPE TEXT CHARACTER SET ASCII в зависимости от типа входного аргумента. Возвращаемое значение дополняется знаком ‘=’, чтобы его длина была кратна 4.

Когда входной аргумент не является BLOB, длина результирующего типа вычисляется как type_length * 4 / 3 с округлением в большую сторону до числа, кратного четырем, где type_length — максимальная длина входного типа в байтах.

Примеры BASE64_ENCODE

Example 305. Использование функции BASE64_ENCODE

```
select base64_encode('Test base64')
from rdb$database;
```

BASE64_ENCODE

================

VGVzdCBiYXNlNjQ=

См. также:
BASE64_DECODE(), HEX_ENCODE().
8.3.5. BIT_LENGTH()

Доступно в
DSQL, PSQL

Синтаксис

```
BIT_LENGTH (string)
```

Таблица 152. Параметры функции BIT_LENGTH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
BIGINT

Функция BIT_LENGTH возвращает длину входной строки в битах. Для многобайтовых наборов символов результат может быть в 8 раз больше, чем количество символов в “формальном” числе байт на символ, записанном в RDB$CHARACTER_SETS.

С параметрами типа CHAR эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если вы хотите получить “логическую” длину в битах, не считая пробелов, то перед передачей аргумента в BIT_LENGTH надо выполнить над ним операцию RIGHT TRIM.

Примеры BIT_LENGTH

```
Example 306. Использование функции BIT_LENGTH

SELECT BIT_LENGTH ('Hello!') FROM RDB$DATABASE
-- возвращает 48

SELECT BIT_LENGTH (_ISO8859_1 'Grüß Di!')
FROM RDB$DATABASE
-- возвращает 64: каждый, и ü, и ß занимают один байт в ISO8859_1

SELECT BIT_LENGTH (CAST (_ISO8859_1 'Grüß di!' AS VARCHAR (24) CHARACTER SET UTF8))
FROM RDB$DATABASE
-- возвращает 80: каждый, и ü, и ß занимают по два байта в UTF8

SELECT BIT_LENGTH (CAST (_ISO8859_1 'Grüß di!' AS CHAR (24) CHARACTER SET UTF8))
FROM RDB$DATABASE
-- возвращает 208: размер всех 24 позиций CHAR и два из них 16-битные
```
8.3.6. CHAR_LENGTH(), CHARACTER_LENGTH()

Доступно в
DSQL, PSQL

Синтаксис

```sql
CHAR_LENGTH (string)
| CHARACTER_LENGTH (string)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
BIGINT

Функция CHAR_LENGTH возвращает длину (в символах) строки, переданной в качестве аргумента.

С параметрами типа CHAR эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если вы хотите получить “логическую” длину без учёта пробелов, то перед передачей аргумента в CHAR[ACTER]_LENGTH надо выполнить над ним операцию RIGHT TRIM.

См. также:
BIT_LENGTH(), OCTET_LENGTH().

8.3.7. HASH()

Доступно в
DSQL, PSQL

Синтаксис

```sql
HASH (str [USING <algorithm>])

<algorithm> ::= { CRC32 }
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Выражение строкового типа</td>
</tr>
</tbody>
</table>
Тип возвращаемого результата:
BIGINT

Функция HASH возвращает некриптографический хэш входной строки. Эта функция полностью поддерживает текстовые BLOB любой длины и с любым набором символов.

Необязательное предложение USING определяет применяемый некриптографический алгоритм хеширования. Когда предложение USING отсутствует, применяется устаревший алгоритм PJW; это идентично его поведению в предыдущих версиях Firebird.

Поддерживаемые алгоритмы:

не указан

Если алгоритм не указан, то используется 64-битный вариант некриптографической хэш-функции PJW (также известной как ELF64). Эта функция очень быстра и может использоваться для общих целей (хеш- таблицы и т. д.), но имеет большое количество коллизий. Для более надежного хеширования следует использовать другие хеш-функции, явно указанные в предложении USING, или криптографические хеши с помощью CRYPT_HASH().

Для этого алгоритма хеширования функция возвращает тип BIGINT.

CRC32

Если в предложении USING указан алгоритм CRC32, то Firebird применяет алгоритм CRC32, используя полином 0x04C11DB7.

Для этого алгоритма функция HASH возвращает результат с типом INTEGER.

Примеры HASH

Example 307. Вычисление хэша с алгоритмом PJW

```sql
SELECT HASH(x) FROM MyTable;
-- результат типа BIGINT
```

Example 308. Вычисление хэша с алгоритмом CRC32

```sql
SELECT HASH(x USING CRC32) FROM MyTable;
-- результат типа INTEGER
```

См. также: CRYPT_HASH()

8.3.8. HEX_DECODE()
Синтаксис

HEX_DECODE (hex_data)

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>hex_data</td>
<td>Данные в шестнадцатеричном представлении.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

VARBINARY или BLOB

Функция HEX_DECODE декодирует строку с шестнадцатеричными данными и возвращает декодированное значение как VARBINARY или BLOB в зависимости от типа входного и размера аргумента. Если длина типа hex_data не кратна 2, во время подготовки возникает ошибка. Если длина значения hex_data не кратна 2, во время выполнения возникает ошибка.

Когда входной аргумент не является BLOB, то длина результирующего типа вычисляется как type_length / 2, где type_length — максимальная длина в байтах типа входного аргумента.

**Примеры HEX_DECODE**

**Example 309. Использование функции HEX_DECODE**

```sql
select cast(hex_decode('48657861646563696D616C' as varchar(12))
from rdb$database;
```

CAST

==========
Hexadecimal

См. также:

HEX_ENCODE(), BASE64_DECODE().

**8.3.9. HEX_ENCODE()**

Доступно в

DSQL, PSQL

Синтаксис

HEX_ENCODE (binary_data)

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary_data</td>
<td>Данные в двоичном представлении.</td>
</tr>
</tbody>
</table>
Параметр | Описание
---|---
binary_data | Двоичные данные для кодирования

**Тип возвращаемого результата:**
VARCHAR CHARACTER SET ASCII или BLOB SUB_TYPE TEXT CHARACTER SET ASCII

Функция HEX_ENCODE кодирует binary_data шестнадцатеричным числом и возвращает закодированное значение как VARCHAR CHARACTER SET ASCII или BLOB SUB_TYPE TEXT CHARACTER SET ASCII в зависимости от входного аргумента.

Когда входной аргумент не является BLOB, то длина результирующего типа вычисляется как type_length * 2, где type_length — максимальная длина в байтах типа входного аргумента.

**Примеры HEX_ENCODE**

**Example 310. Использование функции HEX_ENCODE**

```sql
select hex_encode('Hexadecimal')
from rdb$database;
```

**HEX_ENCODE**
```
======================
48657861646563696D616C
```

**См. также:**
HEX_DECODE(), BASE64_ENCODE()

### 8.3.10. LEFT()

**Доступно в**
DSQL, PSQL

**Синтаксис**

```sql
LEFT (string, length)
```

**Table 157. Параметры функции LEFT**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа.</td>
</tr>
<tr>
<td>length</td>
<td>Целое число. Определяет количество возвращаемых символов.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
VARCHAR или BLOB.
Функция `LEFT` возвращает левую часть строки, количество возвращаемых символов определяется вторым параметром.

Особенности использования:

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если строковый аргумент `BLOB`, результатом будет `BLOB`, в противном случае результатом будет `VARCHAR(N)`, при этом N — будет равно длине строкового параметра;
- Если числовой параметр превысит длину текста, результатом будет исходный текст.

При использовании `BLOB` в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах `BLOB` могут наблюдаться потери производительности.

**Example 311. Использование функции LEFT**

```sql
SELECT LEFT('ABC', 2) FROM rdb$database;
```

**См. также:**

`RIGHT()`, `SUBSTRING()`.

**8.3.11. LOWER()**

Доступно в

DSQL, PSQL, ESQL

Синтаксис

```sql
LOWER (string)
```

**Table 158. Параметры функции LOWER**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

`VAR[CHAR]` или `BLOB`

Функция `LOWER` возвращает входную строку в нижнем регистре. Точный результат зависит от набора символов входной строки. Например, для наборов символов `NONE` и `ASCII` только ASCII символы переводятся в нижний регистр; для `OCTETS` — вся входная строка возвращается без изменений.
Примеры LOWER

Example 312. Использование функции LOWER

select Sheriff from Towns where lower(Name) = 'cooper''s valley'

См. также:
UPPER().

8.3.12. LPAD()

Доступно в
DSQL, PSQL

Синтаксис

LPAD (str, endlen [, padstr])

Table 159. Параметры функции LPAD

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Выражение строкового типа.</td>
</tr>
<tr>
<td>endlen</td>
<td>Длина выходной строки.</td>
</tr>
<tr>
<td>padstr</td>
<td>Строка, которой дополняется исходная строка до указанной длины. По умолчанию является пробелом (&quot; &quot;).</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARCHAR или BLOB.

Функция LPAD дополняет слева входную строку пробелами или определённой пользователем строкой до заданной длины.

Особенности использования:

• Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
• Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(endlen).
• Если аргумент padstr задан, но равен '' (пустой строке), то дополнения строки не происходит! В случае если endlen меньше длины входной строки, то в результате происходит её усечение до длины endlen, даже если параметр padstr равен пустой строке.

⚠ При использовании BLOB в параметрах функции может потребоваться
загрузить объект полностью в память. При больших объёмах BLOB могут наблюдатьсь потери производительности.

Примеры LPAD

Example 313. Использование функции LPAD

```sql
| LPAD ('Hello', 12) | -- возвращает '       Hello' |
| LPAD ('Hello', 12, '-') | -- возвращает '-------Hello' |
| LPAD ('Hello', 12, '') | -- возвращает 'Hello' |
| LPAD ('Hello', 12, 'abc') | -- возвращает 'abcabcaHello' |
| LPAD ('Hello', 12, 'abcdefg hij') | -- возвращает 'abcdefghello' |
| LPAD ('Hello', 2) | -- возвращает 'He' |
| LPAD ('Hello', 2, '-') | -- возвращает 'He' |
| LPAD ('Hello', 2, '') | -- возвращает 'He' |
```

См. также:

RPAD().

8.3.13. OCTET_LENGTH()

Доступно в

DSQL, PSQL

Синтаксис

```sql
OCTET_LENGTH (string)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:

BIGINT

Функция OCTET_LENGTH возвращает количество байт занимаемое строкой.

При работе с параметрами типа CHAR функция возвращает значение всей формальной строковой длины. Для того чтобы узнать “логическую” длину строки в байтах, то перед передачей аргумента функции следует применить RIGHT TRIM.

Следует помнить, что не во всех наборах символов количество байт занимаемых строкой равно количеству символов.
Примеры OCTET_LENGTH

Example 314. Использование функции OCTET_LENGTH

```sql
SELECT OCTET_LENGTH('Hello!')
FROM rdb$database
-- возвратит 6

SELECT OCTET_LENGTH(_.iso8859_1 'Grüß di!')
FROM rdb$database
-- возвратит 8: ü и ß занимают не более 1 байта в ISO8859_1

SELECT
  OCTET_LENGTH(CAST(_.iso8859_1 'Grüß di!' AS VARCHAR(24) CHARACTER SET utf8))
FROM rdb$database
-- возвратит 10: ü и ß занимают 2 байта в UTF8

SELECT
  OCTET_LENGTH(CAST(_.iso8859_1 'Grüß di!' AS CHAR(24) CHARACTER SET utf8))
FROM rdb$database
-- возвратит 26: всего 24 CHAR позиции, и две из них занимают 2 байта
```

См. также:

BIT_LENGTH(), CHAR_LENGTH(), CHARACTER_LENGTH().

8.3.14. OVERLAY()

Доступно в

DSQL, PSQL

Синтаксис

OVERLAY (string PLACING replacement FROM pos [FOR length])

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Строка, в которой происходит замена.</td>
</tr>
<tr>
<td>replacement</td>
<td>Строка, которой заменяется.</td>
</tr>
<tr>
<td>pos</td>
<td>Позиция, с которой происходит замена.</td>
</tr>
<tr>
<td>length</td>
<td>Количество символов, которые будут удалены из исходной строки.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:

VARCHAR или BLOB

Функция OVERLAY предназначена для замены части строки другой строкой.
По умолчанию число удаляемых из строки символов равняется длине заменяемой строки. Дополнительный четвёртый параметр позволяет пользователю задать своё число символов, которые будут удалены.

Особенности использования:

• Функция полностью поддерживает тестовые BLOB с любым набором символов и любой длины;
• Если входная строка имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае тип результата будет VARCHAR(n), где n является суммой длин параметров string и replacement;
• Как и во всех строковых функциях SQL параметр pos является определяющим;
• Если pos больше длины строки, то replacement помещается сразу после окончания строки;
• Если число символов от pos до конца строки меньше, чем длина replacement (или, чем параметр length, если он задан), то строка усекается до значения pos и replacement помещается после него;
• При нулевом параметре length (FOR 0) replacement просто вставляется в строку, начиная с позиции pos;
• Если любой из параметров имеет значение NULL, то и результат будет NULL;
• Если параметры pos и length не являются целым числом, то используется банковское округление (до чётного): 0.5 становится 0, 1.5 становится 2, 2.5 становится 2, 3.5 становится 4 и т.д.

При использовании BLOB функции может потребоваться загрузить весь объект в память. При больших размерах BLOB это может повлиять на производительность.

Примеры OVERLAY

Example 315. Использование функции OVERLAY

| OVERLAY ('Goodbye' PLACING 'Hello' FROM 2) -- Результат: 'Ghelloe' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 5) -- Результат: 'GoodHello' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 8) -- Результат: 'GoodbyeHello' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 20) -- Результат: 'GoodbyeHello' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 0) -- Результат: 'GHellooodbye' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 3) -- Результат: 'GHelloe' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 6) -- Результат: 'GHello' |
| OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 9) -- Результат: 'GHello' |
| OVERLAY ('Goodbye' PLACING '' FROM 4) -- Результат: 'Goodbye' |
| OVERLAY ('Goodbye' PLACING '' FROM 4 FOR 3) -- Результат: 'Gooe' |
| OVERLAY ('Goodbye' PLACING '' FROM 4 FOR 20) -- Результат: 'Goo' |
| OVERLAY ('' PLACING 'Hello' FROM 4) -- Результат: 'Hello' |
| OVERLAY ('' PLACING 'Hello' FROM 4 FOR 0) -- Результат: 'Hello' |
Оверлей:'' PLACING 'Hello' FROM 4 FOR 20) -- Результат: 'Hello'

См. также:
SUBSTRING(), REPLACE.

8.3.15. POSITION()

Доступно в
DSQL, PSQL

Синтаксис

POSITION (substr IN string)
| POSITION (substr, string [, startpos])

Таблица 162. Параметры функции POSITION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>substr</td>
<td>Подстрока, позиция которой ищется.</td>
</tr>
<tr>
<td>string</td>
<td>Строка, в которой ищется позиция.</td>
</tr>
<tr>
<td>startpos</td>
<td>Позиция, с которой начинается поиск подстроки.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
INTEGER

Функция POSITION возвращает позицию первого вхождения подстроки в строку. Отсчет
начинается с 1. Третий аргумент (опциональный) задает позицию в строке, с которой
начинается поиск подстроки, тем самым игнорируя любые вхождения подстроки в строку
до этой позиции. Если совпадение не найдено, функция возвращает 0.

Особенности использования:

- Опциональный третий параметр поддерживается только вторым вариантом синтаксиса
  (синтаксис с запятой);
- Пустую строку, функция считает подстрокой любой строки. Поэтому при входном
  параметре substr, равном '' (пустая строка), и при параметре string, отличном от NULL,
  результатом будет:
  - 1, если параметр startpos не задан;
  - startpos, если startpos не превышает длину параметра string;
  - 0, если startpos больше длины параметра string.

Примеры POSITION
Использование функции `POSITION`

```
POSITION ('be' IN 'To be or not to be') -- Результат: 4
POSITION ('be', 'To be or not to be') -- Результат: 4
POSITION ('be', 'To be or not to be', 4) -- Результат: 4
POSITION ('be', 'To be or not to be', 8) -- Результат: 17
POSITION ('be', 'To be or not to be', 18) -- Результат: 0
POSITION ('be' in 'Alas, poor Yorick!') -- Результат: 0
```

См. также:
`SUBSTRING()`.

### 8.3.16. REPLACE()

Доступно в
DSQL, PSQL

**Синтаксис**

```
REPLACE (str, find, repl)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Строка, в которой делается замена.</td>
</tr>
<tr>
<td>find</td>
<td>Строка, которая ищется.</td>
</tr>
<tr>
<td>repl</td>
<td>Строка, на которую происходит замена.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**
VARCHAR или BLOB

Функция REPLACE заменяет в строке все вхождения одной строки на другую строку.

**Особенности использования:**

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если один из аргументов имеет тип BLOB, то результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(1), где N рассчитывается из длин str, find и repl таким образом, что даже максимальное количество замен не будет вызывать переполнения поля.
- Если параметр find является пустой строкой, то возвращается str без изменений;
- Если параметр repl является пустой строкой, то все вхождения find удаляются из строки str;
- Если любой из аргументов равен NULL, то результатом всегда будет NULL, даже если не
было произведено ни одной замены.

При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры REPLACE

**Example 317. Использование функции REPLACE**

<table>
<thead>
<tr>
<th>Функция</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE ('Billy Wilder', 'il', 'oog')</td>
<td>-- возвращает 'Boogly Woogder'</td>
</tr>
<tr>
<td>REPLACE ('Billy Wilder', 'il', '')</td>
<td>-- возвращает 'Billy Wilder'</td>
</tr>
<tr>
<td>REPLACE ('Billy Wilder', null, 'oog')</td>
<td>-- возвращает NULL</td>
</tr>
<tr>
<td>REPLACE ('Billy Wilder', 'il', null)</td>
<td>-- возвращает NULL</td>
</tr>
<tr>
<td>REPLACE ('Billy Wilder', 'xyz', null)</td>
<td>-- возвращает NULL (!)</td>
</tr>
<tr>
<td>REPLACE ('Billy Wilder', 'xyz', 'abc')</td>
<td>-- возвращает 'Billy Wilder'</td>
</tr>
<tr>
<td>REPLACE ('Billy Wilder', '', 'abc')</td>
<td>-- возвращает 'Billy Wilder'</td>
</tr>
</tbody>
</table>

См. также:
OVERLAY().

**8.3.17. REVERSE()**

Доступно в
DSQL, PSQL

Синтаксис

REVERSE (string)

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARCHAR

Функция REVERSE возвратит строку перевёрнутую "задом наперёд".

Примеры REVERSE

**Example 318. Использование функции REVERSE**

<table>
<thead>
<tr>
<th>Функция</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>REVERSE ('spoonful')</td>
<td>-- возвращает 'lufnoops'</td>
</tr>
<tr>
<td>REVERSE ('Was it a cat I saw?')</td>
<td>-- возвращает '?was I tac a ti saW'</td>
</tr>
</tbody>
</table>
Данная функция очень удобна, если вам предстоит обработать (сортировать или группировать) информацию, которая находится в конце строки. Пример такой информации — доменные имена или имена адресов электронной почты.

```sql
CREATE INDEX ix_people_email ON people
COMPUTED BY (reverse(email));

SELECT * FROM people
WHERE REVERSE(email) STARTING WITH reverse('.br');
```

### 8.3.18. RIGHT()

**Доступно в**

DSQL, PSQL

**Синтаксис**

```sql
RIGHT (string, length)
```

**Table 165. Параметры функции RIGHT**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Выражение строкового типа.</td>
</tr>
<tr>
<td>length</td>
<td>Целое число. Определяет количество возвращаемых символов.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

VARCHAR или BLOB

Функция RIGHT возвращает конечную (правую) часть входной строки. Длина возвращаемой подстроки определяется вторым параметром.

**Особенности использования:**

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если строковый аргумент BLOB, результатом будет BLOB, в противном случае результатом будет VARCHAR(N), при этом N — будет равно длине строкового параметра;
- Если числовой параметр превысит длину текста, результатом будет исходный текст.

При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.
Example 319. Использование функции RIGHT

```
SELECT RIGHT('ABC', 1) FROM rdb$database;
-- результат С
```

См. также:
LEFT(), SUBSTRING().

8.3.19. RPAD()

Доступно в
DSQL, PSQL

Синтаксис

RPAD (str, endlen [, padstr])

Table 166. Параметры функции RPAD

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Выражение строкового типа.</td>
</tr>
<tr>
<td>endlen</td>
<td>Длина выходной строки.</td>
</tr>
<tr>
<td>padstr</td>
<td>Строка, которой дополняется исходная строка до указанной длины. По умолчанию является пробелом (' ').</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARCHAR или BLOB

Функция RPAD дополняет справа входную строку пробелами или определённой пользователем строкой до заданной длины.

Особенности использования:

• Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
• Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(endlen).
• Если аргумент padstr задан, но равен '' (пустой строке), то дополнения строки не происходит! В случае если endlen меньше длины входной строки, то в результате происходит её усечение до длины endlen, даже если параметр padstr равен пустой строке.

При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.
Примеры RPAD

Example 320. Использование функции RPAD

```sql
RPAD ('Hello', 12) -- возвращает 'Hello       '
RPAD ('Hello', 12, '-') -- возвращает 'Hello-------'
RPAD ('Hello', 12, '') -- возвращает 'Helloabcabca'
RPAD ('Hello', 12, 'abcdefgij') -- возвращает 'Helloabcdefg'
RPAD ('Hello', 2) -- возвращает 'He'
RPAD ('Hello', 2, '-') -- возвращает 'He'
RPAD ('Hello', 2, '') -- возвращает 'He'
```

См. также:

LPAD().

8.3.20. SUBSTRING()

Доступно в

DSQL, PSQL

Синтаксис

```sql
SUBSTRING (<substring-args>)

<substring-args> ::=  
    str FROM startpos [FOR length] 
  | str SIMILAR <similar_pattern> ESCAPE <escape>

<similar-pattern> ::=  
    <similar-pattern-R1> 
  | <escape>"<similar pattern_R2><escape>"
  | <similar pattern-R3>
```

Table 167. Параметры функции SUBSTRING

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Выражение строкового типа.</td>
</tr>
<tr>
<td>startpos</td>
<td>Позиция, с которой начинается извлечение подстроки. Целочисленное выражение.</td>
</tr>
<tr>
<td>length</td>
<td>Длина возвращаемой подстроки. Целочисленное выражение.</td>
</tr>
<tr>
<td>similar-pattern</td>
<td>Шаблон регулярного выражения SQL, по которому ищется подстрока.</td>
</tr>
<tr>
<td>escape</td>
<td>Символ экранирования.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARCHAR или BLOB

Функция `SUBSTRING` возвращает подстроку из строки, начиная с заданной позиции до конца строки или до указанной длины, либо извлекает подстроку с использованием шаблона регулярного выражения SQL.

Если любой из входных параметров имеет значение `NULL`, то и результат тоже будет иметь значение `NULL`.

При использовании BLOB в параметрах функции может потребоваться загрузить объект в память полностью. При больших объёмах BLOB могут наблюдаться потери производительности.

Позиционный `SUBSTRING`

В простой позиционной форме (c FROM) эта функция возвращает подстроку, начинающуюся с позиции символа `startpos` (позиция первого символа равна 1). Без аргумента FOR он возвращает все оставшиеся символы в строке. С использованием FOR возвращается `length` символов или остаток строки, в зависимости от того что короче.

Начиная с Firebird 4.0, `startpos` может быть меньше 1. Когда `startpos` меньше 1, подстрока ведет себя так, как если бы строка имела дополнительные позиции 1 - `startpos` перед фактическим первым символом в позиции 1. Значение `length` считается от этого воображаемого начала строки, поэтому результирующая строка может быть короче указанной `length` или даже пустой.

Функция полностью поддерживает двоичные и текстовые BLOB любой длины и с любым набором символов. Если параметр `str` имеет тип BLOB, то и результат будет иметь тип BLOB. Для любых других типов результатом будет тип VARCHAR.

Для входного параметра `str`, не являющегося BLOB, длина результата функции всегда будет равна длине строки `str`, независимо от значений параметров `startpos` и `length`.

**Example 321. Использование функции SUBSTRING**

```sql
select substring('abcdef' from 1 for 2) from rdb$database;
-- результат: 'ab'

select substring('abcdef' from 2) from rdb$database;
-- результат: 'bcdef'

select substring('abcdef' from 0 for 2) from rdb$database;
-- результат: 'a'
-- не 'ab', потому что в позиции 0 нет "ничего"

select substring('abcdef' from -5 for 2) from rdb$database;
-- результат: ''
-- длина заканчивается до фактического начала строки
```
**SUBSTRING по регулярному выражению**

Функция SUBSTRING с регулярным выражением (с SIMILAR) возвращает часть строки соответствующей шаблону регулярного выражения SQL. Если соответствия не найдено, то возвращается NULL.

Шаблон SIMILAR формируется из трёх шаблонов регулярных выражений SQL: R1, R2 и R3. Полностью шаблон имеет форму R1 || '<escape>' || R2 || '<escape>' || R3, где <escape> — это escape-символ, определенный в предложении ESCAPE. R2 — это шаблон, который соответствует подстроке для извлечения и заключен в экранированные двойные кавычки (<escape>"', например, “"’ с escape-символом ‘#’). R1 соответствует префикс строки, а R3 — суффиксу строки. И R1, и R3 необязательны (они могут быть пустыми), но шаблон должен соответствовать всей строке. Другими словами, недостаточно указать шаблон, который находит только подстроку для извлечения.

Экранированные двойные кавычки вокруг R2 можно сравнить с определением одной группы захвата в более распространенном синтаксисе регулярных выражений, таком как PCRE. То есть полный шаблон эквивалентен R1(R2)R3, который должен соответствовать всей входной строке, а группа захвата — это возвращаемая подстрока.

Возвращаемое значение соответствует части R2 регулярного выражения. Для этого значения истинно выражение

```
str SIMILAR TO R1 || R2 || R3 ESCAPE <escape>
```

Если любая часть шаблона из R1, R2 или R3 не является пустой строкой и не имеет формата регулярного выражения SQL, возникает исключение.

Полный формат регулярных выражений SQL описан в Синтаксис регулярных выражений SQL.

**Example 322. Использование функции SUBSTRING с регулярными выражениями**

```
SUBSTRING('abcabc' SIMILAR 'a#"bcab"c' ESCAPE '#') -- bcab
SUBSTRING('abcabc' SIMILAR 'a#%"c' ESCAPE '#') -- bcab
SUBSTRING('abcabc' SIMILAR '_#%"_' ESCAPE '#') -- bcab
SUBSTRING('abcabc' SIMILAR '#{(abc)*#}' ESCAPE '#') -- abcabc
SUBSTRING('abcabc' SIMILAR '#"abc#"' ESCAPE '#') -- <null>
```

См. также:
POSITION(), LEFT(), RIGHT(), CHAR_LENGTH(), CHARACTER_LENGTH(), SIMILAR TO.
8.3.21. TRIM()

Доступно в
DSQL, PSQL

Синтаксис

```
TRIM ([<adjust>] str)
```

```
<adjust> ::= {[[<where>] [what]] FROM
```

```
<where> ::= BOTH | LEADING | TRAILING
```

Table 168. Параметры функции TRIM

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Выражение строкового типа.</td>
</tr>
<tr>
<td>where</td>
<td>Из какого места необходимо удалить подстроку — BOTH</td>
</tr>
<tr>
<td>what</td>
<td>Подстрока, которую надо удалить (неоднократно, если таких вхождений несколько) из входной строки str в её начале и/или конце. По умолчанию является пробелом (’ ’).</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:

VARCHAR или BLOB

Функция TRIM удаляет начальные и /или концевые пробелы (или текст согласно настройкам) из входной строки.

Особенности использования

- Если входной параметр str имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(n), где n является длиной параметра str;
- Подстрока для удаления, если она, конечно, задана, не должна иметь длину больше, чем 32767 байта. Однако при повторениях подстроки в начале и/или конце входного параметра str общее число удаляемых байтов может быть гораздо больше.

При использовании BLOB в параметрах функции может потребоваться загрузить объект в память полностью. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры TRIM
Chapter 8. Встроенные скалярные функции

**Example 323. Использование функции TRIM**

```sql
SELECT TRIM (' Waste no space ') FROM RDB$DATABASE -- Результат: 'Waste no space'

SELECT TRIM (LEADING FROM ' Waste no space ') FROM RDB$DATABASE -- Результат: 'Waste no space '

SELECT TRIM (LEADING '.' FROM ' Waste no space ') FROM RDB$DATABASE -- Результат: ' Waste no space '

SELECT TRIM (TRAILING '!' FROM 'Help!!!!') FROM RDB$DATABASE -- Результат: 'Help'

SELECT TRIM ('la' FROM 'lalala I love you Ella') FROM RDB$DATABASE -- Результат: ' I love you El'
```

См. также: 
OVERLAY(), REPLACE().

**8.3.22. UPPER()**

Доступно в DSQL, PSQL

**Синтаксис**

```sql
UPPER (str)
```

**Table 169. Параметры функции UPPER**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Выражение строкового типа.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

[VAR]CHAR или BLOB

Функция UPPER возвращает входную строку в верхнем регистре. Точный результат зависит от набора символов входной строки. Например, для наборов символов NONE и ASCII только ASCII символы переводятся в верхний регистр; для OCTETS—вся входная строка возвращается без изменений.

**Примеры UPPER**
Example 324. Использование функции UPPER

```
select upper(_iso8859_1 'Débâcle')
from rdb$database
-- returns 'DÉBÂCLE'

select upper(_iso8859_1 'Débâcle' collate fr_fr)
from rdb$database
-- returns 'DEBACLE', following French uppecasing rules
```

См. также:
LOWER().

8.4. Функции для работы с датой и временем

8.4.1. DATEADD()

Доступно в DSQL, PSQL

Синтаксис

```
DATEADD (<args>)
```

<args> ::= <amount> <unit> TO <datetime>
          | <unit>, <amount>, <datetime>

<unit> ::= YEAR | MONTH | WEEK | DAY | WEEKDAY | YEARDAY
          | HOUR | MINUTE | SECOND | MILISECOND
```

Table 170. Параметры функции DATEADD

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>Выражение типа SMALLINT, INTEGER, BIGINT или NUMERIC (отрицательное вычитается).</td>
</tr>
<tr>
<td>unit</td>
<td>Составляющая даты/времени.</td>
</tr>
<tr>
<td>datetime</td>
<td>Выражение типа DATE, TIME или TIMESTAMP.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

DATE, TIME или TIMESTAMP.

Функция DATEADD позволяет добавить заданное число лет, месяцев, недель, часов, минут, секунд, миллисекунд к заданному значению даты/времени.
• С аргументом типа TIMESTAMP и DATE можно использовать любую составляющую даты/времени <unit>;
• Для типа данных TIME разрешается использовать только HOUR, MINUTE, SECOND и MILLISECOND.

Примеры DATEADD

Example 325. Использование функции DATEADD

<table>
<thead>
<tr>
<th>DATEADD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>28 DAY TO CURRENT_DATE</td>
<td></td>
</tr>
<tr>
<td>-6 HOUR TO CURRENT_TIME</td>
<td></td>
</tr>
<tr>
<td>MONTH, 9, DATEOFCONCEPTION</td>
<td></td>
</tr>
<tr>
<td>-38 WEEK TO DATEOFBIRTH</td>
<td></td>
</tr>
<tr>
<td>MINUTE, 90, CAST('NOW' AS TIME)</td>
<td></td>
</tr>
<tr>
<td>? YEAR TO DATE '11-SEP-1973'</td>
<td></td>
</tr>
</tbody>
</table>

SELECT
    CAST(DATEADD(-1 * EXTRACT(MILLISECOND FROM ts) MILLISECOND TO ts) AS VARCHAR(30)) AS t,
    EXTRACT(MILLISECOND FROM ts) AS ms
FROM
    SELECT TIMESTAMP '2014-06-09 13:50:17.4971' as ts
    FROM RDB$DATABASE

<table>
<thead>
<tr>
<th>T</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-06-09 13:50:17.0000</td>
<td>497.1</td>
</tr>
</tbody>
</table>

Cм. также:
DATEDIFF(), Операции, использующие значения даты и времени.

8.4.2. DATEDIFF()

Доступно в
DSQL, PSQL

Синтаксис

DATEDIFF (<args>)

<args> ::= <unit> FROM <moment_1> TO <moment_2>
         | <unit>, <moment_1>, <moment_2>
**Table 171. Параметры функции DATEDIFF**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>Составляющая даты/времени.</td>
</tr>
<tr>
<td>moment_1</td>
<td>Выражение типа DATE, TIME или TIMESTAMP.</td>
</tr>
<tr>
<td>moment_2</td>
<td>Выражение типа DATE, TIME или TIMESTAMP.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата: BIGINT

Функция DATEDIFF возвращает количество лет, месяцев, недель, дней, часов, минут, секунд или миллисекунд между двумя значениями даты/времени.

Особенности использования:

- Параметры DATE и TIMESTAMP могут использоваться совместно. Совместное использование типа TIME с типами DATE и TIMESTAMP не разрешается;
- С аргументом типа TIMESTAMP и DATE можно использовать любую составляющую даты/времени <unit>;
- Для типа данных TIME разрешается использовать только HOUR, MINUTE, SECOND и MILLISECOND.

Функция DATEDIFF не проверяет разницу в более мелких составляющих даты/времени, чем задана в первом аргументе <unit>. В результате получаем:

- DATEDIFF (YEAR, DATE '1-JAN-2009', DATE '31-DEC-2009') вернёт 0, но
- DATEDIFF (YEAR, DATE '31-DEC-2009', DATE '1-JAN-2010') вернёт 1

Однако для более мелких составляющих даты/времени имеем:

- DATEDIFF (DAY, DATE '26-JUN-1908', DATE '11-SEP-1973') вернёт 23818
- DATEDIFF (DAY, DATE '30-NOV-1971', DATE '8-JAN-1972') вернёт 39

Отрицательное значение функции говорит о том, что дата/время в moment_2 меньше, чем в moment_1.

### Примеры DATEDIFF

**Example 326. Использование функции DATEDIFF**

- DATEDIFF (HOUR FROM CURRENT_TIMESTAMP TO TIMESTAMP '12-JUN-2059 06:00')
- DATEDIFF (MINUTE FROM TIME '0:00' TO CURRENT_TIME)
- DATEDIFF (MONTH, CURRENT_DATE, DATE '1-1-1900')
- DATEDIFF (DAY FROM CURRENT_DATE TO CAST (? AS DATE))
См. также:
DATEADD(), Операции, использующие значения даты и времени.

8.4.3. EXTRACT()

Доступно в
DSQL, PSQL

Синтаксис

```
EXTRACT (<part> FROM <datetime>)

<part> ::= 
    YEAR | MONTH | WEEK | DAY | WEEKDAY | YEARDAY
    | HOUR | MINUTE | SECOND | MILLISECOND
    | TIMEZONE_HOUR | TIMEZONE_MINUTE
```

Таблица 172. Параметры функции EXTRACT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>part</td>
<td>Составляющая даты/времени.</td>
</tr>
<tr>
<td>datetime</td>
<td>Выражение типа DATE, TIME или TIMESTAMP.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
SMALLINT или NUMERIC

Функция EXTRACT извлекает составляющие даты и времени из типов данных DATE, TIME и TIMESTAMP.

Таблица 173. Типы и диапазоны результатов функции EXTRACT

<table>
<thead>
<tr>
<th>Составляющая даты/времени</th>
<th>Тип</th>
<th>Диапазон</th>
<th>Комментарий</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>SMALLINT</td>
<td>1–9999</td>
<td>Год</td>
</tr>
<tr>
<td>MONTH</td>
<td>SMALLINT</td>
<td>1–12</td>
<td>Месяц</td>
</tr>
<tr>
<td>WEEK</td>
<td>SMALLINT</td>
<td>1–53</td>
<td>Номер недели в году</td>
</tr>
<tr>
<td>DAY</td>
<td>SMALLINT</td>
<td>1–31</td>
<td>День</td>
</tr>
<tr>
<td>WEEKDAY</td>
<td>SMALLINT</td>
<td>0–6</td>
<td>День недели. 0 — Воскресенье</td>
</tr>
<tr>
<td>YEARDAY</td>
<td>SMALLINT</td>
<td>0–365</td>
<td>Номер дня в году. 0 = 1 января</td>
</tr>
<tr>
<td>HOUR</td>
<td>SMALLINT</td>
<td>0–23</td>
<td>Часы</td>
</tr>
<tr>
<td>MINUTE</td>
<td>SMALLINT</td>
<td>0–59</td>
<td>Минуты</td>
</tr>
<tr>
<td>SECOND</td>
<td>NUMERIC(9,4)</td>
<td>0.0000–59.9999</td>
<td>Секунды. Включает в себя миллисекунды</td>
</tr>
<tr>
<td>MILLISECOND</td>
<td>NUMERIC(9,1)</td>
<td>0.0–999.9</td>
<td>Миллисекунды</td>
</tr>
</tbody>
</table>
Если составляющая даты/времени не присутствует в аргументе дата/время, например SECOND в аргументе с типом DATE или YEAR в TIME, то функция вызовет ошибку.

Из аргумента с типом данных DATE или TIMESTAMP можно извлекать номер недели. В соответствии со стандартом ISO-8601 неделя начинается с понедельника и всегда включает в себя 7 дней. Первой неделей года является первая неделя, у которой в ней больше дней в новом году (по крайней мере, 4): дни 1-3 могут принадлежать предыдущей неделе (52 или 53) прошлого года. По аналогии дни 1-3 текущего года могут принадлежать 1 неделе следующего года.

**Example 327. Использование функции EXTRACT**

```sql
/* получить по дате номер квартала */
SELECT (EXTRACT(MONTH FROM CURRENT_TIMESTAMP)-1)/3+1
FROM RDB$DATABASE
```

См. также:

Типы данных для работы с датой и временем.

### 8.4.4. FIRST_DAY()

**Доступно в**

DSQL, PSQL

**Синтаксис**

```sql
FIRST_DAY(OF <period> FROM date_or_timestamp)
```

<period> ::= YEAR | MONTH | WEEK

**Table 174. Параметры функции FIRST_DAY**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_or_timestamp</td>
<td>Выражение типа DATE или `TIMESTAMP [WITH</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DATE или TIMESTAMP [WITH | WITHOUT] TIME ZONE

Возвращает первый день года, месяца или недели для заданной даты.
Первым днём недели считается воскресенье, как это возвращает функция `EXTRACT` с частью `WEEKDAY`.

Когда в качестве аргумента функции передаётся выражение типа `TIMESTAMP`, то возвращаемое значение сохраняет временную часть.

**Примеры FIRST_DAY**

*Example 328. Использование функции FIRST_DAY*

```sql
SELECT FIRST_DAY(OF MONTH FROM current_date) FROM rdb$database;
SELECT FIRST_DAY(OF YEAR FROM current_timestamp) FROM rdb$database;
SELECT FIRST_DAY(OF WEEK FROM date '2017-11-01') FROM rdb$database;
```

См. также:

*LAST_DAY()*. 

### 8.4.5. LAST_DAY()

Доступно в DSQL, PSQL

**Синтаксис**

```
LAST_DAY(OF <period> FROM date_or_timestamp)
```

<period> ::= YEAR | MONTH | WEEK

**Table 175. Параметры функции LAST_DAY**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_or_timestamp</td>
<td>Выражение типа DATE или <code>TIMESTAMP [WITH</code></td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DATE или `TIMESTAMP [WITH | WITHOUT] TIME_ZONE`

Возвращает последний день года, месяца или недели для заданной даты.

• Последним днём недели считается суббота, как это возвращает функция `EXTRACT` с частью `WEEKDAY`.

• Когда в качестве аргумента функции передаётся выражение типа `TIMESTAMP`, то возвращаемое значение сохраняет временную часть.

**Примеры LAST_DAY**
Использование функции \texttt{LAST\_DAY}

\begin{verbatim}
SELECT LAST_DAY(OF MONTH FROM current_date) FROM rdb$database;
SELECT LAST_DAY(OF YEAR FROM current_timestamp) FROM rdb$database;
SELECT LAST_DAY(OF WEEK FROM date '2017-11-01') FROM rdb$database;
\end{verbatim}

\textit{См. также:}

\texttt{FIRST\_DAY()}

\section{8.5. Функции для работы с типом \textsc{BLOB}}

\subsection{8.5.1. \texttt{BLOB\_APPEND()}}

\textbf{Доступно в}

DSQL, PSQL. Начиная с Firebird 4.0.2.

\textbf{Синтаксис}

\begin{verbatim}
BLOB\_APPEND(<blob> [, <value1>, ... <valueN>]
\end{verbatim}

\textbf{Table 176. Параметры функции \texttt{BLOB\_APPEND}}

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob</td>
<td>BLOB или NULL.</td>
</tr>
<tr>
<td>value</td>
<td>Значение любого типа.</td>
</tr>
</tbody>
</table>

\textbf{Тип возвращаемого результата}

временный не закрытый \textsc{BLOB} с флагом \texttt{BLB\_close\_on\_read}.

Функция \texttt{BLOB\_APPEND} предназначена для конкатенации \textsc{BLOB} без создания промежуточных \textsc{BLOB}. Обычная операция конкатенации с аргументами типа \textsc{BLOB} всегда создаст столько временных \textsc{BLOB}, сколько раз используется.

\textbf{Входные аргументы:}

- Для первого аргумента в зависимости от его значения определено следующее поведение функции:
  - NULL: будет создан новый пустой не закрытый \textsc{BLOB}
  - постоянный \textsc{BLOB} (из таблицы) или временный уже закрытый \textsc{BLOB}: будет создан новый пустой не закрытый \textsc{BLOB} и содержимое первого \textsc{BLOB} будет в него добавлено
  - временный не закрытый \textsc{BLOB}: он будет использован далее
  - другие типы данных преобразуются в строку, будет создан временный не закрытый \textsc{BLOB} с содержимым этой строки
- остальные аргументы могут быть любого типа. Для них определено следующее
поведение:

◦ NULL игнорируется
◦ не BLOB преобразуются в строки (по обычным правилам) и добавляются к содержимому результата
◦ BLOB при необходимости транслитерируются к набору символов первого аргумента и их содержимое добавляется к результату

В качестве выходного значения функция BLOB_APPEND возвращает временный не закрытый BLOB с флагом BLB_close_on_read. Это или новый BLOB, или тот же, что был в первом аргументе. Таким образом ряд операций вида blob = BLOB_APPEND(blob, ...) приведёт к созданию не более одного BLOB (если не пытаться добавить BLOB к самому себе). Этот BLOB будет автоматически закрыт движком при попытке прочитать его клиентом, записать в таблицу или использовать в других выражениях, требующих чтения содержимого.

Проверка BLOB на значение NULL с помощью оператора IS [NOT] NULL не читает его, а следовательно временный открытый BLOB не будет закрыт при таких проверках.

execute block
returns (b blob sub_type text)
  as
  begin
    -- создаст новый временный не закрытый BLOB
    -- и запишет в него строку из 2-ого аргумента
    b = blob_append(null, 'Hello ');
    -- добавляет во временный BLOB две строки не закрывая его
    b = blob_append(b, 'World', '!');
    -- сравнение BLOB со строкой закроет его, ибо для этого надо прочитать BLOB
    if (b = 'Hello World!') then
      begin
        -- ...
      end
    -- создаст временный закрытый BLOB добавив в него строку
    b = b || 'Close';
    suspend;
  end

Используйте функции LIST и BLOB_APPEND для конкатенации BLOB. Это позволит сэкономить объём потребляемой памяти, дисковый ввод/вывод, а также предотвратит разрастание базы данных из-за создания множества временных BLOB при использовании операторов конкатенации.

Example 330. Использование функции BLOB_APPEND

Предположим вам надо собрать JSON на стороне сервера. У нас есть PSQL пакет JSON_UTILS с набором функций для преобразования элементарных типов данных в...
JSON нотацию. Тогда сборка JSON с использованием функции BLOB_APPEND будет выглядеть следующим образом:

```sql
EXECUTE BLOCK
RETURNS (JSON_STR BLOB SUB_TYPE TEXT CHARACTER SET UTF8) AS
  DECLARE JSON_M BLOB SUB_TYPE TEXT CHARACTER SET UTF8;
BEGIN FOR
  SELECT HORSE.CODE_HORSE, HORSE.NAME, HORSE.BIRTHDAY
  FROM HORSE
  WHERE HORSE.CODE_DEPARTURE = 15
  FETCH FIRST 1000 ROW ONLY
  AS CURSOR C
  DO BEGIN
    SELECT LIST(
      '{' ||
      JSON_UTILS.NUMERIC_PAIR('age', MEASURE.AGE) ||
      ',', ||
      JSON_UTILS.NUMERIC_PAIR('height', MEASURE.HEIGHT_HORSE) ||
      ',', ||
      JSON_UTILS.NUMERIC_PAIR('length', MEASURE.LENGTH_HORSE) ||
      ',', ||
      JSON_UTILS.NUMERIC_PAIR('chestaround', MEASURE.CHESTAROUND) ||
      ',', ||
      JSON_UTILS.NUMERIC_PAIR('wristarround', MEASURE.WRISTAROUND) ||
      ',', ||
      JSON_UTILS.NUMERIC_PAIR('weight', MEASURE.WEIGHT_HORSE) ||
      '}'
    ) AS JSON_M
    FROM MEASURE
    WHERE MEASURE.CODE_HORSE = :C.CODE_HORSE
    INTO JSON_M;

  JSON_STR = BLOB_APPEND(
    JSON_STR,
    IIF(JSON_STR IS NULL, [' ', ', ' || ascii_char(13)),
    '{',
    JSON_UTILS.INTEGER_PAIR('code_horse', C.CODE_HORSE),
    ',',
    JSON_UTILS.STRING_PAIR('name', C.NAME),
    ',',
    JSON_UTILS.TIMESTAMP_PAIR('birthday', C.BIRTHDAY),
    ',',
  )
)
```
Аналогичный пример с использованием обычного оператора конкатенации `||` работает в 18 раз медленнее, и делает в 1000 раз больше операций записи на диск.

# 8.6. Функции для работы с типом DECFLOAT

## 8.6.1. COMPARE_DECFLOAT()

*Доступно в* DSQl, PSQL

**Синтаксис**

```
COMPARE_DECFLOAT (decfloat1, decfloat2)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>decfloat1, decfloat2</code></td>
<td>Значения или выражения типа DECFLOAT или быть совместимыми с типом DECFLOAT.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

SMALLINT

Функция COMPARE_DECFLOAT сравнивает два значения типа DECFLOAT, которые могут быть одинаковыми, разными или неупорядоченными. Замыкающие нули учитываются при сравнении.

Функция возвращает:

- 0 — Значения равны;
- 1 — Первое значение меньше чем второе;
- 2 — Первое значение больше чем второе;
- 3 — Значения не упорядочены (одно или оба NaN/sNaN).

В отличие от операторов сравнения (`<`, `=`, `>` и др.) сравнение является точным, т.е. `COMPARE_DECFLOAT(2.17, 2.170)` вернёт 2, а не 0.
См. также: TOTALORDER()

### 8.6.2. NORMALIZE_DECFLOAT()

Доступно в
DSQL, PSQL

**Синтаксис**

```
NORMALIZE_DECFLOAT (decfloat_value)
```

**Table 178. Параметры функции NORMALIZE_DECFLOAT**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>decfloat_value</td>
<td>Значение или выражение типа DECFLOAT или быть совместимым с типом DECFLOAT.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DECFLOAT

Функция NORMALIZE_DECFLOAT возвращает число в нормализованном виде. Это обозначает, что для любого ненулевого значения удаляются завершающие нули с соответствующей коррекцией экспоненты.

**Примеры NORMALIZE_DECFLOAT**

**Example 331. Нормализация различных значений типа DECFLOAT**

```
NORMALIZE_DECFLOAT(12.00) -- возвращает 12
NORMALIZE_DECFLOAT(120) -- возвращает 1.2E+2
```

### 8.6.3. QUANTIZE()

Доступно в
DSQL, PSQL

**Синтаксис**

```
QUANTIZE (decfloat_value, exp_value)
```

**Table 179. Параметры функции QUANTIZE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>decfloat_value</td>
<td>Значение или выражение типа DECFLOAT или быть совместимым с типом DECFLOAT.</td>
</tr>
</tbody>
</table>

**Chapter 8. Встроенные скалярные функции**
### Таблица Параметров и Описание

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp_value</td>
<td>Значение или выражение для использования в качестве показателя степени; должно иметь тип DECFLOAT или быть совместимым с типом DECFLOAT.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DECFLOAT

Функция `QUANTIZE` возвращает значение первого аргумента масштабированным с использованием второго значения в качестве шаблона. Другими словами функция `QUANTIZE` возвращает значение DECFLOAT, равное по значению (за исключением любого округления) и знаку `decfloat_value`, а также экспоненте, равной по значению экспоненте `exp_value`. Функцию `QUANTIZE` можно использовать для реализации округления с точностью до нужного знака, например, округление до ближайшего цента с использованием установленного режима округления DECFLOAT.

Тип возвращаемого значения — DECFLOAT(16), если оба аргумента — DECFLOAT(16), в противном случае тип результата — DECFLOAT(34).

#### Целевой показатель

Целевой показатель — это показатель, используемый в формате хранения Decimal64 или Decimal128 для DECFLOAT из `exp_value`. Это не обязательно то же самое, что экспонента, отображаемая в таких инструментах, как `isql`. Например, значение 1.23E+2 — это коэффициент 123 и показатель степени 0, а значение 1.2 — это коэффициент 12 и показатель степени -1.

Если показатель `decfloat_value` больше, чем показатель `exp_value`, коэффициент `decfloat_value` умножается на степень десяти, и его показатель уменьшается, если показатель меньше, то его коэффициент округляется с использованием текущего режима округления decfloat, и его показатель увеличивается.

Когда невозможно достичь целевого показателя экспоненты, поскольку коэффициент превысит целевую точность (16 или 34 десятичных знака), то либо возникает ошибка "Decfloat float invalid operation", либо возвращается NaN (в зависимости от текущего конфигурации decfloat traps).

Ограничений на exp_value практически нет. Однако почти во всех случаях использования NaN/sNaN/Infinity будет вызывать исключение (если это не разрешено текущей конфигурацией decfloat traps).

Если одно из значений NULL, то результатом функции будет NULL и т.д.

**Example 332. Использование функции QUANTIZE**

```sql
select v, pic, quantize(v, pic) from examples;
```

<table>
<thead>
<tr>
<th>V</th>
<th>PIC</th>
<th>QUANTIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.6.4. TOTALORDER()

Доступно в
DSQL, PSQL

Синтаксис

TOTALORDER (decfloat1, decfloat2)

Table 180. Параметры функции TOTALORDER

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>decfloat1, decfloat2</td>
<td>Значение или выражение типа DECFLOAT или быть совместимым с типом DECFLOAT.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
SMALLINT

Функция TOTALORDER сравнивает два значения типа DECFLOAT, включая специальные значения. Сравнение является точным. Замыкающие нули учитываются при сравнении.

Функция возвращает:

• -1 — если первое значение меньше второго;
• 0 — если значения равны;
• 1 — если первое значение больше второго.

Сравнений значений DECFLOAT происходит в следующем порядке:

-\text{nan} < -\text{snan} < -\text{inf} < -0.1 < -0.10 < -0 < 0 < 0.10 < 0.1 < \text{inf} < \text{snan} < \text{nan}

См. также: COMPARE_DECFLOAT()
8.7. Криптографические функции

В Firebird 4.0 поддерживается только подмножество симметричных алгоритмов шифрования (как блочных так и потоковых), так и RSA.

8.7.1. CRYPT_HASH()

Доступно в
DSQL, PSQL

Синтаксис

CRYPT_HASH (value USING <algorithm>)

<algorithm> ::= { MD5 | SHA1 | SHA256 | SHA512 }

Table 181. Параметры функции CRYPT_HASH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Выражение любого типа. Не строковые и не бинарные типы приводятся к строке.</td>
</tr>
<tr>
<td>algorithm</td>
<td>Алгоритм хеширования.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
VARBINARY

Функция CRYPT_HASH возвращает криптографический хэш входной строки, используя указанный алгоритм. Эта функция полностью поддерживает текстовые BLOB любой длины и с любым набором символов. Предложение USING позволяет указать алгоритм по которому вычисляет хэш.

Алгоритмы MD5 и SHA1 не рекомендуются для использования из-за известных серьезных проблем, которые предоставляются только для обратной совместимости.

Примеры CRYPT_HASH

Example 333. Использование функции CRYPT_HASH

```
SELECT CRYPT_HASH(x USING SHA256) FROM MyTable;
-- результат типа VARBINARY
```

8.7.2. DECRYPT()

Доступно в
DSQL, PSQL
Синтаксис

```
DECRYPT (encrypted_input
    [USING <algorithm>] [MODE <mode>]
    KEY key
    [IV iv] [CTR_LENGTH ctr_length]
    [COUNTER initial_counter]
)
```

<algorithm> ::= <block_cipher> | <stream_cipher>

<block_cipher> ::= AES | ANUBIS | BLOWFISH | KHAZAD | RC5
    | RC6 | SAFER+ | TWOFISH | XTEA

<stream_cipher> ::= CHACHA20 | RC4 | SOBER128

<mode> ::= CBC | CFB | CTR | ECB | OFB

<ctr_type> ::= CTR_BIG_ENDIAN | CTR_LITTLE_ENDIAN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypted_input</td>
<td>Зашифрованный BLOB или (двоичная) строка</td>
</tr>
<tr>
<td>algorithm</td>
<td>Алгоритм шифрования. Поддерживаются как блочные, так и потоковые алгоритмы.</td>
</tr>
<tr>
<td>mode</td>
<td>Режим шифрования. Обязателен для блочных алгоритмов шифрования.</td>
</tr>
<tr>
<td>key</td>
<td>Ключ шифрования.</td>
</tr>
<tr>
<td>iv</td>
<td>Вектор инициализации (IV). Должен быть указан для всех блочных алгоритмов шифрования за исключением ECB и всех потоковых алгоритмов шифрования за исключением RC4.</td>
</tr>
<tr>
<td>ctr_type</td>
<td>Порядок байтов счётчика. Может быть указан только в режиме CTR. По умолчанию используется CTR_LITTLE_ENDIAN.</td>
</tr>
<tr>
<td>ctr_length</td>
<td>Длина счётчика в байтах. Может быть указана только в режиме CTR. По умолчанию равна длине вектора инициализации IV.</td>
</tr>
<tr>
<td>initial_counter</td>
<td>Начальное значение счётчика. Может быть указана только для алгоритма CHACHA20. По умолчанию равно 0.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

BLOB или VARBINARY.

Функция DECRYPT десифрует данные с использованием симметричного шифра. Размеры строк передаваемых в эту функцию должны соответствовать требованиям выбранного алгоритма и режима.
Example 334. Использование функции `DECRYPT`

```sql
select decrypt(x'0154090759DF' using sober128 key 'AbcdAbcdAbcdAbcd'
  iv '01234567')
from rdb$database;

select decrypt(secret_field using aes mode ofb key '0123456701234567'
  iv init_vector)
from secure_table;
```

См. также:

`ENCRYPT()`.

8.7.3. `ENCRYPT()`

Доступно в

DSQL, PSQL

Синтаксис

```sql
ENCRYPT (input
  [USING <algorithm>] [MODE <mode>]
  KEY key
  [IV iv] [<ctr_type>] [CTR_LENGTH ctr_length]
  [COUNTER initial_counter] )

<algorithm> ::= <block_cipher> | <stream_cipher>

<block_cipher> ::= AES | ANUBIS | BLOWFISH | KHAZAD | RC5
  | RC6 | SAFER+ | TWOFISH | XTEA

<stream_cipher> ::= CHACHA20 | RC4 | SOBER128

<mode> ::= CBC | CFB | CTR | ECB | OFB

<ctr_type> ::= CTR_BIG_ENDIAN | CTR_LITTLE_ENDIAN
```

Table 183. Параметры функции `ENCRYPT`

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Выражение строкового типа или BLOB, которое необходимо зашифровать.</td>
</tr>
<tr>
<td>algorithm</td>
<td>Алгоритм шифрования. Поддерживаются как блочные, так и потоковые алгоритмы.</td>
</tr>
</tbody>
</table>
### Параметр | Описание
--- | ---
mode | Режим шифрования. Обязателен для блочных алгоритмов шифрования.
key | Ключ шифрования.
iv | Вектор инициализации (IV). Должен быть указан для всех блочных алгоритмов шифрования за исключением ECB и всех потоковых алгоритмов шифрования за исключением RC4.
ctr_type | Порядок байтов счётчика. Может быть указан только в режиме CTR. По умолчанию используется CTR_LITTLE_ENDIAN.
ctr_length | Длина счётчика в байтах. Может быть указана только в режиме CTR. По умолчанию равна длине вектора инициализации IV.
initial_counter | Начальное значение счётчика. Может быть указана только для алгоритма CHACHA20. По умолчанию равно 0.

#### Тип возвращаемого результата
BLOB или VARBINARY

Функция ENCRYPT шифрует данные с использованием симметричного шифра.

- Эта функция возвращает BLOB SUB_TYPE BINARY, если первым аргументом является BLOB, и VARBINARY для всех других текстовых и двоичных типов.
- Размеры строк (например, key и iv) передаваемых в эту функцию должны соответствовать требованиям выбранного алгоритма и режима. Подробнее см. таблицу Требования алгоритмов шифрования.
  - Как правило, размер iv должен соответствовать размеру блока алгоритма.
  - Для режимов ECB и CBC input должен быть кратным размеру блока, вам нужно будет вручную заполнить нулями или пробелами, если это необходимо.

Особенности различных алгоритмов и режимов выходят за рамки данного справочника по языку.

#### Table 184. Требования алгоритмов шифрования

<table>
<thead>
<tr>
<th>Алгоритм</th>
<th>Размер ключа (байт)</th>
<th>Размер блока (байт)</th>
<th>Примечание</th>
</tr>
</thead>
<tbody>
<tr>
<td>Блочное шифрование</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>16, 24, 32</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>ANUBIS</td>
<td>16 - 40, с шагом 4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>BLOWFISH</td>
<td>8 - 56</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>KHAZAD</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Алгоритм</td>
<td>Размер ключа (байт)</td>
<td>Размер блока (байт)</td>
<td>Примечание</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>RC5</td>
<td>8 - 128</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RC6</td>
<td>8 - 128</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>SAFER+</td>
<td>16, 24, 32</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>TWOFISH</td>
<td>16, 24, 32</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>XTEA</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

**Поточное шифрование**

<table>
<thead>
<tr>
<th>Алгоритм</th>
<th>Размер ключа (байт)</th>
<th>Размер блока (байт)</th>
<th>Примечание</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHACHA20</td>
<td>16, 32</td>
<td>1</td>
<td>Размер (IV) составляет 8 или 12 байт. Для размера 8 initial_counter - это 64-битное целое число, для размера 12 - 32-битное.</td>
</tr>
<tr>
<td>RC4</td>
<td>5 - 256</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SOBER128</td>
<td>4x</td>
<td>1</td>
<td>Размер (IV) составляет 4 байт, длина не зависит от размера ключа.</td>
</tr>
</tbody>
</table>

**Example 335. Использование функции ENCRYPT**

```sql
select encrypt('897897' using sober128 key 'AbcdAbcdAbcdAbcd' iv '01234567')
from rdb$database;
```

**См. также:**

`DECRYPT().`

### 8.7.4. RSA_PRIVATE()

**Доступно в**

DSQL, PSQL

**Синтаксис**

```sql
RSA_PRIVATE (size)
```

**Table 185. Параметры функции RSA_PRIVATE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Размер ключа в байтах.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

VARBINARY

Функция `RSA_PRIVATE` возвращает RSA закрытый ключ заданной длины (в байтах) в PKCS#1
формате как строку VARBINARY.

Example 336. Использование функции RSA_PRIVATE

```sql
select rdb$set_context('USER_SESSION', 'private_key', rsa_private(256))
from rdb$database;
```

См. также:
RSA_PUBLIC().

**8.7.5. RSA_PUBLIC()**

Доступно в
DSQL, PSQL

Синтаксис

```sql
rsa_public (private-key)
```

Table 186. Параметры функции RSA_PUBLIC

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>private-key</td>
<td>RSA закрытый ключ.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARBINARY

Функция RSA_PUBLIC возвращает RSA открытый ключ для заданного RSA закрытого ключа. Оба ключа должны быть в PKCS#1 формате.

Example 337. Использование функции RSA_PUBLIC

Закрытый ключ должен быть инициализирован ранее см. пример в RSA_PRIVATE

```sql
select rdb$set_context('USER_SESSION', 'public_key',
                      rsa_public(rdb$get_context('USER_SESSION', 'private_key'))) 
from rdb$database;
```

См. также:
RSA_PRIVATE().

**8.7.6. RSA_ENCRYPT()**

Доступно в
DSQL, PSQL
RSA ENCRYPT (<data> KEY <public_key> [LPARAM <tag>] [HASH <hash>])

<hash> ::= { MD5 | SHA1 | SHA256 | SHA512 }

Table 187. Параметры функции RSA ENCRYPT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Данные (строка или BLOB) для шифрования.</td>
</tr>
<tr>
<td>public_key</td>
<td>Открытый RSA ключ, который возвращает функция RSA_PUBLIC.</td>
</tr>
<tr>
<td>tag</td>
<td>Дополнительный системный тег, который можно применять для определения того, какая система закодировала сообщение. Значением по умолчанию является NULL.</td>
</tr>
<tr>
<td>hash</td>
<td>Алгоритм хеширования. По умолчанию SHA256.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата:
VARBINARY

Заполняет данные, используя заполнение OAEP, и шифрует их, используя открытый ключ RSA. Обычно используется для шифрования коротких симметричных ключей, которые затем используются в блочных шифрах для шифрования сообщения.

Example 338. Использование функции RSA ENCRYPT

Открытый ключ должен быть инициализирован ранее см. пример в RSA_PUBLIC()

```sql
select rdb$set_context('USER_SESSION', 'msg',
    rsa_encrypt('Some message' key rdb$get_context('USER_SESSION', 'public_key'))
from rdb$database;
```

См. также:
RSA_PUBLIC(), RSA_DECRYPT().

8.7.7. RSA_DECRYPT()

Доступно в
DSQL, PSQL

Синтаксис

RSA_DECRYPT (<data> KEY <private_key> [LPARAM <tag>] [HASH <hash>])

<hash> ::= { MD5 | SHA1 | SHA256 | SHA512 }
Table 188. Параметры функции RSA_DECRYPT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Данные (строка или BLOB) для дешифрования.</td>
</tr>
<tr>
<td>private_key</td>
<td>Закрытый RSA ключ, который возвращает функция RSA_PRIVATE.</td>
</tr>
<tr>
<td>tag</td>
<td>Дополнительный системный тег. Должно быть тем же самым значением, которое передавалось RSA_ENCRYPT. Если оно не совпадает с тем, который использовался во время кодирования, эта функция не расшифровывает пакет. Значением по умолчанию является NULL.</td>
</tr>
<tr>
<td>hash</td>
<td>Алгоритм хеширования. По умолчанию SHA256.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата: VARCHAR

Расшифровывает с использованием закрытого ключа RSA, и удаляет OAEP дополненные данные.

Example 339. Использование функции RSA_DECRYPT

Закрытый ключ должен быть инициализирован ранее см. пример в RSA_PRIVATE(). Данные для расшифровки используются из примера в RSA_ENCRYPT().

```sql
select RSA_DECRYPT(rdb$get_context('USER_SESSION', 'msg'))
    key rdb$get_context('USER_SESSION', 'private_key'))
from RDB$DATABASE;
```

См. также:
RSA_PRIVATE(), RSA_ENCRYPT().

8.7.8. RSA_SIGN_HASH()

Доступно в
DSQL, PSQL

Синтаксис

```
RSA_SIGN_HASH (<data> KEY <private_key> [HASH <hash>] [SALT_LENGTH <length>])
```

<hash> ::= { MD5 | SHA1 | SHA256 | SHA512 }

Table 189. Параметры функции RSA_SIGN_HASH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Данные (строка или BLOB) для кодирования.</td>
</tr>
<tr>
<td>private_key</td>
<td>Закрытый RSA ключ, который возвращает функция RSA_PRIVATE.</td>
</tr>
</tbody>
</table>
Тип возвращаемого результата:
VARBINARY

Выполняет PSS-кодирование дайджеста сообщения для подписи и подписывает его с использованием закрытого ключа RSA. Возвращает подпись сообщения.

Example 340. Использование функции RSA_SIGN_HASH

Закрытый ключ должен быть инициализирован ранее см. пример в RSA_PRIVATE().

```sql
select rdb$set_context('USER_SESSION', 'msg',
    rsa_sign_hash(crypt_hash('Test message' using sha256)
        key rdb$get_context('USER_SESSION', 'private_key')))
from rdb$database;
```

См. также:
RSA_PRIVATE(), RSA_VERIFY_HASH().

### 8.7.9. RSA_VERIFY_HASH()

Доступно в
DSQL, PSQL

Синтаксис

```
RSA_VERIFY_HASH (<data> SIGNATURE <signature> KEY <public_key> [HASH <hash>]
[SALT_LENGTH <length>])
```

Table 190. Параметры функции RSA_VERIFY_HASH

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Данные (строка или BLOB) для кодирования.</td>
</tr>
<tr>
<td>signature</td>
<td>Подпись. Должно быть значением возвращаемым функцией RSA_SIGN_HASH.</td>
</tr>
<tr>
<td>public_key</td>
<td>Открытый RSA ключ, который возвращает функция RSA_PUBLIC.</td>
</tr>
<tr>
<td>hash</td>
<td>Алгоритм хеширования. По умолчанию SHA256.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>length</td>
<td>Указывает на длину желаемой соли и, как правило, должен быть небольшим. Хорошее значение от 8 до 16.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

BOOLEAN

Выполняет PSS-кодирование дайджеста сообщения для подписи и проверяет его цифровую подпись, используя открытый ключ RSA. Возвращает результат проверки подписи.

**Example 341. Использование функции RSA_VERIFY_HASH**

Открытый ключ должен быть инициализирован ранее см. пример в RSA_PUBLIC(). Цифровая подпись получена ранее с помощью функции RSA_SIGN_HASH().

```sql
select rsa_verify_hash(crypt_hash('Test message' using sha256)
  signature rdb$get_context('USER_SESSION', 'msg')
  key rdb$get_context('USER_SESSION', 'public_key'))
from rdb$database;
```

См. также:

RSA_SIGN_HASH(), RSA_PUBLIC().

**8.8. Функции преобразования типов**

**8.8.1. CAST()**

Доступно в DSQL, PSQL

Синтаксис

```sql
CAST(value | NULL AS <type>)
```

`<type>` ::=  
  `<datatype>`  
  | [TYPE OF] domain  
  | TYPE OF COLUMN relname.colname

`<datatype>` ::=  
  `<scalar_datatype>` | `<blob_datatype>` | `<array_datatype>`

`<scalar_datatype>` ::= См. Синтаксис скалярных типов данных

`<blob_datatype>` ::= См. Синтаксис типа данных BLOB
Таблица 191. Параметры функции CAST

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>SQL выражение.</td>
</tr>
<tr>
<td>datatype</td>
<td>Тип данных SQL.</td>
</tr>
<tr>
<td>domain</td>
<td>Домен.</td>
</tr>
<tr>
<td>relname</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>colname</td>
<td>Имя столбца таблицы или представления.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
<type>.

Функция CAST служит для явного преобразования данных из одного типа данных в другой тип данных или домен. Если это невозможно будет выдана ошибка.

Таблица 192. Допустимые преобразования для функции CAST

<table>
<thead>
<tr>
<th>Из типа</th>
<th>В тип</th>
</tr>
</thead>
<tbody>
<tr>
<td>Числовые типы</td>
<td>Числовые типы, [VAR]CHAR, BLOB</td>
</tr>
<tr>
<td>[VAR]CHAR, BLOB</td>
<td>[VAR]CHAR, BLOB, BOOLEAN, Числовые типы, DATE, TIME, TIMESTAMP</td>
</tr>
<tr>
<td>DATE, TIME</td>
<td>[VAR]CHAR, BLOB, TIMESTAMP</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>[VAR]CHAR, BLOB, TIME, DATE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>[VAR]CHAR, BLOB</td>
</tr>
</tbody>
</table>

Имейте в виду, что иногда информация может быть потеряна, например, когда вы преобразуете тип TIMESTAMP к DATE. Кроме того, тот факт, что типы совместимы для функции CAST, ещё не гарантирует, что преобразование будет успешным. “CAST (123456789 AS SMALLINT)” безусловно приведёт к ошибке, так же как и “CAST('Judgement Day' as DATE)”.

Вы можете применить преобразование типа к параметрам оператора:

CAST (? AS INTEGER)

Это дает вам контроль над типом полей ввода.

Преобразование к домену или к его базовому типу

При преобразовании к домену должны быть удовлетворены любые ограничения (NOT NULL и/или CHECK) объявленные для домена, иначе преобразование не будет выполнено. Помните, что проверка CHECK проходит, если его вычисление даёт TRUE или UNKNOWN (NULL). Для следующих операторов:
CREATE DOMAIN quint AS INT CHECK (VALUE >= 5000)
SELECT CAST (2000 AS quint) FROM rdb$database -- (1)
SELECT CAST (8000 AS quint) FROM rdb$database -- (2)
SELECT CAST (null AS quint) FROM rdb$database -- (3)

только (1) завершится с ошибкой.

При использовании модификатора TYPE OF выражение будет преобразовано к базовому типу домена, игнорируя любые ограничения. Для домена quint, объявленного выше, оба преобразования будут эквивалентны и оба будут успешно выполнены:

SELECT CAST (2000 AS TYPE OF quint) FROM rdb$database
SELECT CAST (2000 AS INT) FROM rdb$database

При использовании TYPE OF с [VAR]CHAR типом, его набор символов и порядок сортировки (collate) сохраняются.

CREATE DOMAIN iso20 VARCHAR(20) CHARACTER SET iso8859_1;
CREATE DOMAIN dunl20 VARCHAR(20) CHARACTER SET iso8859_1 COLLATE du_nl;
CREATE TABLE zinnen (zin VARCHAR(20));
COMMIT;
INSERT INTO zinnen VALUES ('Deze');
INSERT INTO zinnen VALUES ('Die');
INSERT INTO zinnen VALUES ('die');
INSERT INTO zinnen VALUES ('deze');
SELECT CAST(zin AS TYPE OF iso20) FROM zinnen ORDER BY 1;
-- returns Deze -> Die -> deze -> die
SELECT CAST(zin AS TYPE OF dunl20) FROM zinnen ORDER BY 1;
-- returns deze -> Deze -> die -> Die

Если определение домена изменяется, то существующие преобразования к домену или его типу могут стать ошибочными. Если такие преобразования происходят в PSQL модулях, то их ошибки могут быть обнаружены. См. Поле RDB$VALID_BLR.

Преобразование к типу столбца

Разрешено преобразовывать выражение к типу столбца существующей таблицы или представления. При этом будет использован только сам тип, для строковых типов будет использован так же набор символов, но не последовательность сортировки. Ограничения и значения по умолчанию исходного столбца не применяются.

CREATE TABLE ttt (  
s VARCHAR(40) CHARACTER SET utf8 COLLATE unicode_ci_ai );
COMMIT;
Если определение столбца изменяется, то существующие преобразования к его типу могут стать ошибочными. Если такие преобразования происходят в PSQL модулях, то их ошибки могут быть обнаружены. См. Поле RDB$VALID_BLR.

**См. также:**
Явное преобразование типов данных.

**Примеры приведения типов**

```sql
SELECT CAST ('12' || '-June-' || '1959' AS DATE) FROM rdb$database
```

Заметьте, что в некоторых случаях вы можете не использовать синтаксис преобразования как в примере выше, так как Firebird поймёт из контекста (сравнение с полем типа DATE) как интерпретировать строку:

```sql
UPDATE People SET AgeCat = 'Old'
WHERE BirthDate < '1-Jan-1943'
```

Но это не всегда возможно. Преобразование в примере ниже не может быть опущено, так как система будет пытаться преобразовать строку к числу чтобы вычесть из неё число:

```sql
SELECT CAST ('TODAY' AS DATE) - 7 FROM rdb$database
```

### 8.9. Функции побитовых операций

#### 8.9.1. BIN_AND()

**Доступно в**
DSQL, PSQL

**Синтаксис**

```sql
BIN_AND (number, number [, number ...])
```

**Table 193. Параметры функции BIN_AND**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Целое число.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**
SMALLINT, INTEGER, BIGINT или INT128

Функция BIN_AND возвращает результат побитовой операции AND (И) аргументов.

См. также:
BIN_OR(), BIN_XOR().

8.9.2. BIN_NOT()

Доступно в
DSQL, PSQL

Синтаксис

```
BIN_NOT (number)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Целое число.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
SMALLINT, INTEGER, BIGINT или INT128

Функция BIN_NOT возвращает результат побитовой операции NOT над аргументом.

См. также:
BIN_OR(), BIN_AND().

8.9.3. BIN_OR()

Доступно в
DSQL, PSQL

Синтаксис

```
BIN_OR (number, number [, number ...])
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Целое число.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
SMALLINT, INTEGER, BIGINT или INT128

Функция BIN.OR возвращает результат побитовой операции OR (ИЛИ) аргументов.

См. также:
**8.9.4. BIN_SHL()**

Доступно в

DSQL, PSQL

**Синтаксис**

```
BIN_SHL (number, shift)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Целое число.</td>
</tr>
<tr>
<td>shift</td>
<td>Количество бит, на которое смещается значение number.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

BIGINT или INT128.

Функция BIN_SHL возвращает первый параметр, побитно смещённый влево на значение второго параметра, т.е. \( a \ll b \) или \( a \cdot 2^b \).

*См. также:*

**8.9.5. BIN_SHR()**

Доступно в

DSQL, PSQL

**Синтаксис**

```
BIN_SHR (number, shift)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Целое число.</td>
</tr>
<tr>
<td>shift</td>
<td>Количество бит на которое смещается значение number.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

BIGINT или INT128.

Функция BIN_SHR возвращает первый параметр, побитно смещённый вправо на значение второго параметра, т.е. \( a \gg b \) или \( a / 2^b \).
• Выполняемая операция является арифметическим сдвигом вправо (SAR), и это означает, что знак первого операнда всегда сохраняется.

См. также:
BIN_SHL().

8.9.6. BIN_XOR()
Доступно в
DSQL, PSQL

Синтаксис

```
BIN_XOR (number, number [, number ...])
```

Таблица 198. Параметры функции BIN_XOR

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Целое число.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
SMALLINT, INTEGER, BIGINT или INT128

Функция BIN_XOR возвращает результат побитовой операции XOR аргументов.

См. также:
BIN_AND(), BIN_OR().

8.10. Функции для работы с UUID

8.10.1. CHAR_TO_UUID()
Доступно в
DSQL, PSQL

Синтаксис

```
CHAR_TO_UUID (ascii_uuid)
```

Таблица 199. Параметры функции CHAR_TO_UUID

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii_uuid</td>
<td>36-символьное представление UUID. ‘-’ (дефис) в положениях 9, 14, 19 и 24; допустимые шестнадцатеричные цифры в любых других позициях, т.е. 'A0bF4E45-3029-2a44-D493-4998c9b439A3'.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
Функция `CHAR_TO_UUID` преобразует читабельную 36-ти символьную символику UUID к соответствующему 16-ти байтовому значению UUID.

Примеры `CHAR_TO_UUID`

Example 342. Использование функции `CHAR_TO_UUID`

```
SELECT CHAR_TO_UUID( 'A0bF4E45-3029-2a44-D493-4998c9b439A3' ) FROM rdb$database
-- returns A0BF4E4530292A44D4934998C9B439A3 (16-byte string)

SELECT CHAR_TO_UUID( 'A0bF4E45-3029-2A44-X493-4998c9b439A3' ) FROM rdb$database
-- error: -Human readable UUID argument for CHAR_TO_UUID must
-- have hex digit at position 20 instead of "X (ASCII 88)"
```

См. также:
`GEN_UUID()`, `UUID_TO_CHAR()`.

8.10.2. `GEN_UUID()`

Доступно в
DSQL, PSQL

Синтаксис

```
GEN_UUID()
```

Тип возвращаемого результата
`BINARY(16)`

Функция возвращает универсальный уникальный идентификатор ID в виде 16-байтной строки символов, отвечающий требованиям стандарта RFC-4122. Функция возвращает строку UUID 4-ой версии, где несколько битов зарезервированы, а остальные являются случайными.

Примеры `GEN_UUID`

Example 343. Использование функции `GEN_UUID`

```
SELECT GEN_UUID() AS GUID FROM rdb$database

GUID
=======
```
См. также:
CHAR_TO_UUID, UUID_TO_CHAR.

8.10.3. UUID_TO_CHAR()

Доступно в
DSQL, PSQL

Синтаксис

```
UUID_TO_CHAR (uuid)
```

Table 200. Параметры функции UUID_TO_CHAR

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>16-байтный UUID.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
CHAR(36)

Функция UUID_TO_CHAR конвертирует 16-ти байтный UUID в его 36-ти знаковое ASCII человекочитаемое представление. Тип возвращаемого значения CHAR(36).

Примеры UUID_TO_CHAR

Example 344. Использование функции UUID_TO_CHAR

```
SELECT UUID_TO_CHAR(GEN_UUID()) FROM RDB$DATABASE;
SELECT UUID_TO_CHAR('876C45F4569B320DBCB4735AC3509E5F') FROM RDB$DATABASE;
-- returns '876C45F4-569B-320D-BCB4-735AC3509E5F'
SELECT UUID_TO_CHAR('Firebird swings!') FROM RDB$DATABASE;
-- returns '46697265-6269-7264-2073-77696E677321'
```

См. также:
GEN_UUID(), CHAR_TO_UUID().
8.11. Функции для работы с генераторами (последовательностями)

8.11.1. GEN_ID()

Доступно в DSQL, PSQL

Синтаксис

```
GEN_ID (generator-name, step)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>generator-name</td>
<td>Имя генератора (последовательности).</td>
</tr>
<tr>
<td>step</td>
<td>Шаг приращения.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

BIGINT

Функция GEN_ID увеличивает значение генератора или последовательности и возвращает новое значение.

Если step равен 0, функция не будет ничего делать со значением генератора и вернёт его текущее значение.

• Начиная с Firebird 2.0 для получения следующего значение последовательности (генератора) стало доступно использование совместимого с SQL-стандартом оператора `NEXT VALUE FOR`.

Если значение параметра step меньше нуля, произойдёт уменьшение значения генератора. Следует быть крайне аккуратным при таких манипуляциях в базе данных, они могут привести к потере целостности данных.

Примеры GEN_ID

**Example 345. Использование функции GEN_ID**

```
NEW.ID = GEN_ID (GEN_TABLE_ID, 1);
```

См. также:

`NEXT VALUE FOR`, `SEQUENCE (GENERATOR)`, `ALTER SEQUENCE`, `SET GENERATOR`. 

Chapter 8. Встроенные скалярные функции

542
8.12. Условные функции

8.12.1. COALESCE() 

Доступно в DSQL, PSQL

Синтаксис

COALESCE (<exp1>, <exp2> [, <expN> ... ])

Table 202. Параметры функции COALESCE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp1, exp2 ... expN</td>
<td>Выражения любого совместимого типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата зависит от типов входных аргументов

Функция COALESCE принимает два или более аргумента возвращает значение первого не-NULL аргумента. Если все аргументы имеют значение NULL, то и результат будет NULL.

Примеры COALESCE 

Example 346. Использование функции COALESCE

В данном примере предпринимается попытка использовать все имеющиеся данные для составления полного имени. Выбирается поле NICKNAME из таблицы PERSONS. Если оно имеет значение NULL, то берётся значение из поля FIRSTNAME. Если и оно имеет значение NULL, то используется строка “‘Mr./Mrs.’”. Затем к значению функции COALESCE добавляется фамилия (поле LASTNAME). Обратите внимание, что эта схема нормально работает, только если выбираемые поля имеют значение NULL или не пустое значение: если одно из них является пустой строкой, то именно оно и возвратится в качестве значения функции COALESCE.

```
SELECT COALESCE(PE.NICKNAME, PE.FIRSTNAME, 'Mr./Mrs.') || ' ' || PE.LASTNAME AS FULLNAME
FROM PERSONS PE
```

Example 347. Использование функции COALESCE с агрегатными функциями

В данном примере в случае получения при суммировании значения NULL запрос вернёт 0.

```
SELECT coalesce (sum (q), 0)
```
FROM bills
WHERE ...

См. также:
CASE.

**8.12.2. DECODE()**

Доступно в
DSQL, PSQL

Синтаксис

```
DECODE(<testexpr>,
    <expr1>, <result1>
    [<expr2>, <result2> …]
    [, <defaultresult>])
```

эквивалентная конструкция CASE

```
CASE <testexpr>
    WHEN <expr1> THEN <result1>
    [WHEN <expr2> THEN <result2> …]
    [ELSE <defaultresult>]
END
```

**Table 203. Параметры функции DECODE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>testexpr</td>
<td>Выражения любого совместимого типа, которое сравнивается с выражениями &lt;expr1&gt;, &lt;expr2&gt; ... &lt;exprN&gt;</td>
</tr>
<tr>
<td>expr1, expr2, ... exprN</td>
<td>Выражения любого совместимого типа, с которыми сравнивают с выражением &lt;testexpr&gt;.</td>
</tr>
<tr>
<td>result1, result2, ... resultN</td>
<td>Возвращаемые выражения любого типа.</td>
</tr>
<tr>
<td>defaultresult</td>
<td>Выражение, возвращаемое если ни одно из условий не было выполнено.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

зависит от типов входных аргументов

Данная функция эквивалентна конструкции Простой CASE, в которой заданное выражение сравнивается с другими выражениями до нахождения совпадения. Результатом является значение, указанное после выражения, с которым найдено совпадение. Если совпадений не найдено, то возвращается значение по умолчанию (если оно, конечно, задано— в
противном случае возвращается NULL).

Совпадение эквивалентно оператору ‘=’, т.е. если testexpr имеет значение NULL, то он не соответствует ни одному из expr, даже тем, которые имеют значение NULL.

Примеры DECODE

**Example 348. Использование функции DECODE**

```sql
select name,
       age,
       decode(upper(sex),
               'M', 'Male',
               'F', 'Female',
               'Unknown'),
       religion
from people
```

См. также:

**CASE.**

8.12.3. **IIF()**

Доступно в DSQL, PSQL

Синтаксис

```
IIF (<condition>, ResultT, ResultF)
```

**Table 204. Параметры функции IIF**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Выражение логического типа.</td>
</tr>
<tr>
<td>resultT</td>
<td>Возвращаемое значение, если condition является истинным.</td>
</tr>
<tr>
<td>resultF</td>
<td>Возвращаемое значение, если condition является ложным.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата зависит от типов входных аргументов

Функция IIF имеет три аргумента. Если первый аргумент является истинной, то результатом будет второй параметр, в противном случае результатом будет третий параметр.

Оператор IIF также можно сравнить в тройным оператором “?:” в C-подобных языках.
По сути, функция IIF это короткая запись оператора CASE

```
CASE WHEN <condition> THEN resultT ELSE resultF END
```

**Примеры IIF**

*Example 349. Использование функции IIF*

```
SELECT IIF(SEX = 'M', 'Sir', 'Madam') FROM CUSTOMERS
```

**См. также:**

CASE.

**8.12.4. MAXVALUE()**

*Доступно в* DSQL, PSQL

*Синтаксис*

```
MAXVALUE (<expr1> [, ... , <exprN> ])
```

*Table 205. Параметры функции MAXVALUE*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1 ... exprN</td>
<td>Выражения любого совместимого типа.</td>
</tr>
</tbody>
</table>

*Тип возвращаемого результата:* тот же что и первый аргумент функции expr1

Возвращает максимальное значение из входного списка чисел, строк или параметров с типом DATE/TIME/TIMESTAMP.

Если один или более входных параметров имеют значение NULL, то результатом функции MAXVALUE тоже будет NULL в отличие от агрегатной функции MAX.

**Примеры MAXVALUE**

*Example 350. Использование функции MAXVALUE*

```
SELECT MAXVALUE(PRICE_1, PRICE_2) AS PRICE
FROM PRICELIST
```
См. также:
MINVALUE().

8.12.5. MINVALUE()

Доступно в
DSQL, PSQL

Синтаксис

MINVALUE (<expr1> [, ... , <exprN> ])

Table 206. Параметры функции MINVALUE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1 ... exprN</td>
<td>Выражения любого совместимого типа.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
тот же что и первый аргумент функции expr1

Возвращает минимальное значение из входного списка чисел, строк или параметров с типом DATE/TIME/TIMESTAMP.

Если один или более входных параметров имеют значение NULL, то результатом функции MINVALUE тоже будет NULL в отличие от агрегатной функции MIN.

Примеры MINVALUE

Example 351. Использование функции MINVALUE

```
SELECT MINVALUE(PRICE_1, PRICE_2) AS PRICE
FROM PRICELIST
```

См. также:
MAXVALUE().

8.12.6. NULLIF()

Доступно в
DSQL, PSQL

Синтаксис

```
NULLIF (<exp1>, <exp2>)
```
### Table 207. Параметры функции NULLIF

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1, expr2</td>
<td>Выражения любого совместимого типа.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

Зависит от типов входных аргументов

Функция возвращает значение первого аргумента, если он неравен второму. В случае равенства аргументов возвращается NULL.

**Примеры NULLIF**

**Example 352. Использование функции NULLIF**

```sql
SELECT AVG(NULLIF(weight, -1)) FROM cargo;
```

Этот запрос возвращает среднее значение поля `weight` по таблице, за исключением строк, где он не указан (равен -1). Если бы не было этой функции простой оператор `avg(weight)` вернул бы некорректное значение.

**См. также:**

`COALESCE()`, `CASE`.

## 8.13. Другие функции

В этом разделе расположены функции, которые сложно отнести к какой-либо категории.

### 8.13.1. MAKE_DBKEY()

**Доступно в**

DSQL, PSQL

**Синтаксис**

```sql
MAKE_DBKEY (<relation>, recnum [, dpnum [, ppnum]])
```

<relation> ::= rel_name | rel_id

**Table 208. Параметры функции MAKE_DBKEY**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>rel_name</td>
<td>Имя таблицы.</td>
</tr>
<tr>
<td>rel_id</td>
<td>Идентификатор таблицы. Можно найти в RDB$RELATIONS.RDB$RELATION_ID.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>recnum</td>
<td>Номер записи. Либо абсолютный (если dpnum и ppnum отсутствуют), либо относительный (если dpnum присутствует)</td>
</tr>
<tr>
<td>dpnum</td>
<td>Номер страницы данных DP. Либо абсолютный (если ppnum отсутствует), либо относительный (если ppnum присутствует)</td>
</tr>
<tr>
<td>ppnum</td>
<td>Номер страницы указателей на данные PP.</td>
</tr>
</tbody>
</table>

Функция `MAKE_DBKEY` создает значение DBKEY, используя имя или идентификатор таблицы, номер записи и, необязательно, логический номер страницы данных и страницы указателя.

Замечания

1. Если первый аргумент (таблица) является строковым выражением или литералом, то он обрабатывается как имя таблицы, и Firebird ищет соответствующий идентификатор таблицы. Поиск чувствителен к регистру.

   В случае строкового литерала идентификатор таблицы оценивается во время подготовки. В случае выражения, идентификатор таблицы оценивается во время выполнения.

   Если таблица не может быть найдена, возникает ошибка `isc_relnotdef`.

2. Если первый аргумент (таблица) является числовым выражением или литералом, то он обрабатывается как идентификатор таблицы и используется «как есть», без проверки существования таблицы.

   Если значение аргумента отрицательно или превышает максимально допустимый идентификатор таблицы (в настоящее время 65535), то возвращается `NULL`.

3. Второй аргумент (`recnum`) представляет собой абсолютный номер записи в отношении (если следующие аргументы — `dpnum` и `ppnum` — отсутствуют) или номер записи относительно первой записи, указанной в следующих аргументах.

4. Третий аргумент (`dpnum`) — это логический номер страницы данных (DP) в таблице (если следующий аргумент — `ppnum` — отсутствует) или номер страницы данных относительно первой страницы данных, адресованной заданным `ppnum`.

5. Четвёртый аргумент (`ppnum`) — это логический номер страницы указателя (PP) в таблице.

6. Все числа начинаются с нуля. Максимально допустимое значение для `dpnum` и `ppnum` составляет $2^{32}$ (4294967296).

   Если указан параметр `dpnum`, значение `recnum` может быть отрицательным.

   Если `dpnum` отсутствует и `recnum` отрицательно, возвращается `NULL`. 
Если указан \textit{ppnum}, то \textit{dpnum} может быть отрицательным.

Если \textit{ppnum} отсутствует и \textit{dpnum} отрицателен, возвращается \texttt{NULL}.

7. Если какой-либо из указанных аргументов имеет значение \texttt{NULL}, результат также равен \texttt{NULL}.

8. Первый аргумент (таблица) описывается как \texttt{INTEGER}, но может быть переопределен приложением как \texttt{VARCHAR} или \texttt{CHAR}.

\textit{recnum}, \textit{dpnum} и \textit{ppnum} описываются как \texttt{BIGINT} (64-разрядное целое число со знаком).

Примеры:

1. Запрос выбирает запись, используя имя таблицы (имя таблицы в верхнем регистре)

   
   ```sql
   select * from rdb$relations where rdb$db_key = make_dbkey('RDB$RELATIONS', 0)
   ```

2. Запрос выбирает запись, используя идентификатор таблицы

   ```sql
   select * from rdb$relations where rdb$db_key = make_dbkey(6, 0)
   ```

3. Запрос выбирает все записи, которые физически находятся на первой странице данных в таблице

   ```sql
   select * from rdb$relations
   where rdb$db_key >= make_dbkey(6, 0, 0)
   and rdb$db_key < make_dbkey(6, 0, 1)
   ```

4. Запрос выбирает все записи, которые физически находятся на первой странице данных 6-й страницы указателя в таблице

   ```sql
   select * from SOMETABLE
   where rdb$db_key >= make_dbkey('SOMETABLE', 0, 0, 5)
   and rdb$db_key < make_dbkey('SOMETABLE', 0, 1, 5)
   ```

8.13.2. \texttt{RDB$ERROR()}

\textit{RDB$ERROR} \texttt{(})\textit{context} \texttt{)}

Доступно в PSQL

\textbf{Синтаксис}

```
RDB$ERROR (<context>)
```
Тип возвращаемого результата
Зависит от контекста
Возвращает значение контекста активного исключения. Тип возвращаемого значения зависит от контекста.

Функция RDB$ERROR всегда возвращает NULL вне блока обработки ошибок WHEN ...

Доступные контексты в качестве аргумента функции RDB$ERROR:

**EXCEPTION**
функция возвращает имя исключения, если активно исключение определено пользователем, или NULL если активно одно из системных исключений. Для контекста EXCEPTION тип возвращаемого значения: VARCHAR(63) CHARACTER SET UTF8.

**MESSAGE**
функция возвращает интерпретированный текст активного исключения. Для контекста MESSAGE тип возвращаемого значения: VARCHAR(1024) CHARACTER SET UTF8.

**GDSCODE**
функция возвращает значение контекстной переменной GDSCODE.

**SQLCODE**
функция возвращает значение контекстной переменной SQLCODE.

**SQLSTATE**
функция возвращает значение контекстной переменной SQLSTATE.

*Example 353. Использование функции RDB$ERROR для сохранения текста ошибки в журнал*

```
...
BEGIN
...
WHEN ANY DO
  EXECUTE PROCEDURE P_LOG_EXCEPTION(RDB$ERROR(MESSAGE));
END
...
```

*См. также:*
WHEN ... DO, EXCEPTION, GDSCODE, SQLCODE, SQLSTATE.
**8.13.3. RDB$GET_TRANSACTION_CN()**

*Доступно в*
DSQL, PSQL

**Синтаксис**

```
RDB$GET_TRANSACTION_CN (transaction_id)
```

**Table 209. Параметры функции RDB$GET_TRANSACTION_CN**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td>Номер (идентификатор) транзакции</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата:**

BIGINT

Возвращает номер подтверждения (Commit Number) для заданной транзакции.

Внутренние механизмы Firebird используют беззнаковое 8-байтовое целое для Commit Number и беззнаковое 6-байтовое целое для номера транзакции. Поэтому, несмотря на то, что язык SQL не имеет беззнаковых целых, а RDB$GET_TRANSACTION_CN возвращает знаковый BIGINT, невозможно увидеть отрицательный номер подтверждения, за исключением нескольких специальных значений, используемых для неподтверждённых транзакций.

Если функция RDB$GET_TRANSACTION_CN возвращает значение больше 1, то это фактический (Commit Number) транзакции, то есть эта транзакция была зафиксирована после запуска базы данных.

В остальных случаях функция может возвращать одно из следующих результатов, указывающих статус фиксации транзакции:

-2  — мёртвые транзакции (отмененные);
-1  — зависшие транзакции (в состоянии limbo 2PC транзакций);
0   — активные транзакции;
1   — для транзакций подтверждённых до старта базы данных или с номером меньше чем OIT (Oldest Interesting Transaction);
NULL — если номер транзакции равен NULL или больше чем Next Transaction.

*Example 354. Использование RDB$GET_TRANSACTION_CN*

```
select rdb$get_transaction_cn(current_transaction) from rdb$database;
select rdb$get_transaction_cn(123) from rdb$database;
```
За более детальной информацией о Commit Number, обратитесь к Firebird 4.0 Release Notes.

8.13.4. RDB$ROLE_IN_USE()

Доступно в
DSQL, PSQL

Синтаксис

```
RDB$ROLE_IN_USE (role_name)
```

Таблица 210. Параметры функции RDB$ROLE_IN_USE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_name</td>
<td>Имя роли использования которой проверяется</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

`BOOLEAN`

Функция RDB$ROLE_IN_USE возвращает используется ли роль текущим пользователем.

Данная функция позволяет проверить использование любой роли: указанной явно (при входе в систему или изменённой с помощью оператора `SET ROLE`) и назначенной неявно (роли назначенные пользователю с использованием предложения `DEFAULT`).

Example 355. Использование функции RDB$ROLE_IN_USE

```
-- Проверяем используется ли явно назначенная или  
-- неявно полученная роль MANAGER
IF (RDB$ROLE_IN_USE('MANAGER')) THEN
BEGIN
...
END
```

Example 356. Список ролей используемых текущим подключением

```
SELECT * FROM RDB$ROLES WHERE RDB$ROLE_IN_USE(RDB$ROLE_NAME)
```

См. также:
GRANT ROLE, SET ROLE, CURRENT_ROLE.
### 8.13.5. RDB$SYSTEM_PRIVILEGE()

**Доступно в**
DSQL, PSQL

**Синтаксис**

```
RDB$SYSTEM_PRIVILEGE (<privilege>)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>Проверяемая системная привилегия</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

BOOLEAN

Функция RDB$SYSTEM_PRIVILEGE используется системная привилегия текущим соединением. Список системных привилегий см. в CREATE ROLE.

**Example 357. Использование функции RDB$SYSTEM_PRIVILEGE**

```
SELECT RDB$SYSTEM_PRIVILEGE(USER_MANAGEMENT) FROM RDB$DATABASE;
```

**См. также:**
CREATE ROLE.
Агрегатные функции

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение. Агрегатные функции, за исключением COUNT, не учитывают значения NULL. Агрегатные функции часто используются совместно с предложением GROUP BY.

Агрегатные функции могут быть использованы в качестве выражений только в следующих случаях:

- Список выбора инструкции SELECT (вложенный или внешний запрос);
- Предложение HAVING.

Синтаксис агрегатных функций

```
<aggregate_function> ::= 
    aggregate_function ([ALL | DISTINCT] <expr>)
    [FILTER (WHERE <condition>)]
```

Агрегатные функции также могут использоваться как оконные с предложением OVER (). Подробнее смотри в Оконные (Аналитические) функции.

9.1. Предложение FILTER

Предложение FILTER расширяет агрегатные функции дополнительным предложением WHERE. Если используется предложение FILTER, то результат агрегата строится только из строк, которые также удовлетворяют условию в дополнительном предложении WHERE.

Как правило, предложение фильтра может быть реализовано с использованием выражения CASE внутри агрегатной функции: условие фильтра должно быть помещено в предложение WHEN, значение, которое должно быть агрегировано в предложение THEN. Поскольку агрегатные функции обычно пропускают значения NULL, неявное предложение ELSE NULL достаточно, чтобы игнорировать не подходящие под условия фильтрации строки. Следующие два выражения эквивалентны:

```
SUM(<expression>) FILTER(WHERE <condition>)
```

и

```
SUM(CASE WHEN <condition> THEN <expression> END)
```

Для COUNT(*) этот пример выглядит иначе, потому что выражение “*” не может быть использовано в предложении THEN. Вместо этого обычно используется любое константное значение не равное NULL.
**COUNT(*) FILTER(WHERE <condition>)**

и

**SUM(CASE WHEN <condition> THEN 1 END)**

**Примеры FILTER**

**Example 358. Использование предложения FILTER**

```sql
SELECT
    invoice_year,
    SUM(revenue) FILTER (WHERE invoice_month = 1) AS jan_revenue,
    SUM(revenue) FILTER (WHERE invoice_month= 2) AS feb_revenue,
    ...
    SUM(revenue) FILTER (WHERE invoice_month = 12) AS dec_revenue
FROM (SELECT
    EXTRACT(YEAR FROM invoices.invoice_date) AS invoice_year,
    EXTRACT(MONTH FROM invoices.invoice_date) AS invoice_month,
    invoices.revenue AS revenue
FROM invoices )
GROUP BY invoice_year
```

**9.2. Основные агрегатные функции**

**9.2.1. AVG()**

**Доступно в**

DSQL

**Синтаксис**

```sql
AVG([[ALL | DISTINCT] <expr>])
```

**Table 212. Параметры функции AVG**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>
Функция `AVG` возвращает среднее значение для группы. Значения `NULL` пропускаются.

- Параметр `ALL` (по умолчанию) применяет агрегатную функцию ко всем значениям.
- Параметр `DISTINCT` указывает на то, что функция `AVG` будет выполнена только для одного экземпляра каждого уникального значения, независимо от того, сколько раз встречается это значение.
- В случае если выборка записей пустая или содержит только значения `NULL`, результат будет содержать `NULL`.

**Примеры AVG**

*Example 359. Использование функции AVG*

```
SELECT
department,  
AVG(salary)
FROM employee  
GROUP BY department
```

*См. также:*

`SELECT`.

### 9.2.2. `COUNT()`

**Доступно в DSQL**

**Синтаксис**

```
COUNT([ALL | DISTINCT] expr) | *
```

*Table 213. Параметры функции COUNT*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

`BIGINT`

Функция `COUNT` возвращает количество значений в группе, которые не являются `NULL`. 
• По умолчанию используется ALL: функция просто считает все значения в наборе, которые не равны NULL.

• Если указан DISTINCT дубликаты исключаются из подсчитываемого набора.

• Если вместо выражения $expr$ указано $COUNT(*)$, будут подсчитаны все записи.
  ◦ не может использоваться с ключевым словом DISTINCT
  ◦ дубликаты записей не исключаются
  ◦ при этом учитываются записи содержащие NULL

• Для пустой выборки данных или если при выборке окажутся одни значения, содержащие NULL, функция возвратит значение равное 0.

Примеры $COUNT$

Example 360. Использование функции $COUNT$

```sql
SELECT
dept_no,
  $COUNT(*)$ AS cnt,
  $COUNT(DISTINCT name)$ AS cnt_name
FROM employee
GROUP BY dept_no
```

См. также:

$SELECT$.

9.2.3. $LIST()$

Доступно в $DSQL$

Синтаксис

$LIST([ALL | DISTINCT] <expr> [, separator])$

Table 214. Параметры функции $LIST$

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>$expr$</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает строковый тип данных или BLOB. Поля типа дата / время и числовые преобразуются к строке. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>separator</td>
<td>Разделитель. Выражение строкового типа. По умолчанию разделителем является запятая.</td>
</tr>
</tbody>
</table>
Тип возвращаемого результата

**BLOB**

Функция LIST возвращает строку, состоящую из значений аргумента, отличных от NULL в группе, разделенных запятой или заданным пользователем разделителем. Если нет значений, отличных от NULL (включая случай, когда группа пуста), возвращается NULL.

- **ALL** (по умолчанию) приводит к обработке всех значений, отличных от NULL. Если указано ключевое слово DISTINCT, то дубликаты удаляются, за исключением случаев, когда expr является BLOB.

- Необязательный аргумент separator может быть любым строковым выражением. Это позволяет указать, например, ascii_char (13) в качестве разделителя.

- Аргументы expr и separator поддерживают BLOB любого размера и набора символов.

- Дата / время и числовые аргументы неявно преобразуются в строки перед объединением.

- Результатом функции является текстовый BLOB, кроме случаев, когда expr является BLOB другого подтипа.

- Порядок значений в списке не определен — порядок, в котором строки объединяются, определяется порядком чтения из исходного набора данных. Для таблиц такой порядок обычно не определяется. Если порядок важен, исходные данные можно предварительно отсортировать используя производную таблицы или аналогичное средство.

**Примеры LIST**

**Example 361. Использование функции LIST**

Получение списка, порядок не определён.

```sql
SELECT LIST (display_name, '; ') FROM GR_WORK;
```

**Example 362. Использование функции LIST с заданным порядком**

Получение списка в алфавитном порядке.

```sql
SELECT LIST (display_name, '; ') FROM (SELECT display_name FROM GR_WORK ORDER BY display_name);
```

См. также:

```sql
SELECT.
```
9.2.4. MAX()

Доступно в
DSQL

Синтаксис

MAX([ALL | DISTINCT] <expr>)

Table 215. Параметры функции MAX

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
тот же что и аргумент функции expr.

Функция MAX возвращает максимальный элемент выборки, которые не равны NULL.

• Если группа пуста или содержит только NULL, результатом будет NULL.
• Если входным аргументом является строка, то функция вернет значение, которое будет последним в сортировке с использованием соответствующего COLLATE.
• Эта функция полностью поддерживает текстовые BLOB любого размера и набора символов.

Информация: Параметр DISTINCT не имеет смысла при использовании функцией MAX и доступен только для совместимости со стандартом.

Примеры MAX

Example 363. Использование функции MAX

```sql
SELECT
department,
MAX(salary)
FROM employee
GROUP BY department
```

См. также:
SELECT, MIN().

9.2.5. MIN()

Доступно в
Функция MIN возвращает минимальный элемент выборки, которые не равны NULL.

- Если группа пуста или содержит только NULL, результатом будет NULL.
- Если входным аргументом является строка, то функция вернет значение, которое будет первым в сортировке с использованием соответствующего COLLATE.
- Эта функция полностью поддерживает текстовые BLOB любого размера и набора символов.

Параметр DISTINCT не имеет смысла при использовании функцией MIN и доступен только для совместимости со стандартом.

Примеры MIN

Example 364. Использование функции MIN

```sql
SELECT dept_no, 
MIN(salary) 
FROM employee 
GROUP BY dept_no
```

См. также:
SELECT, MAX().

9.2.6. SUM()

Доступно в
DSQL
Синтаксис

\[
\text{SUM([ALL | DISTINCT] <expr>)}
\]

### Table 217. Параметры функции SUM

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION, DECFLOAT или масштабируемое целое (INTEGER, BIGINT или INT128) в зависимости от типа аргумента функции expr. Обычно, если это возможно, выбирается тип с большей вместимостью, чем тип выражения expr.

Функция SUM возвращает сумму элементов выборки, которые не равны NULL.

- **ALL** является опцией по умолчанию — обрабатываются все значения из выборки, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты, после чего осуществляется суммирование.
- При пустой выборке, или при выборке из одних NULL функция возвратит NULL.

**Примеры SUM**

**Example 365. Использование функции SUM**

```sql
SELECT dept_no,
       SUM(salary) AS sum_salary
FROM employee
GROUP BY dept_no
```

*См. также:*

SELECT.

### 9.3. Статистические функции

Статистические функции являются агрегатными функциями. Эти функции не учитывают значения NULL. К аргументу статистической функции не применимы параметры ALL и DISTINCT.

Статистические функции часто используются совместно с предложением GROUP BY.
9.3.1. CORR()

Доступно в DSQL

Синтаксис

CORR(<expr1>, <expr2>)

Таблица 218. Параметры функции CORR

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1, expr2</td>
<td>Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
DOUBLE PRECISION

Функция CORR возвращает коэффициент корреляции для пары выражений, возвращающих числовые значения.

Функция CORR(<expr1>, <expr2>) эквивалентна

\[
\frac{\text{COVAR}_{\text{POP}}(\text{expr1}, \text{expr2})}{\text{STDDEV}_{\text{POP}}(\text{expr2}) \times \text{STDDEV}_{\text{POP}}(\text{expr1})}
\]

В статистическом смысле, корреляция — это степень связи между переменными. Связь между переменными означает, что значение одной переменной можно в определённой степени предсказать по значению другой. Коэффициент корреляции представляет степень корреляции в виде числа в диапазоне от -1 (высокая обратная корреляция) до 1 (высокая корреляция). Значение 0 соответствует отсутствию корреляции.

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры CORR

Example 366. Использование функции CORR

```sql
SELECT COVAR_POP(aheight, alength) AS c_corr
FROM measure
```

См. также: COVAR_POP(), STDDEV_POP().
9.3.2. COVAR_POP()

Доступно в
DSQL

Синтаксис

COVAR_POP(<expr1>, <expr2>)

Table 219. Параметры функции COVAR_POP

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1, expr2</td>
<td>Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
DOUBLE PRECISION

Функция COVAR_POP возвращает ковариацию совокупности (population covariance) пар выражений с числовыми значениями.

Функция COVAR_POP(expr1, expr2) эквивалентна

\[
\frac{(\text{SUM}(\text{<expr1>}) \cdot \text{<expr2>}) - \text{SUM}(\text{<expr1>}) \cdot \text{SUM}(\text{<expr2>}) / \text{COUNT(*)})}{\text{COUNT(*)}}
\]

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры COVAR_POP

Example 367. Использование функции COVAR_POP

```sql
SELECT
    COVAR_POP(alength, aheight) AS c_corr
FROM measure
```

См. также:

COVAR_SAMP(), SUM(), COUNT().

9.3.3. COVAR_SAMP()

Доступно в
DSQL
Table 220. Параметры функции COVAR_SAMP

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1, expr2</td>
<td>Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION

Функция COVAR_SAMP возвращает выборочную ковариацию (sample covariance) пары выражений с числовыми значениями.

Функция $\text{COVAR}_S\text{AMP}(\text{expr1}, \text{expr2})$ эквивалентна

\[
\frac{(\text{SUM}(<\text{expr1}>*<\text{expr2}>) - \text{SUM}(<\text{expr1}>)\text{SUM}(<\text{expr2}>) / \text{COUNT}(*))}{\text{COUNT}(*) - 1}
\]

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

**Примеры COVAR_SAMP**

**Example 368. Использование функции COVAR_SAMP**

```sql
SELECT 
    COVAR_SAMP(alength, aheight) AS c_corr
FROM measure
```

См. также:

COVAR_POP(), SUM(), COUNT().

**9.3.4. STDDEV_POP()**

**Доступно в**

DSQL

**Синтаксис**

```sql
STDDEV_POP(<expr>)
```
Таблица 221. Параметры функции STDDEV_POP

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

DOUBLE PRECISION или NUMERIC в зависимости от типа expr.

Функция STDDEV_POP возвращает среднеквадратичное отклонение для группы. Значения NULL пропускаются.

Функция STDDEV_POP(expr) эквивалентна

\[ \sqrt{VAR_POP(expr)} \]

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры STDDEV_POP

Example 369. Использование функции STDDEV_POP

```sql
SELECT dept_no, STDDEV_POP(salary) FROM employee GROUP BY dept_no
```

См. также:

STDDEV_POP(), VAR_POP().

9.3.5. STDDEV_SAMP()

Доступно в DSQL

Синтаксис

STDDEV_SAMP(expr)

Таблица 222. Параметры функции STDDEV_SAMP
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
DOUBLE PRECISION или NUMERIC в зависимости от типа expr

Функция STDDEV_SAMP возвращает стандартное отклонение для группы. Значения NULL пропускаются.

Функция STDDEV_SAMP(expr) эквивалентна

\[ \text{SQRT(VAR_SAMP(expr))} \]

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

Примеры STDDEV_SAMP

Example 370. Использование функции STDDEV_SAMP

```sql
SELECT dept_no,
       STDDEV_SAMP(salary)
FROM employee
GROUP BY dept_no
```

См. также:
STDDEV_POP(), VAR_SAMP().

9.3.6. VAR_POP()

Доступно в
DSQL

Синтаксис

VAR_POP(expr)

Table 223. Параметры функции VAR_POP
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION или NUMERIC в зависимости от типа `expr`

Функция `VAR_POP` возвращает выборочную дисперсию для группы. Значения NULL пропускаются.

Функция `VAR_POP(<expr>)` эквивалентна

\[
\frac{(\text{SUM}(<\text{expr}> * <\text{expr}>) - \text{SUM}(<\text{expr}>) * \text{SUM}(<\text{expr}>) / \text{COUNT}(<\text{expr}>)}}{\text{COUNT}(<\text{expr}>)}
\]

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

**Примеры VAR_POP**

*Example 371. Использование функции VAR_POP*

```sql
SELECT
department_number,
VAR_POP(salary)
FROM employee
GROUP BY department_number
```

*См. также:*

VAR_SAMP(), SUM(), COUNT().

**9.3.7. VAR_SAMP()**

Доступно в DSQL

**Синтаксис**

```
VAR_SAMP(<expr>)
```

*Table 224. Параметры функции VAR_SAMP*
### Параметр Описание

| expr | Выражение возвращает числовой тип данных. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются. |

**Тип возвращаемого результата**

DOUBLE PRECISION или NUMERIC в зависимости от типа expr

Функция VAR_SAMP возвращает несмещённую выборочную дисперсию для группы. Значения NULL пропускаются.

Функция VAR_SAMP(<expr>) эквивалентна

\[
\frac{(\text{SUM}(\text{expr} \times \text{expr}) - \text{SUM}(\text{expr}) \times \text{SUM}(\text{expr}) / \text{COUNT}(\text{expr}))}{\text{COUNT}(\text{expr}) - 1}
\]

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

**Примеры VAR_SAMP**

*Example 372. Использование функции VAR_SAMP*

```sql
SELECT dept_no,
       VAR_SAMP(salary)
FROM employee
GROUP BY dept_no
```

См. также:

VAR_POP(), SUM(), COUNT().

### 9.4. Функции линейной регрессии

Функции линейной регрессии полезны для продолжения линии тренда. Линия тренда — это, как правило, закономерность, которой придерживается набор значений. Линия тренда полезна для прогнозирования будущих значений. Этот означает, что тренд будет продолжаться и в будущем. Для продолжения линии тренда необходимо знать угол наклона и точку пересечения с осью Y. Набор линейных функций включает функции для вычисления этих значений.

В синтаксисе функций, у интерпретируется в качестве переменной, зависящей от х.
9.4.1. REGR_AVGX()

Доступно в DSQL

Синтаксис

```
REGR_AVGX (<y>, <x> )
```

Функция `REGR_AVGX` вычисляет среднее независимой переменной линии регрессии. Функция `REGR_AVGX(<y>, <x>)` эквивалента

```
SUM(<exprX>) / REGR_COUNT(<y>, <x>)
```

где

```
<exprX> ::= CASE WHEN <x> IS NOT NULL AND <y> IS NOT NULL THEN <x> END
```

Функции `REGR_COUNT()` и `REGR_AVGY()`.

9.4.2. REGR_AVGY()

Доступно в DSQL

Синтаксис

```
REGR_AVGY(<y>, <x>)
```

Функция `REGR_AVGY` вычисляет среднее зависимой переменной линии регрессии. Функция `REGR_AVGY(<y>, <x>)` эквивалента

```
SUM(<exprY>) / REGR_COUNT(<y>, <x>)
```

где

```
<exprY> ::= CASE WHEN <y> IS NOT NULL AND <x> IS NOT NULL THEN <y> END
```

Функции `REGR_COUNT()` и `REGR_AVGX()`.
### Table 227. Parameters of function `REGR_COUNT`

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Функция `REGR_AVGY` вычисляет среднее зависимой переменной линии регрессии.

Функция `REGR_AVGY(y, x)` эквивалентна

\[
\frac{\text{SUM}(\text{exprY})}{\text{REGR_COUNT}(y, x)}
\]

где

\[
\text{exprY} ::= \text{CASE WHEN } x \text{ IS NOT NULL AND } y \text{ IS NOT NULL THEN } y \text{ END}
\]

См. также:

- `REGR_COUNT()`, `REGR_AVGX()`

#### 9.4.3. REGR_COUNT()

**Доступно в**

DSQL

**Синтаксис**

\[
\text{REGR_COUNT}(y, x)
\]
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**
BIGINT

Функция REGR_COUNT возвращает количество не пустых пар, используемых для создания линии регрессии.

Функция REGR_COUNT(<y>, <x>) эквивалентна

```
SUM(CASE WHEN <x> IS NOT NULL AND <y> IS NOT NULL THEN 1 END)
```

См. также: SUM()

**9.4.4. REGR_INTERCEPT()**

**Доступно в**
DSQL

**Синтаксис**

```
REGR_INTERCEPT(<y>, <x>)
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>x</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**
DOUBLE PRECISION

Функция REGR_INTERCEPT вычисляет точку пересечения линии регрессии с осью Y.

Функция REGR_INTERCEPT(<y>, <x>) эквивалентна
Примеры \texttt{REGR\_INTERCEPT}

\textit{Example 373. Прогнозирование объёмов продаж}

\begin{verbatim}
WITH RECURSIVE years(byyear) AS
    (SELECT 1991 FROM rdb$database UNION ALL
    SELECT byyear+1 FROM years WHERE byyear < 2020
    ),
    s AS
    (SELECT EXTRACT(YEAR FROM order_date) AS byyear,
    SUM(total_value) AS total_value
    FROM sales GROUP BY 1
    ),
    regr AS
    (SELECT REGR\_INTERCEPT(total_value, byyear) as intercept,
    REGR\_SLOPE(total_value, byyear) as slope
    FROM s)
SELECT years.byyear AS byyear,
    intercept + (slope * years.byyear) AS total_value
FROM years CROSS JOIN regr
\end{verbatim}

\begin{tabular}{lrr}
\hline
\textbf{BYYEAR} & \textbf{TOTAL\_VALUE} \\
\hline
1991 & 118377,35 \\
1992 & 414557,62 \\
1993 & 710737,89 \\
1994 & 1006918,16 \\
1995 & 1303098,43 \\
1996 & 1599278,69 \\
1997 & 1895458,96 \\
1998 & 2191639,23 \\
1999 & 2487819,50 \\
2000 & 2783999,77 \\
\ldots & \\
\hline
\end{tabular}

См. также:
\texttt{REGR\_AVGY()}, \texttt{REGR\_AVGX()}, \texttt{REGR\_SLOPE()}. 

\textbf{9.4.5. REGR\_R2()}

Доступно в
DSQL
**Синтаксис**

REGR_R2(<y>, <x>)

**Table 229. Параметры функции REGR_R2**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>x</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION

Функция REGR_R2 вычисляет коэффициент детерминации, или R-квадрат, линии регрессии.

Функция REGR_R2(<y>, <x>) эквивалентна

\[ \text{POWER(CORR(<y>, <x>), 2)} \]

См. также:

CORR(), POWER()

**9.4.6. REGR_SLOPE()**

**Доступно в**

DSQL

**Синтаксис**

REGR_SLOPE(<y>, <x>)

**Table 230. Параметры функции REGR_SLOPE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>x</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION

Функция `REGR_SLOPE` вычисляет угол наклона линии регрессии.

Функция `REGR_SLOPE(<y>, <x>)` эквивалентна

```latex
covar_pop(<y>, <x>) / var_pop(<exprX>)
<exprX> ::= case when <x> is not null and <y> is not null then <x> end
```

см. также:

`COVAR_POP()`, `VAR_POP()`.

**9.4.7. REGR_SXX()**

Доступно в

DSQL

**Синтаксис**

`REGR_SXX(<y>, <x>)`

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>x</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION
Диагностическая статистика, используемая для анализа регрессии.

Функция \( \text{REGR}_\text{SXX}(\text{y}, \text{x}) \) вычисляется следующим образом:

\[
\text{REGR}_\text{COUNT}(\text{y}, \text{x}) \times \text{VAR}\_\text{POP}(<\text{exprX}>)
\]

\(<\text{exprX}> :=
\quad \text{CASE WHEN} \quad \text{x IS NOT NULL AND} \quad \text{y IS NOT NULL THEN} \quad \text{x END}
\]

См. также:
\( \text{REGR}\_\text{COUNT}(), \text{VAR}\_\text{POP}(). \)

### 9.4.8. \( \text{REGR}_\text{SXY}() \)

Доступно в DSQL

**Синтаксис**

\[
\text{REGR}_\text{SXY}(\text{y}, \text{x})
\]

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{y} )</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>( \text{x} )</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

DOUBLE PRECISION

Диагностическая статистика, используемая для анализа регрессии.

Функция \( \text{REGR}_\text{SXY}(\text{y}, \text{x}) \) вычисляется следующим образом:

\[
\text{REGR}_\text{COUNT}(\text{y}, \text{x}) \times \text{COVAR}\_\text{POP}(\text{y}, \text{x})
\]

См. также:
\( \text{REGR}\_\text{COUNT}(), \text{COVAR}\_\text{POP}(). \)
9.4.9. REGR_SYY()

Доступно в
DSQL

Синтаксис

REGR_SYY(<y>, <x>)

Table 233. Параметры функции REGR_SYY

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;y&gt;</td>
<td>Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>&lt;x&gt;</td>
<td>Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
DOUBLE PRECISION

Диагностическая статистика, используемая для анализа регрессии.

Функция REGR_SYY(<y>, <x>) вычисляется следующим образом:

\[
\text{REGR_COUNT}(<y>, <x>) \times \text{VAR_POP}(<\text{exprY}>)
\]

\[
<\text{exprY}> :=
\begin{cases}
\text{CASE WHEN } <x> \text{ IS NOT NULL AND } <y> \text{ IS NOT NULL THEN } <y> \end{cases}
\]

См. также:
REGR_COUNT(), VAR_POP().
Chapter 10. Оконные (аналитические) функции

Согласно SQL спецификации оконные функции (также известные как аналитические функции) являются своего рода агрегатными функциями, не уменьшающими степень детализации. При этом агрегированные данные выводятся вместе с неагрегированными.

Синтаксически вызов оконной функции есть указание её имени, за которым всегда следует ключевое слово OVER() с возможными аргументами внутри скобок. В этом и заключается её синтаксическое отличие от обычной функции или агрегатной функции. Оконные функции могут находиться только в списке SELECT и предложении ORDER BY.

Предложение OVER может содержать разбивку по группам ("секционирование"), сортировку и рамку окна.

Доступно в
DSQL

Синтаксис

```
<window_function> ::=  
    <aggregate_function> OVER <window_name_or_spec>  
    | <window_function_name> ([<expr> [, <expr> ...]]) OVER <window_name_or_spec>

>window_name_or_spec> ::=  
    <window_specification> | window_name

>window_function_name> ::=  
    <ranking_function>  
    | <navigation_function>

>window_specification> ::=  
    ([window_name] [window_partition] [window_order] [window_frame])

>window_partition> ::= PARTITION BY <expr> [, <expr> ...]

>window_order> ::=  
    ORDER BY <expr> [<direction>] [<nulls_placement>] 
    [, <expr> [<direction>] [<nulls_placement>] ...]

<direction> ::= ASC | DESC

>nulls_placement> ::= NULLS {FIRST | LAST}

>window_frame> ::=  
    {ROWS | RANGE} <window_frame_extent>

>window_frame_extent> ::=  
```

578
<window_frame_start> ::= UNBOUNDED PRECEDING | <expr> PRECEDING | CURRENT ROW

<window_frame_between> ::= BETWEEN <window_frame_bound_1> AND <window_frame_bound_2>

<window_frame_bound_1> ::= UNBOUNDED PRECEDING | <expr> PRECEDING | CURRENT ROW | <expr> FOLLOWING

<window_frame_bound_2> ::= <expr> PRECEDING | CURRENT_ROW | <expr> FOLLOWING | UNBOUNDED FOLLOWING

<aggregate_function> ::= Агрегатные функции

<ranking_function> ::= DENSE_RANK | RANK | PERCENT_RANK | CUME_DIST | NTILE | ROW_NUMBER

<navigation_function> ::= LEAD | LAG | FIRST_VALUE | LAST_VALUE | NTH_VALUE

<query-spec> ::= SELECT [<first-clause>] [<skip-clause>] [<distinct-clause>] <select-list> <from-clause> [<where-clause>] [<group-clause>] [<having-clause>] [<named-windows-clause>] [<order-clause>] [<rows-clause>] [<offset-clause>] [<limit clause>] [<plan-clause>]

<named-windows-clause> ::= WINDOW <window-definition> [, <window-definition>] ...
**10.1. Агрегатные функции**

Все агрегатные функции могут быть использованы в качестве оконных функций, при добавлении предложения OVER.

Допустим, у нас есть таблица EMPLOYEE со столбцами ID, NAME и SALARY. Нам необходимо показать для каждого сотрудника, соответствующую ему заработную плату и процент от фонда заработной платы.

Простым запросом это решается следующим образом:

```sql
select
    id,
    department,
    salary,
    salary / (select sum(salary) from employee) percentage
from employee
order by id;
```

**Результат**

<table>
<thead>
<tr>
<th>id</th>
<th>department</th>
<th>salary</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R &amp; D</td>
<td>10.00</td>
<td>0.2040</td>
</tr>
<tr>
<td>2</td>
<td>SALES</td>
<td>12.00</td>
<td>0.2448</td>
</tr>
</tbody>
</table>
Запрос повторяется и может работать довольно долго, особенно если EMPLOYEE является сложным представлением.

Этот запрос может быть переписан в более быстрой и элегантной форме с использованием оконных функций:

```sql
SELECT id, department, salary, salary / sum(salary) OVER () as percentage
FROM employee
ORDER BY id;
```

Здесь `sum(salary) OVER ()` вычисляет сумму всех зарплат из запроса (таблицы сотрудников).

### 10.2. Секционирование

Как и для агрегатных функций, которые могут работать отдельно или по отношению к группе, оконные функции тоже могут работать для групп, которые называются "секциями" (partition) или разделами.

**Синтаксис**

```sql
<window function>(...) OVER (PARTITION BY <expr> [, <expr> ...])
```

Для каждой строки, оконная функция обсчитывает только строки, которые попадают в то же самую секцию, что и текущая строка.

Агрегирование над группой может давать более одной строки, таким образом, к результирующему набору, созданному секционированием, присоединяются результаты из основного запроса, используя тот же список выражений, что и для секции.

Продолжая пример с сотрудниками, вместо того чтобы считать процент зарплаты каждого сотрудника от суммарной зарплаты сотрудников, посчитаем процент от суммарной зарплаты сотрудников того же отдела:

**Example 374. Секционирование в OVER**

```sql
SELECT id, department, salary,
```
salary / sum(salary) OVER (PARTITION BY department) percentage
from employee
order by id;

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>department</th>
<th>salary</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R &amp; D</td>
<td>10.00</td>
<td>0.3448</td>
</tr>
<tr>
<td>2</td>
<td>SALES</td>
<td>12.00</td>
<td>0.6000</td>
</tr>
<tr>
<td>3</td>
<td>SALES</td>
<td>8.00</td>
<td>0.4000</td>
</tr>
<tr>
<td>4</td>
<td>R &amp; D</td>
<td>9.00</td>
<td>0.3103</td>
</tr>
<tr>
<td>5</td>
<td>R &amp; D</td>
<td>10.00</td>
<td>0.3448</td>
</tr>
</tbody>
</table>

10.3. Сортировка

Предложение ORDER BY может быть использовано с секционированием или без него. Предложение ORDER BY внутри OVER задаёт порядок, в котором оконная функция будет обрабатывать строки. Этот порядок не обязан совпадать с порядком вывода строк.

Для стандартных агрегатных функций, предложение ORDER BY внутри предложения OVER заставляет возвращать частичные результаты агрегации по мере обработки записей.

Example 375. Сортировка в OVER

```
SELECT
    id,
    salary,
    SUM(salary) OVER (ORDER BY salary) AS cumul_salary
FROM employee
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>cumul_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>8.00</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>17.00</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>37.00</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>37.00</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>49.00</td>
</tr>
</tbody>
</table>

В этом случае cumul_salary возвращает частичную/накопительную агрегацию (функции SUM). Может показаться странным, что значение 37.00 повторяется для идентификаторов 1 и 5, но так и должно быть. Сортировка (ORDER BY) ключей группирует их вместе, и агрегат
вычисляется единожды (но суммируя сразу два значения 10.00). Чтобы избежать этого, вы можете добавить поле ID в конце предложения ORDER BY.

Это происходит потому, что не задана рамка окна, которая по умолчанию, с указанием ORDER BY состоит из всех строк от начала раздела до текущей строки и строк, равных текущей по значению выражения ORDER BY (т.е. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW). Без ORDER BY рамка по умолчанию состоит из всех строк раздела. Подробней о рамке окна (кадра окна) будет рассказано далее.

Вы можете использовать несколько окон с различными сортировками, и дополнять предложение ORDER BY опциями ASC/DESC и NULLS {FIRST | LAST}.

С секциями предложение ORDER BY работает таким же образом, но на границе каждой секции агрегаты сбрасываются.

Все агрегатные функции могут использовать предложение ORDER BY, за исключением LIST().

Следующий пример показывает сумму кредита, накопленную сумму выплат и остаток по выплатам.

**Example 376. Использование OVER(ORDER BY …) для кумулятивных сумм**

```
SELECT
  payments.id AS id,
  payments.bydate AS bydate,
  credit.amount AS credit_amount,
  payments.amount AS pay,
  SUM(payments.amount) OVER(ORDER BY payments.bydate) AS s_amount,
  SUM(payments.amount) OVER(ORDER BY payments.bydate,
                                payments.id) AS s_amount2,
  credit.amount - SUM(payments.amount) OVER(ORDER BY payments.bydate,
                                              payments.id) AS balance
FROM credit
JOIN payments ON payments.credit_id = credit.id
WHERE credit.id = 1
ORDER BY payments.bydate
```

Результат

<table>
<thead>
<tr>
<th>ID</th>
<th>BYDATE</th>
<th>CREDIT_AMOUNT</th>
<th>PAY</th>
<th>S_AMOUNT</th>
<th>S_AMOUNT2</th>
<th>BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.01.2015</td>
<td>1000000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>900000</td>
</tr>
<tr>
<td>2</td>
<td>15.02.2015</td>
<td>1000000</td>
<td>15000</td>
<td>25000</td>
<td>25000</td>
<td>750000</td>
</tr>
<tr>
<td>3</td>
<td>15.03.2015</td>
<td>1000000</td>
<td>13000</td>
<td>40000</td>
<td>38000</td>
<td>620000</td>
</tr>
<tr>
<td>4</td>
<td>15.03.2015</td>
<td>1000000</td>
<td>20000</td>
<td>40000</td>
<td>40000</td>
<td>600000</td>
</tr>
<tr>
<td>5</td>
<td>15.04.2015</td>
<td>1000000</td>
<td>20000</td>
<td>60000</td>
<td>60000</td>
<td>400000</td>
</tr>
<tr>
<td>6</td>
<td>15.05.2015</td>
<td>1000000</td>
<td>15000</td>
<td>75000</td>
<td>75000</td>
<td>250000</td>
</tr>
<tr>
<td>7</td>
<td>15.06.2015</td>
<td>1000000</td>
<td>15000</td>
<td>100000</td>
<td>90000</td>
<td>100000</td>
</tr>
<tr>
<td>8</td>
<td>15.06.2015</td>
<td>1000000</td>
<td>10000</td>
<td>100000</td>
<td>100000</td>
<td>0</td>
</tr>
</tbody>
</table>

Chapter 10. Оконные (аналитические) функции
10.4. Рамка окна

Набор строк внутри секции, которым оперирует оконная функция, называется рамкой окна (кадры окна). Рамка окна определяет, какие строки следует учитывать для текущей строки при оценке оконной функции.

Рамка окна состоит из трёх частей: единица (unit), начальная граница и конечная граница. В качестве единицы может быть использовано ключевые слова RANGE или ROWS, которые определяют, каким образом будут работать границы окна. Границы окна определяются следующими выражениями:

- `<expr> PRECEDING`
- `<expr> FOLLOWING`
- CURRENT ROW

Предложения ROWS и RANGE требуют, чтобы было указано предложение ORDER BY. Если предложение ORDER BY отсутствует, то для агрегатных функций рамка окна состоит из всех строк в разбиении. Если задано предложение ORDER BY, то по умолчанию рамка окна состоит из всех строк, от начала разбиения до текущей строки, плюс любые следующие строки, которые равны текущей строке в соответствии с предложением ORDER BY, т.е. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Предложение ROWS ограничивает строки внутри секции путем указания фиксированного числа строк, предшествующих или следующих после текущей строки. В качестве альтернативы предложение RANGE логически ограничивает строки внутри секции путем указания диапазона значений в отношении к значению текущей строки. Предшествующие и последующие строки определяются на основании порядка, заданного в предложении ORDER BY.

- Если рамка окна задаётся с помощью предложения RANGE, то предложение ORDER BY может содержать только одно выражение и выражение должно быть числового типа, DATE, TIME или TIMESTAMP. Для `<expr> PRECEDING` выражение expr вычитается из выражения в ORDER BY, а для `<expr> FOLLOWING` — добавляется. Для CURRENT ROW выражение в ORDER BY используется как есть.

Затем все строки (внутри секции) между границами считаются частью результирующей рамки окна.

- Если рамка окна задаётся с помощью предложения ROWS, то на предложение ORDER BY не накладывается ограничений на количество и типы выражений. В этом случае фраза `<expr> PRECEDING` указывает количество строк предшествующее текущей строке, соответственно фраза `<expr> FOLLOWING` указывает количество строк после текущей строки.

UNBOUNDED PRECEDING и UNBOUNDED FOLLOWING работают одинаково для предложений ROWS и RANGE. Фраза UNBOUNDED PRECEDING указывает, что окно начинается с первой строки секции. UNBOUNDED PRECEDING может быть указано только как начальная точка окна. Фраза UNBOUNDED FOLLOWING указывает, что окно заканчивается последней строкой секции. UNBOUNDED FOLLOWING может
быть указано только как конечная точка окна.

Фраза CURRENT ROW указывает, что окно начинается или заканчивается на текущей строке при использовании совместно с предложением ROWS, или что окно заканчивается на текущем значении при использовании с предложением RANGE. CURRENT ROW может быть задана и как начальная, и как конечная точка.

Предложение BETWEEN используется совместно с ключевым словом ROWS или RANGE для указания нижней (начальной) или верхней (конечной) граничной точки окна. Верхняя граница не может быть меньше нижней границы.

Если указана только начальная точка окна, то конечной точкой окна считается CURRENT ROW. Например, если указано ROWS 1 PRECEDING, то это аналогично указанию ROWS BETWEEN 1 PRECEDING AND CURRENT ROW.

Некоторые оконные функции игнорируют выражение рамки:

- ROW_NUMBER, LAG и LEAD всегда работают как ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.
- DENSE_RANK, RANK, PERCENT_RANK и CUME_DIST работают как RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.
- FIRST_VALUE, LAST_VALUE и NTH_VALUE работают на рамке, но RANGE работает идентично ROWS.

Таким образом, предложения ROWS и RANGE позволяют довольно гибко настроить размер плавающего окна. Чаще всего встречаются следующие варианты:

- Нижняя граница фиксирована (совпадает с первой строкой упорядоченной группы), а верхняя граница ползёт (совпадает с текущей строкой упорядоченной группы). В этом случае получаем нарастающий итог (кумулятивный агрегат). В этом случае размер окна меняется (расширяется в одну сторону) и само окно движется за счёт расширения. Возможна и обратная ситуация, когда нижняя граница ползёт, а верхняя зафиксирована. В этом случае окно будет сужаться.
- Если верхняя и нижняя границы фиксированы относительно текущей строки, например 1 строка до текущей и 2 после текущей, то получаем скользящий агрегат. В этом случае размер окна фиксирован, а само окно скользит.

10.4.1. Окна диапазона

Окна диапазона объединяют строки в соответствии с заданным порядком. Например, если рамка окна задана выражением RANGE 5 PRECEDING, то будет сгенерировано перемещающееся окно, включающее предыдущие строки группы, значение которых меньше текущего не более чем на 5.

*Example 377. Использование окон диапазона*

```sql
SELECT id, salary, SUM(salary) OVER() AS s1,
```
**SUM**(salary) **OVER**(ORDER BY salary) AS s2,
**SUM**(salary) **OVER**(ORDER BY salary
  RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS s3,
**SUM**(salary) **OVER**(ORDER BY salary
  RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS s4,
**SUM**(salary) **OVER**(ORDER BY salary
  RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS s5,
**SUM**(salary) **OVER**(ORDER BY salary
  RANGE BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s6,
**SUM**(salary) **OVER**(ORDER BY salary
  RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS s7,
**SUM**(salary) **OVER**(ORDER BY salary RANGE 1 PRECEDING) AS s8

**FROM**
employee

<table>
<thead>
<tr>
<th>ID</th>
<th>SALARY</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>49.00</td>
<td>8.00</td>
<td>8.00</td>
<td>49.00</td>
<td>49.00</td>
<td>17.00</td>
<td>17.00</td>
<td>8.00</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>49.00</td>
<td>17.00</td>
<td>17.00</td>
<td>41.00</td>
<td>49.00</td>
<td>29.00</td>
<td>37.00</td>
<td>17.00</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>49.00</td>
<td>37.00</td>
<td>37.00</td>
<td>32.00</td>
<td>49.00</td>
<td>20.00</td>
<td>29.00</td>
<td>29.00</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>49.00</td>
<td>37.00</td>
<td>37.00</td>
<td>32.00</td>
<td>49.00</td>
<td>20.00</td>
<td>29.00</td>
<td>29.00</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>49.00</td>
<td>49.00</td>
<td>49.00</td>
<td>49.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
</tr>
</tbody>
</table>

Для того чтобы понять, какие значения будут входить в диапазон, можно использовать функции **FIRST_VALUE** и **LAST_VALUE**. Это помогает увидеть диапазоны окна и проверить, корректно ли установлены параметры.

**10.4.2. Окна строк**

Окна строк задаются в физических единицах, строках. Например, если рамка окна задана выражением **ROWS 5 PRECEDING**, то окно будет включать в себя до 6 строк: текущую и пять предыдущих (порядок определяется конструкцией **ORDER BY**).

**Example 378. Использование окон диапазона**

```sql
SELECT
  id,
  salary,
  **SUM**(salary) **OVER()** AS s1,
  **SUM**(salary) **OVER**(ORDER BY salary) AS s2,
  **SUM**(salary) **OVER**(ORDER BY salary
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS s3,
  **SUM**(salary) **OVER**(ORDER BY salary
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS s4,
  **SUM**(salary) **OVER**(ORDER BY salary
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS s5,
  **SUM**(salary) **OVER**(ORDER BY salary
    ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s6,
  **SUM**(salary) **OVER**(ORDER BY salary
    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS s7,
  **SUM**(salary) **OVER**(ORDER BY salary RANGE 1 PRECEDING) AS s8
```
10.5. Именованные окна

Для того чтобы не писать каждый раз сложные выражения для задания окна, имя окна можно задать в предложении WINDOW. Имя окна может быть использовано в предложении OVER для ссылки на определение окна, кроме того оно может быть использовано в качестве базового окна для другого именованного или встроенного (в предложении OVER) окна. Окна с рамкой (с предложениями RANGE и ROWS) не могут быть использованы в качестве базового окна, но могут быть использованы в предложении OVER window_name. Окно, которое использует ссылку на базовое окно, не может иметь предложение PARTITION BY и не может переопределять сортировку с помощью предложения ORDER BY.

*Example 379. Использование именованных окон*

```sql
SELECT id, department, salary, 
count(*) OVER (w1), 
first_value(salary) OVER (w2), 
last_value(salary) OVER (w2), 
sum(salary) OVER (w2 ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s
FROM employee
WINDOW w1 AS (PARTITION BY department), 
w2 AS (w1 ORDER BY salary)
ORDER BY department, salary;
```
10.6. Ранжирующие функции

Ранжирующие функции вычисляют порядковый номер ранга внутри секции окна.

Эти функции могут применяться с использованием секционирования и сортировки и без них. Однако их использование без сортировки почти никогда не имеет смысла.

Функции ранжирования могут быть использованы для создания различных типов инкрементных счётчиков. Рассмотрим SUM(1) OVER (ORDER BY SALARY) в качестве примера того, что они могут делать, каждая из них различным образом. Ниже приведён пример запроса, который позволяет сравнить их поведение по сравнению с SUM.

```sql
SELECT id, salary, DENSE_RANK() OVER (ORDER BY salary), RANK() OVER (ORDER BY salary), PERCENT_RANK() OVER (ORDER BY salary), CUME_DIST() OVER (ORDER BY salary), NTILE(3) OVER (ORDER BY salary), ROW_NUMBER() OVER (ORDER BY salary), SUM(1) OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>dense_rank</th>
<th>rank</th>
<th>percent_rank</th>
<th>cume_dist</th>
<th>ntile</th>
<th>row_number</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>1</td>
<td>1</td>
<td>0.000000000000000</td>
<td>0.200000000000000</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>2</td>
<td>2</td>
<td>0.250000000000000</td>
<td>0.400000000000000</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>3</td>
<td>3</td>
<td>0.500000000000000</td>
<td>0.800000000000000</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>3</td>
<td>3</td>
<td>0.500000000000000</td>
<td>0.800000000000000</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>4</td>
<td>5</td>
<td>1.000000000000000</td>
<td>1.000000000000000</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

10.6.1. DENSE_RANK()

Доступно в DSQL

Синтаксис

```sql
DENSE_RANK() OVER {<windowSpecification> | window_name}
```

Тип возвращаемого результата

BIGINT
Возвращает ранг строк в секции результирующего набора без промежутков в ранжировании. Строки с одинаковыми значениями <order_exp> получают одинаковый ранг в пределах группы <partition_exp>, если она указана. Ранг строки равен количеству различных значений рангов в секции, предшествующих текущей строке, увеличенному на единицу.

**Example 380. Использование DENSE_RANK**

```sql
SELECT
    id,
    salary,
    DENSE_RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>dense_rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>4</td>
</tr>
</tbody>
</table>

См. также:

`SELECT, PARTITION BY, ORDER BY, RANK(), ROW_NUMBER().`

### 10.6.2. RANK()

**Доступно в**

**DSQL**

**Синтаксис**

```
RANK() OVER {<window_specification> | window_name}
```

**Тип возвращаемого результата**

`BIGINT`

Возвращает ранг каждой строки в секции результирующего набора. Строки с одинаковыми значениями <order_exp> получают одинаковый ранг в пределах группы <partition_exp>, если она указана. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки.
Example 381. Использование RANK

```sql
SELECT
    id,
    salary,
    RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>5</td>
</tr>
</tbody>
</table>

См. также:

```sql
SELECT, PARTITION BY, ORDER BY, DENSE_RANK(), ROW_NUMBER().
```

10.6.3. PERCENT_RANK()

Доступно в

DSQL

**Синтаксис**

```sql
PERCENT_RANK() OVER {<window_specification> | window_name}
```

**Тип возвращаемого результата**

DOUBLE PRECISION

Возвращает относительный ранг текущей строки в группе строк. Функция PERCENT_RANK используется для вычисления относительного положения значения в секции или результирующем наборе запроса. Диапазон значений, возвращаемый функцией PERCENT_RANK, больше 0 и меньше или равен 1. В первой строке любого набора PERCENT_RANK равна 0. Значения NULL по умолчанию включаются и рассматриваются как наименьшие возможные значения.

Функция PERCENT_RANK вычисляется как \((RANK - 1) / (total\_rows - 1)\), где `total\_rows` общее количество строк в секции.
Example 382. Использование PERCENT RANK

```
SELECT
    id,
    salary,
    PERCENT_RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>percent_rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>1.0</td>
</tr>
</tbody>
</table>

См. также:
SELECT, PARTITION BY, ORDER BY, RANK, CUME_DIST.

10.6.4. CUME_DIST()

Доступно в
DSQL

Синтаксис

```
CUME_DIST() OVER {<window_specification> | window_name}
```

Тип возвращаемого результата
DOUBLE PRECISION

Функция CUME_DIST рассчитывает кумулятивное распределение значения в наборе данных. Возвращаемое значение находится в диапазоне от 0 до 1. Функция CUME_DIST рассчитывается как (число строк, предшествующих или равных текущей) / (общее число строк). Для равных значений всегда вычисляется одно и то же значение накопительного распределения. Значения NULL по умолчанию включаются и рассматриваются как наименьшие возможные значения.

Example 383. Использование CUME_DIST

```
SELECT
    id,
    salary,
```
CUME_DIST() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>cume_dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>0.8</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>1.0</td>
</tr>
</tbody>
</table>

См. также:

SELECT, PARTITION BY, ORDER BY, RANK(), PERCENT_RANK().

10.6.5. NTILE()

Доступно в DSQL

Синтаксис

NTILE(<expr>) OVER {<window_specification> | window_name}

Table 235. Параметры функции NTILE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение целочисленного типа. Указывает количество групп, на которые необходимо разделить каждую секцию.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата

BIGINT

Функция NTILE распределяет строки упорядоченной секции в заданное количество групп так, чтобы размеры групп были максимально близки. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы, которой принадлежит строка.

Если количество строк в секции не делится на expr, то формируются группы двух размеров, отличающихся на единицу. Группы большего размера следуют перед группами меньшего размера в порядке, заданном в предложении OVER.

Example 384. Использование NTILE

SELECT
```sql
id, salary, NTILE(3) OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>ntile</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>3</td>
</tr>
</tbody>
</table>

См. также:

SELECT, PARTITION BY, ORDER BY.

### 10.6.6. ROW_NUMBER()

Доступно в

DSQL

**Синтаксис**

```sql
ROW_NUMBER() OVER {<window_specification> | window_name}
```

**Тип возвращаемого результата**

BIGINT

Возвращает последовательный номер строки в секции результирующего набора, где 1 соответствует первой строке в каждой из секций.

**Example 385. Использование ROW_NUMBER**

```sql
SELECT id, salary, ROW_NUMBER() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>row_number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**См. также:**

```sql
SELECT, PARTITION BY, ORDER BY, RANK(), DENSE_RANK().
```

### 10.7. Навигационные функции

Навигационные функции получают простые (не агрегированные) значения выражения из другой строки запроса в той же секции.

Функции FIRST_VALUE, LAST_VALUE и NTH_VALUE оперируют на рамке окна (кадрах окна). По умолчанию, если задано предложение ORDER BY, то рамка состоит из всех строк, от начала разбиения до текущей строки, плюс любые следующие строки, которые равны текущей строке в соответствии с предложением ORDER BY, т.е.

```sql
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Из-за этого результаты функций NTH_VALUE и в особенности LAST_VALUE могут показаться странными. Для устранения этого "недостатка" вы можете задать другую рамку окна, например:

```sql
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

**Example 386. Навигационные функции**

```sql
SELECT id, salary, 
FIRST_VALUE(salary) OVER (ORDER BY salary), 
LAST_VALUE(salary) OVER (ORDER BY salary), 
NTH_VALUE(salary, 2) OVER (ORDER BY salary), 
LAG(salary) OVER (ORDER BY salary), 
LEAD(salary) OVER (ORDER BY salary) 
FROM employee 
ORDER BY salary;
```

<table>
<thead>
<tr>
<th>Результат</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Вариант с изменённой рамкой окна для функций LAST_VALUE и NTH_VALUE

```sql
SELECT id, salary, 
    FIRST_VALUE(salary) OVER (ORDER BY salary),
    LAST_VALUE(salary) OVER w,
    NTH_VALUE(salary, 2) OVER w,
    LAG(salary) OVER (ORDER BY salary),
    LEAD(salary) OVER (ORDER BY salary)
FROM employee
WINDOW w AS (ORDER BY salary ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
ORDER BY salary;
```

Результат

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>first_value</th>
<th>last_value</th>
<th>nth_value</th>
<th>lag</th>
<th>lead</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>8.00</td>
<td>12.00</td>
<td>9.00</td>
<td>&lt;null&gt;</td>
<td>9.00</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>8.00</td>
<td>9.00</td>
<td>9.00</td>
<td>8.00</td>
<td>10.00</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>8.00</td>
<td>12.00</td>
<td>9.00</td>
<td>9.00</td>
<td>10.00</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>8.00</td>
<td>12.00</td>
<td>9.00</td>
<td>10.00</td>
<td>12.00</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>8.00</td>
<td>12.00</td>
<td>9.00</td>
<td>10.00</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

### 10.7.1. FIRST_VALUE()

**Доступно в**

DSQL

**Синтаксис**

```
FIRST_VALUE(<expr>) OVER {<window_specification> | window_name}
```

**Table 236. Параметры функции FIRST_VALUE**
Параметр | Описание
--- | ---
expr | Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

**Тип возвращаемого результата**
точнее, что и аргумент функции expr

Возвращает первое значение из упорядоченного набора значений рамки окна.

**Cм. также:**
SELECT, PARTITION BY, ORDER BY, LAST_VALUE(), NTH_VALUE().

### 10.7.2. LAG()

**Доступно в**
DSQL

**Синтаксис**

```
LAG(<expr> [ , <offset> [ , <default> ] ] )
OVER {<window_specification> | window_name}
```

**Таблица 237. Параметры функции LAG**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>offset</td>
<td>Количество строк до строки перед текущей строкой, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1. offset может быть столбцом, вложенным запросом или другим выражением, с помощью которого вычисляется целая положительная величина, или другим типом, который может быть неявно преобразован в B16INT. offset не может быть отрицательным значением или аналитической функцией.</td>
</tr>
<tr>
<td>default</td>
<td>Значение по умолчанию, которое возвращается, в случае если смещение (offset) указывает за пределы секции. По умолчанию равно NULL</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**
точнее, что и аргумент функции expr

Функция LAG обеспечивает доступ к строке с заданным физическим смещением (offset) перед началом текущей строки.
Если смещение (offset) указывает за пределы секции, то будет возвращено значение default, которое по умолчанию равно NULL.

Примеры:

Example 387. Использование функции LAG

Предположим у вас есть таблица rate, которая хранит курс валюты на каждый день. Необходимо проследить динамику изменения курса за последние пять дней.

```sql
SELECT
    bydate,
    cost,
    cost - LAG(cost) OVER(ORDER BY bydate) AS change,
    100 * (cost - LAG(cost) OVER(ORDER BY bydate)) / 
    LAG(cost) OVER(ORDER BY bydate) AS percent_change
FROM rate
WHERE bydate BETWEEN DATEADD(-4 DAY TO current_date) AND current_date
ORDER BY bydate
```

Результат

<table>
<thead>
<tr>
<th>bydate</th>
<th>cost</th>
<th>change</th>
<th>percent_change</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.10.2014</td>
<td>31.00</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>28.10.2014</td>
<td>31.53</td>
<td>0.53</td>
<td>1.7096</td>
</tr>
<tr>
<td>29.10.2014</td>
<td>31.40</td>
<td>-0.13</td>
<td>-0.4123</td>
</tr>
<tr>
<td>30.10.2014</td>
<td>31.67</td>
<td>0.27</td>
<td>0.8598</td>
</tr>
<tr>
<td>31.10.2014</td>
<td>32.00</td>
<td>0.33</td>
<td>1.0419</td>
</tr>
</tbody>
</table>

См. также: SELECT, PARTITION BY, ORDER BY, LEAD().

10.7.3. LAST_VALUE()

Доступно в DSQL

Синтаксис

```
LAST_VALUE(<expr>) OVER {<window_specification> | window_name}
```

Table 238. Параметры функции LAST_VALUE
### Параметр

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

Тот же что и аргумент функции `expr`

Возвращает последнее значение из упорядоченного набора значений рамки окна.

**Cм. также:**

`SELECT, PARTITION BY, ORDER BY, FIRST_VALUE(), NTH_VALUE()`.

#### 10.7.4. LEAD() (LEAD)

**Доступно в**

DSQL

**Синтаксис**

```
LEAD(<expr> [, <offset> [, <default>]])
OVER {<window_specification> | window_name}
```

**Table 239. Параметры функции LEAD**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>offset</td>
<td>Количество строк после текущей строки до строки, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1. <code>offset</code> может быть столбцом, вложенным запросом или другим выражением, с помощью которого вычисляется целая положительная величина, или другим типом, который может быть неявно преобразован в BIGINT. <code>offset</code> не может быть отрицательным значением или аналитической функцией.</td>
</tr>
<tr>
<td>default</td>
<td>Значение по умолчанию, которое возвращается, в случае если смещение (<code>offset</code>) указывает за пределы секции. По умолчанию равно NULL.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

Тот же что и аргумент функции `expr`

Функция LEAD обеспечивает доступ к строке на заданном физическом смещении (`offset`) после текущей строки.
Если смещение (offset) указывает за пределы секции, то будет возвращено значение default, которое по умолчанию равно NULL.

См. также:
SELECT, PARTITION BY, ORDER BY, LAG().

10.7.5. NTH_VALUE()

Доступно в
DSQL

Синтаксис

```
NTH_VALUE(<expr> [, <offset>]) [FROM FIRST | FROM LAST]
OVER {<window_specification> | window_name}
```

Table 240. Параметры функции NTH_VALUE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.</td>
</tr>
<tr>
<td>offset</td>
<td>Номер записи, начинная с первой (опция FROM FIRST) или последней (опция FROM LAST) записи.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
точке того как и аргумент функции expr

Функция NTH_VALUE возвращает N-ое значение, начиная с первой (опция FROM FIRST) или последней (опция FROM LAST) записи. По умолчанию используется опция FROM FIRST. Смещение 1 от первой записи будет эквивалентно функции FIRST_VALUE, смещение 1 от последней записи будет эквивалентно функции LAST_VALUE.

См. также:
SELECT, PARTITION BY, ORDER BY, FIRST_VALUE(), LAST_VALUE().

10.8. Агрегатные функции внутри оконных

В качестве аргументов оконных функций, а также в предложении OVER разрешено использование агрегатных функций (но не оконных). В этом случае сначала вычисляются агрегатные функции, а только затем на них накладываются окна оконных функций.

При использовании агрегатных функций в качестве аргументов оконных функций, все столбцы, не используемые в агрегатных функциях должны быть указаны в предложении GROUP BY.
Example 388. Использование агрегатной функции в качестве аргумента оконной

```sql
SELECT
    code_employee_group,
    AVG(salary) AS avg_salary,
    RANK() OVER(ORDER BY AVG(salary)) AS salary_rank
FROM employee
GROUP BY code_employee_group
```
Chapter 11. Системные пакеты

Системные пакеты предоставляют служебные хранимые процедуры и функции.

Список системных пакетов

Пакет RDB$TIME_ZONE_UTIL

Утилиты поддержки часовых поясов

11.1. Пакет RDB$TIME_ZONE_UTIL

Пакет RDB$TIME_ZONE_UTIL содержит процедуры и функции для работы с часовым поясом.

11.1.1. Функция RDB$TIME_ZONE_UTIL.DATABASE_VERSION()

Функция RDB$TIME_ZONE_UTIL.DATABASE_VERSION возвращает версию базы данных часовых поясов (из библиотеки icu).

Тип возвращаемого результата

VARCHAR(10) CHARACTER SET ASCII

Example 389. Использование функции RDB$TIME_ZONE_UTIL.DATABASE_VERSION

```
SELECT rdb$time_zone_util.database_version()
FROM rdb$database;
```

DATABASE_VERSION

================

2021a

11.1.2. Процедура RDB$TIME_ZONE_UTIL.TRANSITIONS()

Процедура RDB$TIME_ZONE_UTIL.TRANSITIONS возвращает набор правил для часового пояса между начальной и конечной временной меткой.

Table 241. Входные параметры процедуры RDB$TIME_ZONE_UTIL.TRANSITIONS

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Тип</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$TIME_ZONE_NAME</td>
<td>CHAR(63)</td>
<td>Наименование часового пояса</td>
</tr>
<tr>
<td>RDB$FROM_TIMESTAMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Начало интервала дат</td>
</tr>
<tr>
<td>RDB$TO_TIMESTAMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Окончание интервала дат</td>
</tr>
</tbody>
</table>

Table 242. Выходные параметры процедуры RDB$TIME_ZONE_UTIL.TRANSITIONS
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Тип</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$START_TIMESTAMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Дата начала действия правила</td>
</tr>
<tr>
<td>RDB$END_TIMESTAMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Дата окончания действия правила</td>
</tr>
<tr>
<td>RDB$ZONE_OFFSET</td>
<td>SMALLINT</td>
<td>Смещение времени в минутах для заданного часового пояса</td>
</tr>
<tr>
<td>RDB$DST_OFFSET</td>
<td>SMALLINT</td>
<td>Летнее смещение времени в минутах для заданного часового пояса</td>
</tr>
<tr>
<td>RDB$EFFECTIVE_OFFSET</td>
<td>SMALLINT</td>
<td>Эффективное смещение, вычисляется как RDB$ZONE_OFFSET + RDB$DST_OFFSET</td>
</tr>
</tbody>
</table>

Example 390. Использование процедуры RDB$TIME_ZONE_UTIL.TRANSITIONS

```
SELECT RDB$START_TIMESTAMP, RDB$END_TIMESTAMP, RDB$ZONE_OFFSET AS ZONE_OFF, RDB$DST_OFFSET AS DST_OFF, RDB$EFFECTIVE_OFFSET AS OFF
FROM rdb$time_zone_util.transitions('America/Sao.Paulo', timestamp '2017-01-01', timestamp '2019-01-01');
```
Chapter 12. Контекстные переменные

12.1. CURRENT_CONNECTION

Доступно в
DSQL, PSQL

Синтаксис

CURRENT_CONNECTION

Тип возвращаемого результата
BIGINT

Переменная CURRENT_CONNECTION хранит уникальный идентификатор текущего соединения. Значение переменной хранится в странице заголовка базы и сбрасывается после restore. Переменная увеличивается на единицу при каждом последующем соединении с базой данных (соединения также могут быть внутренними вызванными самим ядром). Следовательно, переменная показывает количество подключений произошедших к базе после её восстановления (или после её создания).

Example 391. Использование переменной CURRENT_CONNECTION

SELECT CURRENT_CONNECTION FROM RDB$DATABASE

См. также:
CURRENT_TRANSACTION.

12.2. CURRENT_DATE

Доступно в
DSQL, PSQL, ESQL

Синтаксис

CURRENT_DATE

Тип возвращаемого результата
DATE

Переменная CURRENT_DATE возвращает текущую дату сервера.
**Example 392. Использование переменной CURRENT_DATE**

```sql
CREATE DOMAIN DDATE_DNN AS DATE DEFAULT CURRENT_DATE NOT NULL
```

См. также:

`CURRENT_TIMESTAMP, CURRENT_TIME`.

### 12.3. CURRENT_ROLE

**Доступно в**

DSQL, PSQL

**Синтаксис**

```
CURRENT_ROLE
```

**Тип возвращаемого результата**

VARCHAR(63)

Контекстная переменная CURRENT_ROLE служит для определения роли, с которой произошло подключение к базе данных, или которая была установлена позже с помощью оператора SET ROLE. В случае если произошло подключение без указания роли и роль не была указана позже с помощью оператора SET ROLE, переменная принимает значение NONE.

Контекстная переменная CURRENT_ROLE содержит только имя явно указанной роли (при входе в систему или с помощью SET ROLE), неявно определяемые роли (выданные оператором GRANT с использованием ключевого слова DEFAULT) не будут попадать в неё. Для того чтобы узнать, используется ли текущим пользователем неявно указанная роль, используйте системную функцию `RDB$ROLE_IN_USE()`.

**Example 393. Использование переменной CURRENT_ROLE**

```sql
SELECT CURRENT_ROLE FROM RDB$DATABASE
```

Такое же значение можно будет получить и в результате выполнения запроса:

```sql
SELECT RDB$GET_CONTEXT ('SYSTEM', 'CURRENT_ROLE')
FROM RDB$DATABASE;
```
См. также:

```
SET ROLE, RDB$GET_CONTEXT(), RDB$ROLE_IN_USE().
```

### 12.4. CURRENT_TIME

Доступно в

DSQL, PSQL, ESQI

**Синтаксис**

```
CURRENT_TIME ([<precision>])
```

<precision> ::= 0 | 1 | 2 | 3

**Table 243. Параметры контекстной переменной CURRENT_TIME**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Точность. Значение по умолчанию 0. Не поддерживается в ESQI.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

TIME WITH TIME ZONE

Переменная CURRENT_TIME возвращает текущее время в часовом поясе сессии, включая информацию о часовом поясе. Точность определяет, сколько учитывать знаков после запятой в долях секунды. По умолчанию точность равна 0.

В блоке кода PSQL (процедура, триггер, исполняемый блок) значение CURRENT_TIME не меняется по мере выполнения. При вызове вложенного кода, значение также не изменится и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте `CAST('NOW' AS TIME)`.

**Example 394. Использование переменной CURRENT_TIME**

```
SELECT CURRENT_TIME(2) FROM RDB$DATABASE;
-- результат будет (например) 23:35:33.1200 Europe/Moscow
```

См. также:

```
CURRENT_TIMESTAMP, CURRENT_DATE.
```

### 12.5. CURRENT_TIMESTAMP

Доступно в

DSQL, PSQL, ESQI
**Синтаксис**

```
CURRENT_TIMESTAMP [(<precision>)]
<precision> ::= 0 | 1 | 2 | 3
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Точность. Значение по умолчанию 3. Не поддерживается в ESQL.</td>
</tr>
</tbody>
</table>

**Table 244. Параметры контекстной переменной CURRENT_TIMESTAMP**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Точность. Значение по умолчанию 3. Не поддерживается в ESQL.</td>
</tr>
</tbody>
</table>

**Тип возвращаемого результата**

**TIMESTAMP WITH TIME ZONE**

Переменная CURRENT_TIMESTAMP возвращает текущую дату и время в часовом поясе сессии, включая информацию о часовом поясе. Точность определяет, сколько учитывать знаков после запятой в долях секунды. Точность по умолчанию равна 3.

В блоке кода PSQL (процедура, триггер, исполняемый блок) значение CURRENT_TIMESTAMP не меняется по мере выполнения. При вызове вложенного кода, значение также не изменится и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте `CAST('NOW' AS TIMESTAMP)`.

**Example 395. Использование переменной CURRENT_TIMESTAMP**

```
SELECT CURRENT_TIMESTAMP(2) FROM RDB$DATABASE;
-- результат будет (например) 02.03.2014 23:35:33.1200 Europe/Moscow
```

**См. также:**

CURRENT_TIME, CURRENT_DATE.

### 12.6. CURRENT_TRANSACTION

**Доступно в**

DSQL, PSQL

**Синтаксис**

```
CURRENT_TRANSACTION
```

**Тип возвращаемого результата**

**BIGINT**

Переменная CURRENT_TRANSACTION содержит уникальный номер текущей транзакции.
Значение `CURRENT_TRANSACTION` хранится в странице заголовка базы данных и сбрасывается в 0 после восстановления (или создания базы). Оно увеличивается при старте новой транзакции.

Example 396. Использование переменной CURRENT_TRANSACTION

```sql
SELECT CURRENT_TRANSACTION FROM RDB$DATABASE;
NEW.TRANS_ID = CURRENT_TRANSACTION;
```

См. также:
`CURRENT_CONNECTION`, `RDB$GET_CONTEXT()`.

12.7. CURRENT_USER

Доступно в
DSQL, PSQL

Синтаксис

```sql
CURRENT_USER
```

Тип возвращаемого результата
VARCHAR(63)

Переменная CURRENT_USER содержит имя текущего подключенного пользователя базы данных.

Example 397. Использование переменной CURRENT_USER

```sql
NEW.ADDED_BY = CURRENT_USER;
```

См. также:
`USER`, `CURRENT_ROLE`.

12.8. DELETING

Доступно в
PSQL

Синтаксис

```sql
DELETING
```
Тип возвращаемого результата
BOOLEAN

Контекстная переменная DELETING доступна только в коде табличных триггеров. Используется в триггерах на несколько типов событий и показывает, что триггер сработал при выполнении операции DELETE.

**Example 398. Использование переменной DELETING**

```sql
... IF (DELETING) THEN BEGIN
    INSERT INTO REMOVED_CARS (ID, MAKE, MODEL, REMOVED) VALUES (OLD.ID, OLD.MAKE, OLD.MODEL, CURRENT_TIMESTAMP);
END ...
```

**См. также:**
**INSERTING, UPDATING.**

### 12.9. GDSCODE

Доступно в PSQL

**Синтаксис**

```
GDSCODE
```

Тип возвращаемого результата
INTEGER

В блоке обработки ошибок WHEN ... DO контекстная переменная GDSCODE содержит числовое представление текущего кода ошибки Firebird. До версии Firebird 2.0 GDSCODE можно было получить только с использованием конструкции WHEN GDSCODE. Теперь эту контекстную переменную можно также использовать в блоках WHEN ANY, WHEN SQLCODE и WHEN EXCEPTION при условии, что код ошибки соответствует коду ошибки Firebird. Вне обработчика ошибок GDSCODE всегда равен 0. Вне PSQL GDSCODE не существует вообще.

**Example 399. Использование переменной GDSCODE**

```sql
... WHEN GDSCODE GRANT_OBJ_NOTFOUND,
     GDSCODE GRANT_FLD_NOTFOUND,
...```
Обратите внимание, пожалуйста: после WHEN GDSCODE вы должны использовать символьные имена — такие, как `grant_obj_notfound` и т.д. Но контекстная переменная GDSCODE — целое число. Для сравнения его с определённой ошибкой вы должны использовать числовое значение, например, 335544551 для `grant_obj_notfound`.

См. также: SQLCODE, SQLSTATE.

12.10. INSERTING

Доступно в PSQL

Синтаксис

```
INSERTING
```

Тип возвращаемого результата

BOOLEAN

Контекстная переменная INSERTING доступна только коде табличных триггеров. Используется в триггерах на несколько типов событий и показывает, что триггер сработал при выполнении операции INSERT.

Example 400. Использование переменной INSERTING

```
... 
IF (INSERTING OR UPDATING) THEN 
BEGIN 
  IF (NEW.SERIAL_NUM IS NULL) THEN 
    NEW.SERIAL_NUM = GEN_ID (GEN_SERIALS, 1); 
  END 
... 
```

См. также:
12.11. LOCALTIME

Доступно в
DSQL, PSQL, ESQL

Синтаксис

LOCALTIME [(<precision>)]

<precision> ::= 0 | 1 | 2 | 3

Table 245. Параметры контекстной переменной LOCALTIME

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Точность. Значение по умолчанию 0. Не поддерживается в ESQL.</td>
</tr>
</tbody>
</table>

Тип возвращаемого результата
TIME WITHOUT TIME ZONE

Переменная LOCALTIME возвращает текущее время в часовом поясе сессии, без информации о часовом поясе. Точность определяет, сколько учитывать знаков после запятой в долях секунды. Точность по умолчанию равна 0.

В блоке кода PSQL (процедура, триггер, исполняемый блок) значение LOCALTIME не меняется по мере выполнения. При вызове вложенного кода, значение также не изменится и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте CAST('NOW' AS TIME WITHOUT TIME ZONE).

Example 401. Использование переменной LOCALTIME

```sql
SELECT LOCALTIME(2) FROM RDB$DATABASE;
-- результат будет (например) 23:35:33.1200
```

См. также:
CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_DATE.

12.12. LOCALTIMESTAMP

Доступно в
DSQL, PSQL, ESQL
В блоке кода PSQL (процедура, триггер, исполняемый блок) значение LOCALTIMESTAMP не меняется по мере выполнения. При вызове вложенного кода, значение также не изменится и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте CAST('NOW' AS TIMESTAMP WITHOUT TIME ZONE).

Example 402. Использование переменной LOCALTIMESTAMP

```sql
SELECT LOCALTIMESTAMP(2) FROM RDB$DATABASE;
```

-- результат будет (например) 02.03.2014 23:35:33.1200

См. также:
CURRENT_TIMESTAMP, CURRENT_TIME, CURRENT_DATE.

### 12.13. NEW

Доступно в
PSQL

**Синтаксис**

`NEW`

Контекстная переменная `NEW` доступна только в коде табличных триггеров. Значение `NEW` содержит новые значения полей данных, которое возникли в базе во время операции обновления или вставки.

В `AFTER` триггерах переменная доступна только для чтения.
Для табличных триггеров, срабатывающих на несколько типов событий, переменная NEW доступна всегда. Однако в случае если триггер сработал на операцию удаления, то для него новая версия данных не имеет смысла. В этой ситуации чтение переменной NEW всегда вернёт NULL.

Попытка записи в переменную NEW в AFTER триггере вызовет исключение в коде.

**Example 403. Использование переменной NEW**

```sql
... IF (NEW.SERIAL_NUM IS NULL) THEN
    NEW.SERIAL_NUM = GEN_ID (GEN_SERIALS, 1);
...```

**См. также:**

OLD.

### 12.14. OLD

**Доступно в**

PSQL

**Синтаксис**

OLD

Контекстная переменная OLD доступна только коде триггеров. Значения, содержащееся в OLD, хранит прошлые значения полей, которые были в базе до операции изменения или удаления.

Переменная OLD доступна только для чтения.

Для табличных триггеров, срабатывающих на несколько типов событий, значения для переменной OLD всегда возможны. Однако для триггеров, сработавших на вставку записи, значение данной переменной не имеет смысла, поэтому в этой ситуации чтение OLD возвратит NULL, а попытка записи в неё вызовет исключение в коде.

**Example 404. Использование переменной OLD**

```sql
... IF (NEW.QUANTITY IS DISTINCT FROM OLD.QUANTITY) THEN
    DELTA = NEW.QUANTITY - OLD.QUANTITY;
...```
См. также:
NEW.

12.15. RESETTING

Доступно в PSQL

Синтаксис

RESETTING

Тип возвращаемого результата
BOOLEAN

Контекстная переменная RESETTING доступна только коде триггеров на события ON CONNECT и ON DISCONNECT, и может использоваться в любом месте, где можно использовать логический предикат. Системная переменная RESETTING, позволяет обнаружить случай, когда триггер базы данных срабатывает из-за сброса сеанса, например с помощью оператора ALTER SESSION RESET. Её значение TRUE, если выполняется сброс сеанса, и FALSE в противном случае.

Example 405. Использование переменной RESETTING

```sql
... IF (RESETTING) THEN BEGIN -- выполняется сброс сеанса END ... ...
```

См. также:
ALTER SESSION RESET.

12.16. ROW_COUNT

Доступно в PSQL

Синтаксис

ROW_COUNT
Тип возвращаемого результата
BIGINT

Контекстная переменная ROW_COUNT содержит число строк, затронутых последним оператором DML (INSERT, UPDATE, DELETE, SELECT или FETCH) в текущем триггере, хранимой процедуре или исполняемом блоке.

Поведение с SELECT и FETCH:

- После выполнения singleton SELECT запроса (запроса, который может вернуть не более одной строки данных), ROW_COUNT равна 1, если была получена строка данных и 0 в противном случае;
- В цикле FOR SELECT переменная ROW_COUNT увеличивается на каждой итерации (начиная с 0 в качестве первого значения);
- После выборки (FETCH) из курсора, ROW_COUNT равна 1, если была получена строка данных и 0 в противном случае. Выборка нескольких записей из одного курсора не увеличивает ROW_COUNT после 1.

Переменная ROW_COUNT не может быть использована для определения количества строк, затронутых при выполнении операторов EXECUTE STATEMENT или EXECUTE PROCEDURE. Для оператора MERGE переменная ROW_COUNT будет содержать 0 или 1, даже если было затронуто более записей.

Не используйте переменную ROW_COUNT внутри DML операторов. Дело в том, что эта переменная сбрасывает своё значение в 0 перед началом выполнения любого DML оператора, а потому вы можете получить не то что ожидаете.

В вышеприведённом примере в столбец affected_rows будут записаны нулевые значения, даже если оператором UPDATE были затронуты строки. Для того чтобы исправить эту ошибку, необходимо сохранить значение контекстной переменной ROW_COUNT в локальную переменную PSQL модуля и использовать эту локальную переменную в DML операторе.
update t2 set
  evt='upd',
  old_id = old.id, old_x = old.x,
  new_id = new.id, new_x = new.x
where new_id = old.id;

rc = row_count;
insert into t2log(evt, affected_rows) values('upd', rc);
...

Example 406. Использование переменной row_count

... update figures set number = 0 where id = :id;
if (row_count = 0) then
  insert into figures (id, number)
  values (:id, 0);
...

12.17. SQLCODE

Доступно в
PSQL

Синтаксис

SQLCODE

type возвращаемого результата

integer

В блоках обработки ошибок WHEN ... DO контекстная переменная SQLCODE содержит текущий код ошибки SQL. До Firebird 2.0 значение SQLCODE можно было получить только в блоках обработки ошибок WHEN SQLCODE и WHEN ANY. Теперь она может быть отлична от нуля в блоках WHEN GDSCODE и WHEN EXCEPTION при условии, что ошибка, вызвавшее срабатывание блока, соответствует коду ошибки SQL. Вне обработчиков ошибок SQLCODE всегда равен 0, а вне PSQL не существует вообще.

Example 407. Использование переменной SQLCODE

... when any do
begin
  if (sqlcode <> 0) then
    msg = 'обнаружена ошибка SQL!';

**12.18. SQLSTATE**

**Доступно в**
PSQL

**Синтаксис**

```sql
SQLSTATE
```

**Тип возвращаемого результата**

CHAR(5)

В блоках обработки ошибок `WHEN … DO` контекстная переменная `SQLSTATE` переменная содержит 5 символов SQL-2003 — совместимого кода состояния, переданного оператором, вызвавшим ошибку. Вне обработчиков ошибок `SQLSTATE` всегда равен '00000', а вне PSQL не существует вообще.

- `SQLSTATE` предназначен для замены `SQLCODE`. Последняя, в настоящее время устарела и будет удалена будущих версиях Firebird;
- Любой код `SQLSTATE` состоит из двух символов класса и трёх символов подкласса. Класс 00 (успешное выполнение), 01 (предупреждение) и 02 (нет данных) представляют собой условия завершения. Каждый код статуса вне этих классов является исключением. Поскольку классы 00, 01 и 02 не вызывают ошибку, они никогда не будут обнаруживаться в переменной `SQLSTATE`.

**Example 408. Использование переменной SQLSTATE**

```sql
WHEN ANY DO
BEGIN
    MSG = CASE SQLSTATE
        WHEN '22003' THEN
            'Число вышло за пределы диапазона!'
        WHEN '22012' THEN
            'Деление на ноль!'
        WHEN '23000' THEN
            'Нарушение ограничения целостности!'
    END

    ... 
END
```

**См. также:**

`GDSCODE, SQLSTATE`. 

---

**Chapter 12. Контекстные переменные**
ELSE 'Ошибок нет! SQLSTATE = ' || SQLSTATE;
END;
EXCEPTION EX_CUSTOM MSG;
END

См. также:
GDS_CODE, SQLCODE, Коды ошибок SQLSTATE.

12.19. UPDATING

Доступно в
PSQL

Синтаксис

UPDATING

Тип возвращаемого результата
BOOLEAN

Контекстная переменная UPDATING доступна только коде табличных триггеров. Используется в триггерах на несколько типов событий и показывает, что триггер сработал при выполнении операции UPDATE.

Example 409. Использование переменной UPDATING

... IF (INSERTING OR UPDATING) THEN BEGIN IF (NEW.SERIAL_NUM IS NULL) THEN NEW.SERIAL_NUM = GEN_ID (GEN_SERIALS, 1); END ...

См. также:
INSERTING, DELETING.

12.20. USER

Доступно в
DSQL, PSQL
Синтаксис

USER

Тип возвращаемого результата
VARCHAR(63)

Переменная USER содержит имя текущего подключенного пользователя базы данных.

Example 410. Использование переменной USER

```
NEW.ADDED_BY = USER;
```

См. также:
CURRENT_USER, CURRENT_ROLE.
Chapter 13. Управление транзакциями

Всё в Firebird выполняется в рамках транзакций. Транзакция — логическая единица изолированной работы группы последовательных операций над базой данных. Изменения над данными остаются обратимыми до тех пор, пока клиентское приложение не выдаст серверу инструкцию COMMIT.

Firebird имеет небольшое количество SQL операторов, которые могут использоваться клиентскими приложениями для старта, управления, подтверждения или отмены транзакций, но достаточное для всех задач над базой данных:

**SET TRANSACTION**
- задание параметров транзакции и её старт;

**COMMIT**
- завершение транзакции и сохранение изменений;

**ROLLBACK**
- отмена изменений произошедший в рамках транзакции;

**SAVEPOINT**
- установка точки сохранения для частичного отката изменений, если это необходимо;

**RELEASE SAVEPOINT**
- удаление точки сохранения.

### 13.1. SET TRANSACTION

**Назначение**
Задаёт параметры транзакции и стартует её.

**Доступно в**
DSQL, ESQL

**Синтаксис**

```
SET TRANSACTION
   [NAME tr_name]
   [<tr_option> ...]
```

<tr_option> ::= 
- READ {ONLY | WRITE} 
- [NO] WAIT 
- [ISOLATION LEVEL] <isolation level> 
- NO AUTO UNDO 
- RESTART REQUESTS 
- IGNORE LIMBO 
- LOCK TIMEOUT seconds
```
Table 247. Параметры оператора SET TRANSACTION

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>tr_name</td>
<td>Имя транзакции. Доступно только в ESQL.</td>
</tr>
<tr>
<td>seconds</td>
<td>Время ожидания оператора (statement) в секундах при возникновении конфликта.</td>
</tr>
<tr>
<td>tables</td>
<td>Список таблиц для резервирования.</td>
</tr>
<tr>
<td>dbhandles</td>
<td>Список баз данных, к которым база данных может получить доступ. Доступно только в ESQL.</td>
</tr>
<tr>
<td>table_spec</td>
<td>Спецификация резервирования таблицы.</td>
</tr>
<tr>
<td>tablename</td>
<td>Имя таблицы для резервирования.</td>
</tr>
<tr>
<td>dbhandle</td>
<td>Хендл базы данных, к которой транзакция может получить доступ. Доступно только в ESQL.</td>
</tr>
<tr>
<td>snapshot number</td>
<td>Номер снимка другой транзакции, данные снимка базы данных которой должны быть общими с новой транзакцией.</td>
</tr>
</tbody>
</table>

Оператор SET TRANSACTION задаёт параметры транзакции и стартует её. Старт транзакции осуществляется только клиентскими приложениями, но не сервером (за исключением автономных транзакций и некоторых фоновых системных потоков/процессов, например, таких как sweep).

Каждое клиентское приложение может запускать произвольное количество одновременно выполняющихся транзакций. Фактически есть ограничение на общее количество выполняемых транзакций во всех клиентских приложениях, работающих с одной конкретной базой данных с момента последнего восстановления базы данных с резервной копии или с момента первоначального создания базы данных. Это количество равняется числу $2^{48} - 1$ то есть $~ 2,8 \times 10^{14}$ транзакций. В API и MON$ таблицах номер транзакции представляет собой 64 битное число.
Все предложения в операторе SET TRANSACTION являются необязательными. Если в операторе запуска транзакции на выполнение не задано никакого предложения, то предполагается старт транзакции со значениями всех характеристик по умолчанию (режим доступа, режим разрешения блокировок и уровень изолированности).

По умолчанию транзакция стартует со следующими характеристиками.

```sql
SET TRANSACTION
READ WRITE
WAIT ISOLATION LEVEL SNAPSHOT;
```

При старте со стороны клиента любой транзакции (заданной явно или по умолчанию) сервер передаёт клиенту дескриптор транзакции (целое число). На стороне сервера транзакциям последовательно присваиваются номера. Этот номер средствами SQL можно получить, используя контекстную переменную CURRENT_TRANSACTION.

### 13.1.1. Параметры транзакции

Основными характеристиками транзакции являются:

- режим доступа к данным (READ WRITE, READ ONLY);
- режим разрешения блокировок (WAIT, NO WAIT) с возможным дополнительным уточнением LOCK TIMEOUT;
- уровень изоляции (READ COMMITTED, SNAPSHOT, SNAPSHOT TABLE STABILITY);
- средства резервирования или освобождения таблиц (предложение RESERVING).

#### Имя транзакции

Необязательное предложение NAME задаёт имя транзакции. Предложение NAME доступно только в Embedded SQL. Если предложение NAME не указано, то оператор SET TRANSACTION применяется к транзакции по умолчанию. За счёт именованных транзакций позволяет одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноимённая переменная базового языка. В DSQSL, это ограничение предотвращает динамическую спецификацию имен транзакций.

#### Режим доступа

Для транзакций существует два режима доступа к данным базы данных: READ WRITE и READ ONLY.

- При режиме доступа READ WRITE операции в контексте данной транзакции могут быть как операциями чтения, так и операциями изменения данных. Это режим по умолчанию.
- В режиме READ ONLY в контексте данной транзакции могут выполняться только операции выборки данных SELECT. Любая попытка изменения данных в контексте такой транзакции приведёт к исключением базы данных. Однако это не относиться к глобальным временным таблицам (GTT), которые разрешено модифицировать в READ.
В Firebird API для режимов доступа предусмотрены следующие константы: isc_tpb_write соответствует режиму READ WRITE, isc_tpb_read — READ ONLY.

Режим разрешения блокировок

При работе с одной и той же базой данных нескольких клиентских приложений могут возникать блокировки. Блокировки могут возникать, когда одна транзакция неподтверждённые изменения в строку таблицы или удаляет строку, а другая транзакция пытается изменять или удалять эту же строку. Такие блокировки называются конфликтом обновления.

Блокировки также могут возникнуть и в других ситуациях при использовании некоторых уровней изоляции транзакций.

Существуют два режима разрешения блокировок: WAIT и NO WAIT.

Режим WAIT

В режиме WAIT (режим по умолчанию) при появлении конфликта с параллельными транзакциями, выполняющими конкурирующие обновления данных в той же базе данных, такая транзакция будет ожидать завершения конкурирующей транзакции путём её подтверждения (COMMIT) или отката (ROLLBACK). Иными словами, клиентское приложение будет переводено в режим ожидания до момента разрешения конфликта.

Если для режима WAIT задать предложение LOCK TIMEOUT, то ожидание будет продолжаться только указанное в этом предложении количество секунд. По истечении этого срока будет выдано сообщение об ошибке: “Lock time-out on wait transaction” (Истечение времени ожидания блокировки для транзакции WAIT).

Этот режим даёт несколько отличные формы поведения в зависимости от уровня изоляции транзакций.

В Firebird API режиму WAIT соответствует константа isc_tpb_wait.

Режим NO WAIT

Если установлен режим разрешения блокировок NO WAIT, то при появлении конфликта блокировки данная транзакция немедленно вызовет исключение базы данных.

В Firebird API режиму NO WAIT соответствует константа isc_tpb_nowait.

LOCK TIMEOUT это отдельная опция транзакции, но может использоваться только для транзакций WAIT. Указание LOCK TIMEOUT с транзацией NO WAIT вызовет ошибку “invalid parameter in transaction parameter block -Option isc_tpb_lock_timeout is not valid if isc_tpb_nowait was used previously in TPB”.

ISOLATION LEVEL

Уровень изолированности транзакций — значение, определяющее уровень, при котором в
транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Изменения, внесённые некоторым оператором, будут видны всем последующим операторам, запущенным в рамках этой же транзакции, независимо от её уровня изолированности. Изменения произведённые в рамках другой транзакции остаются невидимыми для текущей транзакции до тех пор, пока они не подтверждены. Уровень изолированности, а иногда, другие атрибуты, определяет, как транзакции будут взаимодействовать с другой транзакцией, которая хочет подтвердить изменения.

Необходимое предложение ISOLATION LEVEL задаёт уровень изолированности запускаемой транзакции. Это самая важная характеристика транзакции, которая определяет её поведение по отношению к другим одновременно выполняющимся транзакциям.

Существует три уровня изолированности транзакции:

- **SNAPSHOT**
- **SNAPSHOT TABLE STABILITY**
- **READ COMMITTED** с уточнениями (NO RECORD_VERSION или RECORD_VERSION или READ CONSISTENCY)

**Уровень изолированности SNAPSHOT**

Уровень изолированности SNAPSHOT (уровень изолированности по умолчанию) означает, что этой транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции. Любые подтверждённые изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции в процессе ее активности без её перезапуска. Чтобы увидеть эти изменения, нужно завершить транзакцию (подтвердить её или выполнить полный откат, но не откат на точку сохранения) и запустить транзакцию заново.

Изменения, вносимые автономными транзакциями, также не будут видны в контексте той ("внешней") транзакции, которая запустила эти автономные транзакции, если она работает в режиме SNAPSHOT.

В Firebird API режиму изолированности SNAPSHOT соответствует константа isc_tpb_concurrency.

**Предложение AT NUMBER**

Транзакцию с уровнем изолированности SNAPSHOT можно запустить на основе другой транзакции, если известен номер её снимка. В этом случае эта новая транзакция может видеть те же самые данные, что и транзакция на основе которой она запущена.

Эта функциональность позволяет создать параллельные процессы (в разных подключениях), считающие согласованные данные из базы данных. Например, процесс резервного копирования может создавать несколько потоков, параллельно считывающих данные из базы данных. Или веб-служба работать с распределёнными вспомогательными службами, выполняя некоторую обработку.

Это достигается созданием транзакции с использованием синтаксиса
SET TRANSACTION SNAPSHOT AT NUMBER snapshot_number

или через API с использованием константы isc_tpb_at_snapshot_number.

Значение snapshot_number из первой транзакции можно получить используя следующий запрос

RDB$GET_CONTEXT('SYSTEM', 'SNAPSHOT_NUMBER')

или через API информации о транзакции с константой fb_info_tra_snapshot_number.

Обратите внимание, snapshot_number должен быть номером снимка активной транзакции.

Уровень изолированности **SNAPSHOT TABLE STABILITY**

Уровень изолированности транзакции SNAPSHOT TABLE STABILITY позволяет, как и в случае SNAPSHOT, также видеть только те изменения, фиксация которых произошла не позднее момента старта этой транзакции. При этом после старта такой транзакции в других клиентских транзакциях невозможно выполнение изменений ни в каких таблицах этой базы данных, уже каким-либо образом измененных первой транзакцией. Все такие попытки в параллельных транзакциях приведут к исключениям базы данных. Просматривать любые данные другие транзакции могут совершенно свободно.

При помощи предложения резервирования RESERVING можно разрешить другим транзакциям изменять данные в некоторых таблицах.

Если на момент старта клиентом транзакции с уровнем изолированности SNAPSHOT TABLE STABILITY какая-нибудь другая транзакция выполнила неподтверждённое изменение данных любой таблицы базы данных, то запуск транзакции с таким уровнем изолированности приведёт к ошибке базы данных.

В Firebird API режиму изолированности SNAPSHOT TABLE STABILITY соответствует константа isc_tpb_consistency.

Уровень изолированности **READ COMMITTED**

Уровень изолированности READ COMMITTED позволяет в транзакции без её перезапуска видеть все подтверждённые изменения данных базы данных, выполненные в других параллельных транзакциях. Неподтверждённые изменения не видны в транзакциях этого уровня изолированности.

Для получения обновлённого списка строк интересующей таблицы необходимо лишь повторное выполнение оператора SELECT в рамках активной транзакции READ COMMITTED без её перезапуска.

В Firebird API режиму изолированности READ COMMITTED соответствует константа isc_tpb_read_committed.
Для этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов: RECORD_VERSION и NO RECORD_VERSION. Как видно из их имени они являются взаимоисключающими.

• NO RECORD_VERSION является в некотором роде механизмом двухфазной блокировки. В этом случае транзакция не может прочитать любую запись, которая была изменена параллельной активной (неподтвержденной) транзакцией.

  ◦ Если указана стратегия разрешения блокировок NO WAIT, то будет немедленно выдано соответствующее исключение.

  ◦ Если указана стратегия разрешения блокировок WAIT, то это приведёт к ожиданию завершения или отката конкурирующей транзакции. Если конкурирующая транзакция откатьивается, или, если она завершается и её идентификатор старее (меньше), чем идентификатор текущей транзакции, то изменения в текущей транзакции допускаются. Если конкурирующая транзакция завершается и её идентификатор новее (больше), чем идентификатор текущей транзакции, то будет выдана ошибка конфликта блокировок.

• При задании RECORD_VERSION транзакция всегда читает последнюю подтверждённую версию записей таблиц, независимо от того, существуют ли изменённые и ещё не подтверждённые версии этих записей. В этом случае режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции при её старте.

В Firebird API для способа разрешения конфликтов NO RECORD_VERSION соответствует константа isc_tpb_no_rec_version, a RECORD_VERSION — isc_tpb_rec_version.

Начиная с Firebird 4.0 эти опции являются устаревшими. По умолчанию они игнорируются и запускается транзакция READ COMMITTED READ CONSISTENCY. Это можно изменить установив параметр ReadConsistency (см. firebird.conf) в 0. В этом случае опции не игнорируются и работают точно так же как в предыдущих версиях. В будущих версиях этот параметр в firebird.conf может быть удалён.

**READ CONSISTENCY**

Если указана эта опция, то транзакция с режимом изолированности READ COMMITED делает стабильный снимок базы данных на время выполнения оператора. Каждый новый оператор верхнего уровня создает собственный моментальный снимок базы данных, чтобы видеть последние подтверждённые данные. Вложенные операторы (тригеры, вложенные хранимые процедуры и функции, динамические операторы и т. д.) используют тот же самый моментальный снимок базы данных, созданный оператором верхнего уровня. Таким образом обеспечивается согласованное чтение на момент начала выполнения оператора верхнего уровня. В Firebird 4.0 этот режим используется по умолчанию для транзакций с режимом изолированности READ COMMITED.

В Firebird API для стабильного снимка на уровне SQL оператора READ CONSISTENCY соответствует константа isc_tpb_read_consistency.
Обработка конфликта обновлений

Когда оператор выполняется в транзакции с режимом изолированности READ COMMITTED READ CONSISTENCY вид базы данных неизменен (подобно транзакции SNAPSHOT). Поэтому бесполезно ждать фиксации параллельной транзакции в надежде перечитать новую версию зафиксированной записи. При чтении поведение похоже на транзакцию READ COMMITTED RECORD_VERSION — оператор не ждёт завершения активной транзакции и обходит цепочку бекверсий, в которой ищет версию записи видимую для текущего моментального снимка.

Для режима изолированности READ COMMITTED READ CONSISTENCY обработка конфликтов обновлений Firebird значительно изменяется. При обнаружении конфликта обновления выполняется следующее:

a. режим изолированности транзакции временно переключается в режим READ COMMITTED NO RECORD VERSION;

b. Firebird устанавливает блокировку записи на конфликтную запись;

c. Firebird продолжает оценивать оставшиеся записи для удаления/обновления в курсоре, а также продолжает ставить на них блокировки;

d. когда больше нет записей для извлечения, запускается механизм для отмены всех выполненных действий, выполненных оператором верхнего уровня, и сохраняются все установленные блокировки для каждой обновлённой/удалённой/заблокированной записи, все вставленные записи удаляются;

e. затем Firebird восстанавливает режим изолированности транзакции как READ COMMITTED READ CONSISTENCY, создает новый снимок уровня оператора и перезапускает выполнение оператора верхнего уровня.

Такой алгоритм позволяет гарантировать, что после перезапуска уже обновленные записи останутся заблокированными, они будут видны новому снимку и могут быть обновлены снова без дальнейших конфликтов. Кроме того, из-за режима согласованности чтения набор измененных записей остается согласованным.

Замечания

• Приведенный выше алгоритм перезапуска применяется к операторам UPDATE, DELETE, SELECT WITH LOCK и MERGE, с предложением RETURNING и без него, выполняемым непосредственно из пользовательского приложения или в составе некоторого объекта PSQL (хранимая процедура, функция, триггер, EXECUTE BLOCK и т.д.);

• если оператор UPDATE/DELETE расположена на каком-то явном курсоре (WHERE CURRENT OF), то Firebird пропускает шаг (c) выше, то есть не извлекает и не устанавливает блокировки записи для оставшихся записей курсора;

• если оператор верхнего уровня SELECT (или EXECUTE BLOCK возвращающий набор данных) и конфликт обновления происходит после того, как одна или несколько записей были возвращены приложению, то ошибка конфликта обновления сообщается как обычно и перезапуск не инициируется;
• рестарт не инициируется для операторов в автономных блоках (IN AUTONOMOUS TRANSACTION DO …);
• после 10 попыток Firebird прерывает алгоритм перезапуска, снимает все блокировки записи, восстанавливает режим изоляции транзакции как READ COMMITTED READ CONSISTENCY и сообщает о конфликте обновления;
• любая не обработанная ошибка на шаге (с) выше останавливает алгоритм перезапуска, и Firebird продолжает обработку обычным способом, например, ошибка может была перехвачена и обработана блоком PSQL WHEN или сообщена приложению, если она не обработана;
• триггеры UPDATE/DELETE срабатывают многократно для одной и той же записи, если выполнение оператора было перезапущено и запись обновлена/удалена снова;
• по историческим причинам isc_update_conflict сообщается как вторичный код ошибки с первичным кодом ошибки isc_deadlock.

NO AUTO UNDO

При использовании опции NO AUTO UNDO оператор ROLLBACK только помечает транзакцию как отменённую без удаления созданных в этой транзакции версий, которые будут удалены позднее в соответствии с выбранной политикой сборки мусора (см. параметр GCPolicy в firebird.conf).

Эта опция может быть полезна при выполнении транзакции, в рамках которой производится много отдельных операторов, изменяющих данные, и при этом есть уверенность, что эта транзакция будет чаще всего завершаться успешно, а не откатываться.

Для транзакций, в рамках которых не выполняется никаких изменений, опция NO AUTO UNDO игнорируется.

IGNORE LIMBO

При указании опции IGNORE LIMBO игнорируются записи, создаваемые “потерянными” (т.е. не завершёнными) транзакциями (limbo transaction). Транзакции считаются “потерянной”, если не завершён второй этап двухфазного подтверждения (two-phase commit).

AUTO COMMIT

При указании опции AUTO COMMIT транзакция автоматически подтверждается после успешного выполнения любого оператора. Если в процессе выполнения оператора произойдет ошибка, то транзакция будет отката. После подтверждения или отката транзакция продолжает оставаться активной, сохраняя свой идентификатор.

Опция AUTO COMMIT использует “мягкое” подтверждение (COMMIT RETAIN) и “мягкий” откат (ROLLBACK RETAIN) транзакции. Мягкое подтверждение не освобождает ресурсов сервера и удерживает сборку мусора, что может негативно отразиться на производительности.
Предложение RESERVING в операторе SET TRANSACTION резервирует указанные в списке таблицы. Резервирование запрещает другим транзакциям вносить в эти таблицы изменения или (при определённых установках характеристик предложения резервирования) даже читать данные из этих таблиц, в то время как выполняется данная транзакция. Либо, наоборот, в этом предложении можно указать список таблиц, в которые параллельные транзакции могут вносить изменения, даже если запускается транзакция с уровнем изоляции SNAPSHOT TABLE STABILITY.

В одном предложении резервирования можно указать произвольное количество резервируемых таблиц используемой базы данных.

Если опущено одно из ключевых слов SHARED или PROTECTED, то предполагается SHARED. Если опущено все предложение FOR, то предполагается FOR SHARED READ. Варианты осуществления резервирования таблиц по их названиям не являются очевидными.

**Table 248. Совместимости различных блокировок**

<table>
<thead>
<tr>
<th></th>
<th>SHARED READ</th>
<th>SHARED WRITE</th>
<th>PROTECTED READ</th>
<th>PROTECTED WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARED READ</td>
<td>да</td>
<td>да</td>
<td>да</td>
<td>да</td>
</tr>
<tr>
<td>SHARED WRITE</td>
<td>да</td>
<td>да</td>
<td>нет</td>
<td>нет</td>
</tr>
<tr>
<td>PROTECTED READ</td>
<td>да</td>
<td>нет</td>
<td>да</td>
<td>нет</td>
</tr>
<tr>
<td>PROTECTED WRITE</td>
<td>да</td>
<td>нет</td>
<td>нет</td>
<td>нет</td>
</tr>
</tbody>
</table>

Для транзакции запущенной в режиме изоляированности SNAPSHOT для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах их резервирования следующие варианты поведения:

- **SHARED READ** — не оказывает никакого влияния на выполнение параллельных транзакций;
- **SHARED WRITE** — на поведение параллельных транзакций с уровнями изоляированности SNAPSHOT и READ COMMITTED не оказывает никакого влияния, для транзакций с уровнем изоляированности SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;
- **PROTECTED READ** — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изоляированности, попытка внесения изменений приводит к исключению базы данных;
- **PROTECTED WRITE** — для параллельных транзакций с уровнями изоляированности SNAPSHOT и READ COMMITTED запрещает запись в указанные таблицы, для транзакций с уровнем изоляированности SNAPSHOT TABLE STABILITY запрещает также и чтение данных из резервируемых таблиц.

Для транзакции запущенной в режиме изоляированности SNAPSHOT TABLE STABILITY для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от
их уровня изолированности допустимы при различных способах их резервирования следующие варианты поведения:

• **SHARED READ** — позволяет всем параллельным транзакциям независимо от их уровня изолированности не только читать, но и выполнять любые изменения в резервируемых таблицах (если параллельная транзакция имеет режим доступа READ WRITE);

• **SHARED WRITE** — для всех параллельных транзакций с уровнем доступа READ WRITE и с уровнями изолированности SNAPSHOT и READ COMMITTED позволяет читать данные из таблиц и писать данные в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;

• **PROTECTED READ** — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изолированности;

• **PROTECTED WRITE** — для параллельных транзакций с уровнями изолированности SNAPSHOT и READ COMMITTED запрещает запись в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает также и чтение данных из резервируемых таблиц.

Для транзакции запущенной в режиме изолированности READ COMMITTED для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах их резервирования следующие варианты поведения:

• **SHARED READ** — позволяет всем параллельным транзакциям независимо от их уровня изолированности не только читать, но и выполнять любые изменения в резервируемых таблицах (при уровне доступа READ WRITE);

• **SHARED WRITE** — для всех транзакций с уровнем доступа READ WRITE и с уровнями изолированности SNAPSHOT и READ COMMITTED позволяет читать и писать данные в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;

• **PROTECTED READ** — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изолированности;

• **PROTECTED WRITE** — для параллельных транзакций с уровнями изолированности SNAPSHOT и READ COMMITTED разрешает только чтение данных и запрещает запись в указанные в данном списке таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только изменение данных, но и чтение данных из резервируемых таблиц.

Предложение **USING** может быть использовано для сохранения системных ресурсов за счёт ограничения количества баз данных, к которым имеет доступ транзакция. Доступно только в Embedded SQL.

*См. также:*

**COMMIT, ROLLBACK.**
13.2. COMMIT

Назначение
Подтверждение транзакции.

Доступно в
DSQL, ESQL

Синтаксис

COMMIT [WORK] [TRANSACTION tr_name] [RELEASE] [RETAIN [SNAPSHOT]];

Table 249. Параметры оператора COMMIT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>tr_name</td>
<td>Имя транзакции. Доступно только в ESQL.</td>
</tr>
</tbody>
</table>

Оператор COMMIT подтверждает все изменения в данных, выполненные в контексте данной транзакции (добавления, изменения, удаления). Новые версии записей становятся доступными для других транзакций, и если предложение RETAIN не используется, то освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

Если в процессе подтверждения транзакции возникли ошибки в базе данных, то транзакция не подтверждается. Пользовательская программа должна обработать ошибочную ситуацию и заново подтвердить транзакцию или выполнить ее откат.

Необязательное предложение TRANSACTION задаёт имя транзакции. Предложение TRANSACTION доступно только в Embedded SQL. Если предложение TRANSACTION не указано, то оператор COMMIT применяется к транзакции по умолчанию.

За счёт именованных транзакций позволяет одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноименная переменная базового языка. В DSQL, это ограничение предотвращает динамическую спецификацию имен транзакций.

Необязательное ключевое слово WORK может быть использовано лишь для совместимости с другими системами управления реляционными базами данных.

Ключевое слово RELEASE доступно только в Embedded SQL. Оно позволяет отключиться ото всех баз данных после завершения текущей транзакции. RELEASE поддерживается только для обратной совместимости со старыми версиями Interbase. В настоящее время вместо него используется оператор ESQL DISCONNECT.

Если используется предложение RETAIN [SNAPSHOT], то выполняется так называемое мягкое (soft) подтверждение. Выполненные действия в контексте данной транзакции фиксируются в базе данных, а сама транзакция продолжает оставаться активной, сохраняя свой
идентификатор, а также состояние курсоров, которое было до мягкой фиксации транзакции. В этом случае нет необходимости опять стартовать транзакцию и заново выполнять оператор SELECT для получения данных.

Если уровень изоляции такой транзакции SNAPSHOT или SNAPSHOT TABLE STABILITY, то после мягкого подтверждения транзакция продолжает видеть состояние базы данных, которое было при первоначальном запуске транзакции, то есть клиентская программа не видит новых подтверждённых результатов изменения данных других транзакций. Кроме того, мягкое подтверждение не освобождает ресурсов сервера (открытые курсоры не закрываются).

Для транзакций, которые выполняют только чтение данных из базы данных, рекомендуется также использовать оператор COMMIT, а не ROLLBACK, поскольку этот вариант требует меньшего количества ресурсов сервера и улучшает производительность всех последующих транзакций.

**Cм. также:**

SET TRANSACTION, ROLLBACK.

### 13.3. ROLLBACK

**Назначение**

Откат транзакции.

**Доступно в**

DSQL, ESQL

**Синтаксис**

```
ROLLBACK [WORK] [TRANSACTION tr_name]
 [RETAIN [SNAPSHOT] | TO SAVEPOINT sp_name] [RELEASE];
```

**Table 250. Параметры оператора ROLLBACK**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>tr_name</td>
<td>Имя транзакции. Доступно только в ESQL.</td>
</tr>
<tr>
<td>sp_name</td>
<td>Имя точки сохранения. Доступно только в DSQL.</td>
</tr>
</tbody>
</table>

Оператор ROLLBACK отменяет все изменения данных базы данных (добавление, изменение, удаление), выполненные в контексте этой транзакции. Оператор ROLLBACK никогда не вызывает ошибок. Если не указано предложение RETAIN, то при его выполнении освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

Необязательное предложение TRANSACTION задаёт имя транзакции. Предложение TRANSACTION доступно только в Embedded SQL. Если предложение TRANSACTION не указано, то оператор ROLLBACK применяется к транзакции по умолчанию.
За счёт именованных транзакций позволяет одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноименная переменная базового языка. В DSQL, это ограничение предотвращает динамическую спецификацию имён транзакций.

Необязательное ключевое слово WORK может быть использовано лишь для совместимости с другими системами управления реляционными базами данных.

Ключевое слово RETAIN указывает, что все действия по изменению данных в контексте этой транзакции, отменяются, а сама транзакция продолжает оставаться активной, сохраняя свой идентификатор, а также состояние курсоров, которое было до мягкой фиксации транзакции. Таким образом, выделенные ресурсы для транзакции не освобождаются.

Для уровней изоляции SNAPSHOT и SNAPSHOT TABLE STABILITY состояние базы данных остаётся в том виде, которое база данных имела при первоначальном старте такой транзакции, однако в случае уровня изоляции READ COMMITTED база данных будет иметь вид, соответствующий новому состоянию на момент выполнения оператора ROLLBACK RETAIN. В случае отмены транзакции с сохранением её контекста нет необходимости заново выполнять оператор SELECT для получения данных из таблицы.

См. также:

SET TRANSACTION, COMMIT.

13.3.1. ROLLBACK TO SAVEPOINT

Необязательное предложение TO SAVEPOINT в операторе ROLLBACK задаёт имя точки сохранения, на которую происходит откат. В этом случае отменяются все изменения, произошедшие в рамках транзакции, начиная с созданной точки сохранения (SAVEPOINT).

Оператор ROLLBACK TO SAVEPOINT выполняет следующие операции:

• Все изменения в базе данных, выполненные в рамках транзакции начиная с созданной точки сохранения, отменяются. Пользовательские переменные, заданные с помощью функции RDB$SET_CONTEXT() остаются неизменными;

• Все точки сохранения, создаваемые после названной, уничтожаются. Все более ранние точки сохранения, как сама точка сохранения, остаются. Это означает, что можно откатываться к той же точке сохранения несколько раз;

• Все явные и неявные блокированные записи, начиная с точки сохранения, освобождаются. Другие транзакции, запросившие ранее доступ к строкам, заблокированным после точки сохранения, должны продолжать ожидать, пока транзакция не фиксируется или откатывается. Другие транзакции, которые ещё не запрашивали доступ к этим строкам, могут запросить и сразу же получить доступ к разблокированным строкам.

См. также:

SAVEPOINT.
13.4. SAVEPOINT

Назначение
Создание точки сохранения.

Доступно в
DSQL

Синтаксис

SAVEPOINT sp_name

Table 251. Параметры оператора SAVEPOINT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_name</td>
<td>Имя точки сохранения. Должно быть уникальным в рамках транзакции.</td>
</tr>
</tbody>
</table>

Оператор SAVEPOINT создаёт SQL:99 совместимую точку сохранения, к которой можно позже откатьывать работу с базой данных, не отменяя все действия, выполненные с момента старта транзакции. Механизмы точки сохранения также известны под термином "вложенные транзакции" ("nested transactions").

Если имя точки сохранения уже существует в рамках транзакции, то существующая точка сохранения будет удалена, и создаётся новая с тем же именем.

Для отката изменений к точке сохранения используется оператор ROLLBACK TO SAVEPOINT.

Внутренний механизм точек сохранения может использовать большие объёмы памяти, особенно если вы обновляете одни и те же записи многократно в одной транзакции. Если точка сохранения уже не нужна, но вы ещё не готовы закончить транзакцию, то можно ее удалить оператором RELEASE SAVEPOINT, тем самым освобождая ресурсы.

Example 411. DSQL сессия с использованием точек сохранения

```sql
CREATE TABLE TEST (ID INTEGER);
COMMIT;
INSERT INTO TEST VALUES (1);
COMMIT;
INSERT INTO TEST VALUES (2);
SAVEPOINT Y;
DELETE FROM TEST;
SELECT * FROM TEST; -- возвращает пустую строку
ROLLBACK TO Y;
SELECT * FROM TEST; -- возвращает две строки
ROLLBACK;
```
См. также:
ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT.

13.5. RELEASE SAVEPOINT

Назначение
Удаление точки сохранения.

Доступно в
DSQL

Синтаксис

```
RELEASE SAVEPOINT sp_name [ONLY]
```

Table 252. Параметры оператора RELEASE SAVEPOINT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_name</td>
<td>Имя точки сохранения</td>
</tr>
</tbody>
</table>

Оператор RELEASE SAVEPOINT удаляет именованную точку сохранения, освобождая все связанные с ней ресурсы. По умолчанию удаляются также все точки сохранения, создаваемые после указанной. Если указано предложение ONLY, то удаляется только точка сохранения с заданным именем.

См. также:
SAVEPOINT.

13.6. Внутренние точки сохранения

По умолчанию сервер использует автоматическую системную точку сохранения уровня транзакции для выполнения её отката. При выполнении оператора ROLLBACK, все изменения, выполненные в транзакции, откатываются до системной точки сохранения и после этого транзакция подтверждается.

Когда объем изменений, выполняемых под системной точкой сохранения уровня транзакции, становится большим (затрагивается порядка 50000 записей), сервер освобождает системную точку сохранения и, при необходимости отката транзакции, использует механизм TIP.

Если вы ожидаете, что объем изменений в транзакции будет большим, то можно задать опцию NO AUTO UNDO в операторе SET TRANSACTION, или — если используется API — установить флаг TPB isc_tpb_no_auto_undo. В обеих вариантах предотвращается создание системной точки сохранения уровня
13.7. Точки сохранения и PSQL

Использование операторов управления транзакциями в PSQL не разрешается, так как это нарушит атомарность оператора, вызывающего процедуру. Но Firebird поддерживает вызов и обработку исключений в PSQL, так, чтобы действия, выполняемые в хранимых процедурах и триггерах, могли быть выборочно отменены без полного отката всех действий в них. Внутренне автоматические точки сохранения используется для:

- отмены всех действий внутри блока `BEGIN … END`, где происходит исключение;
- отмены всех действий, выполняемых в хранимой процедуре/триггере (или, в случае селективной хранимой процедуры, всех действий, выполненных с момента последнего оператора `SUSPEND`), если они завершаются преждевременно из-за непредусмотренной ошибки или исключения.

Каждый блок обработки исключений PSQL также ограничен автоматическими точками сохранения сервера.

Сами по себе блок `BEGIN … END` не создаёт автоматическую точку сохранения. Она создаётся только в блоках, которых присутствует блок `WHEN` для обработки исключений или ошибок.
## Chapter 14. Безопасность

Базы данных, как и данные, хранимые в файлах базы данных, должны быть защищены. Firebird обеспечивает двухуровневую защиту данных — аутентификация пользователя на уровне сервера и привилегии на уровне базы данных. В данной главе рассказывается, каким образом управлять безопасностью вашей базы данных на каждом из уровней.

### 14.1. Аутентификация пользователя

Безопасность всей базы данных зависит от проверки подлинности идентификатора пользователя. Подлинность пользователя может выполняться различными способами в зависимости от установок параметра AuthServer в файле конфигурации firebird.conf. Этот параметр содержит список доступных плагинов проверки подлинности. Если проверить подлинность с помощью первого плагина не удалось, то сервер перейдет к следующему плагину и т.д. Если ни один плагин не подтвердил подлинность, то пользователь получает сообщение об ошибке.

Информация о пользователях, зарегистрированных для конкретного сервера Firebird, хранится в особой базе данных безопасности (security database) — security4.fdb. Для каждой БД база данных безопасности может переопределена в файле databases.conf (параметр SecurityDatabase). Любая база данных может быть базой данных безопасности для самой себя.

Имя пользователя может состоять максимум из 63 символов. Максимальная длина пароля зависит от плагина проверки подлинности и плагина управления пользователями (параметр UserManager), регистр — учитывается. По умолчанию будет выбран первый плагин из списка плагинов управления пользователями. Этот плагин можно изменить в SQL командах управления пользователями. Для плагина SRP эффективная длина пароля ограничена 20 байтами.* Для плагина Legacy_UserManager максимальная длина пароля равна 8 байт.

*Почему эффективная длина пароля ограничена 20 символами?*

На длину пароля нет ограничения в 20 байт и он может быть использован. Хеши различных паролей, длина которых более 20 байт, тоже различны. Предел эффективности наступает из-за ограниченной длины хеша в SHA1 равном 20 байт или 160 бит. Рано или поздно найдётся более короткий пароль с тем же хешем с помощью атаки Brute Force. Именно поэтому часто говорят, что эффективная длина пароля для алгоритма SHA1 составляет 20 байт.

Встроенная версия сервера (embedded), не использует аутентификацию. Тем не менее имя пользователя, и если необходимо роль, должны быть указаны в параметрах подключения, поскольку они используются для контроля доступа к объектам базы данных.

Пользователь SYSDBA или пользователь вошедший с ролью RDB$ADMIN, получают неограниченный доступ к базе данных. Если пользователь является владельцем базы данных, то без указания роли RDB$ADMIN он получает неограниченный доступ ко всем
объектам принадлежащим этой базе данных.

14.1.1. Специальные учётные записи

SYSDBA

В Firebird существует специальная учётная запись SYSDBA, которая существует вне всех ограничений безопасности и имеет полный доступ ко всем базам данных сервера.

Особенности POSIX

В POSIX системах, включая MacOSX, имя пользователя POSIX будет интерпретировано как имя пользователя Firebird Embedded, если имя пользователя не указано явно.

Пользователь SYSDBA в POSIX

В POSIX системах, кроме MacOSX, пользователь SYSDBA не имеет пароля по умолчанию. Если полная установка осуществляется с помощью стандартных скриптов, то одноразовый пароль будет создан и сохранён в текстовом файле в том же каталоге, что и security4.fdb, обычно это /opt/firebird/. Файл с паролем имеет имя SYSDBA.password.

При выполнении установки с помощью определённого распространяемого установщика, расположение файла базы данных безопасности и файла с паролем может отличаться от стандартного.

Пользователь root в POSIX

В POSIX системах пользователь root может выступать в роли SYSDBA. Firebird Embedded в этом случае будет трактовать имя пользователя root как SYSDBA, и вы будете иметь доступ ко всем базам данных сервера.

Особенности Windows

В операционных системах семейства Windows NT вы также можете пользоваться учётными записями OC. Для этого необходимо, чтобы в файле конфигурации firebird.conf (параметр AuthServer) в списке плагинов присутствовал провайдер Win_Sspi. Кроме того, этот плагин должен присутствовать и в списке плагинов клиентской стороны (параметр AuthClient).

Администраторы операционной системы Windows автоматически не получают права SYSDBA при подключении к базе данных (если, конечно, разрешена доверенная авторизация). Имеет ли администраторы автоматические права SYSDBA, зависит от установки значения флага AUTO ADMIN MAPPING.

До Firebird 3.0 при включенной доверительной аутентификации, пользователи прошедшие проверку по умолчанию автоматически отображались в CURRENT_USER. В Firebird 3 и выше отображение должно быть сделано явно для систем с несколькими базами данных безопасности и включенной доверительной аутентификацией. См. CREATE MAPPING.
14.1.2. Владелец базы данных

Владелец базы данных — это либо текущий пользователь (CURRENT_USER), который был момент создания, либо пользователь который был указан в параметрах USER в операторе CREATE DATABASE.

Владелец базы данных является администратором в ней и имеет полный доступ ко всем объектам этой базы данных, даже созданных другими пользователями.

14.1.3. Роль RDB$ADMIN

Системная роль RDB$ADMIN, присутствует в каждой базе данных. Предоставление пользователю роли RDB$ADMIN в базе данных даёт ему права SYSDBA, но только в текущей базе данных.

Привилегии вступают в силу сразу после входа в обычную базу данных с указанием роли RDB$ADMIN, после чего пользователь получает полный контроль над всеми объектами базы данных.

Роль RDB$ADMIN может быть предоставлена с использованием ключевого слова DEFAULT. В этом случае пользователь автоматически будет получать административные привилегии даже если он не указал роль RDB$ADMIN при входе.

Предоставление в базе данных безопасности даёт возможность создавать, изменять и удалять учётные записи пользователей.

В обоих случаях пользователь с правами RDB$ADMIN роли может всегда передавать эту роль другим. Другими словами, “WITH ADMIN OPTION” уже встроен в эту роль и эту опцию можно не указывать.

Предоставление роли RDB$ADMIN в обычной базе данных

Для предоставления и удаления роли RDB$ADMIN в обычной базе данных используются операторы GRANT и REVOKE, как и для назначения и отмены остальных ролей.

Синтаксис

```
GRANT [DEFAULT] [ROLE] RDB$ADMIN TO username
REVOKE [DEFAULT] [ROLE] RDB$ADMIN FROM username
```

Table 253. Параметры операторов установки и отмены роли RDB$ADMIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Имя пользователя, которому назначается или отбирается роль RDB$ADMIN.</td>
</tr>
</tbody>
</table>

Если в операторе GRANT присутствует ключевое слово DEFAULT, то пользователь автоматически будет получать административные привилегии даже если он не указал роль RDB$ADMIN при входе. Привилегии на роль RDB$ADMIN могут давать только
Использование роли RDB$ADMIN в обычной базе данных

Для использования прав роли RDB$ADMIN пользователь просто указывает её при соединении с базой данных, или же роль RDB$ADMIN выдали пользователю с использованием ключевого слова DEFAULT. Он также может указать её позднее с помощью оператора SET ROLE.

Предоставление роли RDB$ADMIN в базе данных пользователей

Так как никто не может соединиться с базой данных пользователей, то операторы GRANT и REVOKE здесь не могут использоваться. Вместо этого роль RDB$ADMIN предоставляют и удаляют SQL командами управления пользователями: CREATE USER и ALTER USER, в которых указываются специальные опции GRANT ADMIN ROLE и REVOKE ADMIN ROLE.

Синтаксис (неполный)

```
CREATE USER newuser
  PASSWORD 'password'
...
GRANT ADMIN ROLE
...
ALTER USER existinguser
  GRANT ADMIN ROLE

ALTER USER existinguser
  REVOKE ADMIN ROLE
```

Table 254. Параметры операторов установки и отмены роли RDB$ADMIN

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>newuser</td>
<td>Имя вновь создаваемого пользователя. Максимальная длина 63 символа.</td>
</tr>
<tr>
<td>existinguser</td>
<td>Имя существующего пользователя.</td>
</tr>
<tr>
<td>password</td>
<td>Пароль пользователя. Чувствительно к регистру.</td>
</tr>
</tbody>
</table>

Пожалуйста, помните, что GRANT ADMIN ROLE и REVOKE ADMIN ROLE это не операторы GRANT и REVOKE. Это параметры для CREATE USER и ALTER USER.

Привилегии на роль RDB$ADMIN могут давать только администраторы.

См. также:

GRANT, REVOKE.
Выполнение той же задачи используя утилиту gsec

То же самое можно сделать используя утилиту gsec указав параметр -admin для сохранения атрибута RDB$ADMIN учётной записи пользователя:

```
gsec -add new_user -pw password -admin yes
   gsec -mo existing_user -admin yes
   gsec -mo existing_user -admin no
```

В зависимости от административного статуса текущего пользователя для утилиты gsec может потребоваться больше параметров, таких как -user и -pass, или -trusted.

Использование роли RDB$ADMIN в базе данных пользователей

Для управления учётными записями пользователей через SQL пользователь, имеющий права на роль RDB$ADMIN, должен подключиться к базе данных с этой ролью. Так как к базе данных пользователей не имеет права соединяться никто, то пользователь должен подключиться к обычной базе данных, где он также имеет права на роль RDB$ADMIN. Он определяет роль при соединении с обычной базой данных и может в ней выполнить любой SQL запрос. Это не самое элегантное решение, но это единственный способ управлять пользователями через SQL запросы.

Если нет обычной базы данных, где у пользователя есть права на роль RDB$ADMIN, то управление учётными записями посредством SQL запросов недоступно.

Использование роли RDB$ADMIN в gsec

Для управления пользователями через утилиту gsec роль RDB$ADMIN должна быть указана в переключателе -role.

**AUTO ADMIN MAPPING**

*Операционная система*

только Windows.

Администраторы операционной системы Windows автоматически не получают права SYSDBA при подключении к базе данных (если, конечно, разрешена доверенная авторизация). Имеют ли администраторы автоматические права SYSDBA зависит от установки значения флага AUTO ADMIN MAPPING. Это флаг в каждой из баз данных, который по умолчанию выключен. Если флаг AUTO ADMIN MAPPING включен, то он действует, когда администратор Windows:

a. подключается с помощью доверенной аутентификации
b. не определяет никакой роли при подключении.

После успешного “auto admin” подключения текущей ролью будет являться RDB$ADMIN.
Включение и выключение флага AUTO ADMIN MAPPING в обычной базе данных осуществляется следующим образом:

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING  -- включение
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING  -- выключение
```

Эти операторы могут быть выполнены пользователями с достаточными правами, а именно:

- владелец базы данных;
- администраторы.

Оператор

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING
```

является упрощённым видом оператора создания отображения предопределённой группы `DOMAIN_ANY_RID_ADMINS` на роль `RDB$ADMIN`:

```
CREATE MAPPING WIN_ADMINS
  USING PLUGIN WIN_SSPI
  FROM Predefined_Group DOMAIN_ANY_RID_ADMINS
  TO ROLE RDB$ADMIN;
```

Соответственно оператор

```
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING
```

эквивалентен оператору

```
DROP MAPPING WIN_ADMINS;
```

Подробней см. Отображение объектов безопасности.

В обычных базах данных статус AUTO ADMIN MAPPING проверяется только во время подключения. Если Administrtor имеет роль RDB$ADMIN потому, что произошло автоматическое отображение во время входа, то он будет удерживать эту роль на протяжении всей сессии, даже если он или кто-то другой в это же время выключает автоматическое отображение.

Точно также, включение AUTO ADMIN MAPPING не изменит текущую роль в RDB$ADMIN для
Администраторов, которые уже подключились.

Включение и выключение AUTO ADMIN MAPPING в базе данных безопасности

Оператором ALTER ROLE RDB$ADMIN невозможно включить или выключить флаг AUTO ADMIN MAPPING в базе данных пользователей. Однако вы можете создать глобальное отображение предопределённой группы DOMAIN_ANY_RID_ADMINS на роль RDB$ADMIN следующим образом:

```
CREATE GLOBAL MAPPING WIN_ADMINS USING PLUGIN WIN_SSPI
FROM Predefined_Group
DOMAIN_ANY_RID_ADMINS
TO ROLE RDB$ADMIN;
```

Кроме того для включения AUTO ADMIN MAPPING в базе данных пользователей можно использовать утилиту командной строки gsec:

```
gsec -mapping set

gsec -mapping drop
```

В зависимости от административного статуса текущего пользователя для утилиты gsec может потребоваться больше параметров, таких как -user и -pass, или -trusted.

Только SYSDBA может включить AUTO ADMIN MAPPING, если он выключен, но любой администратор может выключить его.

При выключении AUTO ADMIN MAPPING пользователь отключает сам механизм, который предоставлял ему доступ и, таким образом, он не сможет обратно включить AUTO ADMIN MAPPING. Даже в интерактивном gsec сеансе новая установка флага сразу вступает в силу.

14.1.4. Администраторы

Администратор — это пользователь, которые имеет достаточные права для чтения и записи, создания, изменения и удаления любого объекта в базе данных. В таблице показано, как привилегии “Суперпользователя” включены в различных контекстах безопасности Firebird.

<table>
<thead>
<tr>
<th>Пользователь</th>
<th>Роль RDB$ADMIN</th>
<th>Замечание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDBA</td>
<td>Автоматически</td>
<td>Существует автоматически на уровне сервера. Имеет полные привилегии ко всем объектам во всех базах данных. Может создавать, изменять и удалять пользователей, но не имеет прямого доступа к базе данных безопасности.</td>
</tr>
</tbody>
</table>

Chapter 14. Безопасность
<table>
<thead>
<tr>
<th>Пользователь</th>
<th>Роль RDB$ADMIN</th>
<th>Замечание</th>
</tr>
</thead>
<tbody>
<tr>
<td>Пользователь root в POSIX</td>
<td>Автоматически</td>
<td>Так же как SYSDBA. Только в Firebird Embedded.</td>
</tr>
<tr>
<td>Суперпользователь в POSIX</td>
<td>Автоматически</td>
<td>Так же как SYSDBA. Только в Firebird Embedded.</td>
</tr>
<tr>
<td>Владелец базы данных</td>
<td>Автоматически</td>
<td>Так же как SYSDBA, но только в этой базе данных.</td>
</tr>
<tr>
<td>Администраторы Windows</td>
<td>Устанавливается в CURRENT_ROLE, если вход успешен</td>
<td>Так же как SYSDBA, если соблюдаются следующие условия:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• В файле конфигурации firebird.conf (параметр AuthServer) в списке плагинов присутствовал провайдер Win_Sspi. Кроме того, этот плагин должен присутствовать и в списке плагинов клиентской стороны (параметр AuthClient).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Во всех базах данных, где требуется полномочия суперпользователя должен быть включен AUTO ADMIN MAPPING или создано отображение предопределенной группы DOMAIN_ANY_RID_ADMINS на роль RDB$ADMIN.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• При входе не указана роль.</td>
</tr>
<tr>
<td>Обычный пользователь</td>
<td>Должна быть предварительно выдана и должна быть указана при входе</td>
<td>Так же как SYSDBA, но только в тех базах данных, где эта роль предоставлена.</td>
</tr>
<tr>
<td>Пользователь POSIX</td>
<td>Должна быть предварительно выдана и должна быть указана при входе</td>
<td>Та кже как SYSDBA, но только в тех базах данных, где эта роль предоставлена. Только в Firebird Embedded.</td>
</tr>
<tr>
<td>Пользователь Windows</td>
<td>Должна быть предварительно выдана и должна быть указана при входе</td>
<td>Так же как SYSDBA, но только в тех базах данных, где эта роль предоставлена. Доступно только если в файле конфигурации firebird.conf (параметр AuthServer) в списке плагинов присутствовал провайдер Win_Sspi. Кроме того, этот плагин должен присутствовать и в списке плагинов клиентской стороны (параметр AuthClient).</td>
</tr>
</tbody>
</table>

14.2. Управление пользователями

В данном разделе описываются операторы создания, модификации и удаления учётных записей пользователей Firebird средствами операторов SQL. Такая возможность предоставлена следующим пользователям:
• SYSDBA;
• Любому пользователю, имеющему права на роль RDB$ADMIN в базе данных пользователей и права на ту же роль для базы данных в активном подключении. Пользователь должен подключаться к базе данных с ролью RDB$ADMIN или получить её права, если роль назначена в качестве роли по умолчанию;
• Любому пользователю с ролью, которой назначена системная привилегия USER_MANAGEMENT в базе данных безопасности. Пользователь должен подключаться к базе данных с этой ролью или получить её права, если роль назначена в качестве роли по умолчанию;
• При включенном флаге AUTO ADMIN MAPPING в базе данных пользователей (security4.fdb или той, что установлена для вашей базы данных в файле databases.conf) — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication) без указания роли. При этом не важно, включен или выключен флаг AUTO ADMIN MAPPING в самой базе данных.

Непривилегированные пользователи могут использовать только оператор ALTER USER для изменения собственной учётной записи.

14.2.1. CREATE USER

Назначение

Создание учётной записи пользователя Firebird.

Доступно в

DSQL

Синтаксис

```
CREATE USER username
    PASSWORD 'password'
    [<user_option> [<user_option> ...]]
    [TAGS ( <tag> [, <tag> ...] )]

<user_option> ::= 
    FIRSTNAME 'firstname'
    | MIDDLENAME 'middlename'
    | LASTNAME 'lastname'
    | {ACTIVE | INACTIVE}
    | USING PLUGIN pluginname
    | GRANT ADMIN ROLE

<tag> ::= 
    tagname = 'string_value'
```

Table 256. Параметры оператора CREATE USER
<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Имя пользователя. Максимальная длина 63 символов.</td>
</tr>
<tr>
<td>password</td>
<td>Пароль пользователя. Чувствительно к регистру.</td>
</tr>
<tr>
<td>firstname</td>
<td>Вспомогательная информация: имя пользователя. Максимальная длина 32 символа.</td>
</tr>
<tr>
<td>middlename</td>
<td>Вспомогательная информация: &quot;второе имя&quot; (отчество, &quot;имя отца&quot;) пользователя. Максимальная длина 32 символа.</td>
</tr>
<tr>
<td>lastname</td>
<td>Вспомогательная информация: фамилия пользователя. Максимальная длина 32 символа.</td>
</tr>
<tr>
<td>pluginname</td>
<td>Имя плагина управления пользователями, в котором необходимо создать нового пользователя.</td>
</tr>
<tr>
<td>tagname</td>
<td>Имя пользовательского атрибута. Максимальная длина 63 символа. Имя атрибута должно подчиняться правилам наименования SQL идентификаторов.</td>
</tr>
<tr>
<td>string_value</td>
<td>Значение пользовательского атрибута. Максимальная длина 255 символов.</td>
</tr>
</tbody>
</table>

Оператор `CREATE USER` создаёт учётную запись пользователя Firebird. Пользователь должен отсутствовать в текущей базе данных безопасности Firebird иначе будет выдано соответствующее сообщение об ошибке.

Начиная с Firebird 3.0 имена пользователей подчиняются общему правилу наименования идентификаторов объектов метаданных. Таким образом, пользователь с именем "Alex" и с именем "ALEX" будут разными пользователями.

```
CREATE USER ALEX PASSWORD 'bz23ds';
-- это пользователь такой же как и первый
CREATE USER Alex PASSWORD 'bz23ds';
-- это пользователь такой же как и первый
CREATE USER "ALEX" PASSWORD 'bz23ds';
-- это уже другой пользователь
CREATE USER "Alex" PASSWORD 'bz23ds';
```

Предложение `PASSWORD` задаёт пароль пользователя. Максимальная длина пароля зависит от того какой плагин управления пользователями задействован (параметр `UserManager`). Для плагина SRP эффективная длина пароля ограничена 20 байтами *. Плагин Legacy_UserManager максимальная длина пароля равна 8 байт.

*Почему эффективная длина пароля ограничена 20 символами?*

На длину пароля нет ограничения в 20 байт и он может быть использован.

* `.`
Хеши различных паролей, длина которых более 20 байт, тоже различны. Предел эффективности наступает из-за ограниченной длины хеша в SHA1 равном 20 байт или 160 бит. Рано или поздно найдётся более короткий пароль с тем же хешем с помощью атаки Brute Force. Именно поэтому часто говорят, что эффективная длина пароля для алгоритма SHA1 составляет 20 байт.

Необязательные предложения FIRSTNAME, MIDDLENAME и LASTNAME задают дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Если при создании учётной записи будет указан атрибут INACTIVE, то пользователь будет создан в “неактивном состоянии”, т.е. подключиться с его учётной записью будет невозможно. При указании атрибута ACTIVE пользователь будет создан в активном состоянии. По умолчанию пользователь создаётся активным. Данная возможность доступна только при использовании алгоритма SHA в качестве менеджера пользователей.

Если указана опция GRANT ADMIN ROLE, то новая учётная запись пользователя создаётся с правами роли RDB$ADMIN в текущей базе данных безопасности. Это позволяет вновь созданному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра UserManager в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре UserManager.

Важно
Учтите что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи. Поэтому пользователя созданного с помощью одного плагина управления пользователями можно удалить или изменить, указав только тот же самый плагин.

Кроме того вы можете задать неограниченное количество пользовательских атрибутов с помощью необязательного предложения TAGS. Данная возможность доступна только при использовании Srp в качестве менеджера пользователей.

Кто может создать пользователя

• SYSDBA и другие пользователи являющиеся администраторами в базе данных безопасности (с ролью RDB$ADMIN);
• Пользователи вошедшие с ролью или получившие её привилегии (роль назначена по умолчанию), которой назначена системная привилегия USER_MANAGEMENT.
Примеры CREATE USER

Example 412. Создание пользователя

```
CREATE USER bigshot PASSWORD 'buckshot';
```

Example 413. Создание пользователя с помощью плагина управления пользователями Legacy_UserManager

```
CREATE USER godzilla PASSWORD 'robot'
USING PLUGIN Legacy_UserManager;
```

Example 414. Создание пользователя с пользовательскими атрибутами.

```
CREATE USER john PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
LASTNAME 'Doe'
TAGS (BIRTHYEAR = '1970', CITY = 'New York');
```

Example 415. Создание пользователя в неактивном состоянии.

```
CREATE USER john PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
LASTNAME 'Doe'
INACTIVE;
```

Example 416. Создание пользователя с возможностью управления пользователями

```
CREATE USER superuser PASSWORD 'kMn8Kjh'
GRANT ADMIN ROLE;
```

См. также:

ALTER USER, CREATE OR ALTER USER, DROP USER.

14.2.2. ALTER USER

Назначение:
Изменение учётной записи пользователя Firebird.
Доступно в: DSQL.

Синтаксис:

```
ALTER {USER username | CURRENT USER}
    [SET] [<user_option> [<user_option> ...]]
    [USING PLUGIN pluginname]
    [{GRANT | REVOKE} ADMIN ROLE]
    [TAGS ( <tag> [, <tag> ...] )]
```

```
[user_option] ::= 
    PASSWORD 'password'
  | FIRSTNAME 'firstname'
  | MIDDLENAME 'middlename'
  | LASTNAME 'lastname'
  | {ACTIVE | INACTIVE}
```

```
<tag> ::= 
    tagname = 'string_value'
  | DROP tagname
```

Описание параметров оператора смотри в **CREATE USER**.

Оператор `ALTER USER` изменяет данные учётной записи пользователя. В операторе `ALTER USER` должно присутствовать хотя бы одно из необязательных предложений.

Необязательное предложение `PASSWORD` задаёт новый пароль пользователя.

Необязательные предложения `FIRSTNAME`, `MIDDLENAME` и `LASTNAME` позволяют изменить дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Атрибут `INACTIVE` позволяет сделать учётную запись неактивной. Это удобно когда необходимо временно отключить учётную запись без её удаления. Атрибут `ACTIVE` позволяет вернуть неактивную учётную запись в активное состояние. Данная возможность доступна только при использовании `SrP` в качестве менеджера пользователей.

Необязательное предложение `TAGS` позволяет задать, изменить или удалить пользовательские атрибуты. Если в списке атрибутов, атрибута с заданным именем не было, то он будет добавлен, иначе его значение будет изменено. Атрибуты не указанные в списке не будут изменены. Для удаления пользовательского атрибута перед его именем в списке атрибутов необходимо указать ключевое слово `DROP`. Данная возможность доступна только при использовании `SrP` в качестве менеджера пользователей.

Предложение `GRANT ADMIN ROLE` предоставляет указанному пользователю привилегии роли `RDB$ADMIN` в текущей базе данных безопасности. Это позволяет указанному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.
Предложение REVOKE ADMIN ROLE отбирает у указанного пользователя привилегии роли RDB$ADMIN в текущей базе данных безопасности. Это запрещает указанному пользователю управлять учётными записями пользователей.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра UserManager в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре UserManager.

Важно:
Учтите что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи. Поэтому пользователя созданного с помощью одного плагина управления пользователями можно удалить или изменить, указав только тот же самый плагин.

Если требуется изменить свою учётную запись, то вместо указания имени текущего пользователя можно использовать ключевое слово CURRENT USER.

**Кто может модифицировать учётную запись?**

Модифицировать чужую учётную запись могут:

- SYSDBA и другие пользователи являющиеся администраторами в базе данных безопасности (с ролью RDB$ADMIN);
- Пользователи вошедшие с ролью или получившие её привилегии (роль назначена по умолчанию), которой назначена системная привилегия USER_MANAGEMENT.

Свои собственные учётные записи могут изменять любые пользователи, однако это не относится к опциям {GRANT | REVOKE} ADMIN ROLE и атрибуту ACTIVE/INACTIVE для изменения которых необходимы административные привилегии.

**Примеры ALTER USER**

*Example 417. Изменение пользователя и выдача ему привилегии управления пользователями.*

```
ALTER USER bobby PASSWORD '67-UiT_G8'
GRANT ADMIN ROLE;
```

*Example 418. Изменение пароля пользователя, созданного с помощью плагина управления пользователями Legacy_UserManager.*

```
ALTER USER godzilla PASSWORD 'robot12'
USING PLUGIN Legacy_UserManager;
```
Изменение дополнительных атрибутов своей учётной записи.

Example 419.

```sql
ALTER CURRENT USER
    FIRSTNAME 'No_Jack'
    LASTNAME 'Kennedy';
```

Отключение учётной записи пользователя.

Example 420.

```sql
ALTER USER dan INACTIVE;
```

Отбор привилегии управления пользователями у пользователя.

Example 421.

```sql
ALTER USER dumbbell
    REVOKE ADMIN ROLE;
```

Изменение пользовательских атрибутов своей учётной записи.

Example 422.

```sql
ALTER CURRENT USER
    TAGS (BIRTHYEAR = '1971', DROP CITY);
```

Атрибуту BIRTHDAY будет установлено новое значение, а атрибут CITY будет удалён.

См. также:

CREATE USER, CREATE OR ALTER USER, DROP USER.

14.2.3. CREATE OR ALTER USER

Назначение

Создание или изменение учётной записи пользователя Firebird.

Доступно в

DSQL

Синтаксис

```sql
ALTER USER _username_
    [SET] [<user_option> [<user_option> ...]]
    [USING PLUGIN _pluginname_]
    [{GRANT | REVOKE} ADMIN ROLE]
    [TAGS ( <tag> [, <tag> ...] )]
```
Описание параметров оператора смотри в CREATE USER.

Оператор CREATE OR ALTER USER создаёт новую или изменяет учётную запись. Если пользователя не существует, то он будет создан с использованием предложения CREATE USER. Если он уже существует, то он будет изменён, при этом существующие привилегии сохраняются.

Примеры CREATE OR ALTER USER

Example 423. Создание или изменение пользователя.

```
CREATE OR ALTER USER john
PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
LASTNAME 'Doe'
INACTIVE;
```

См. также:
CREATE USER, ALTER USER.

14.2.4. DROP USER

Назначение
Удаление учётной записи пользователя Firebird

Доступно в
DSQL

Синтаксис

```
DROP USER username
[USING PLUGIN pluginname]
```

Table 257. Параметры оператора DROP USER
Оператор DROP USER удаляет учётную запись пользователя Firebird.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра UserManager в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре UserManager.

Учитите что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи. Поэтому пользователя созданного с помощью одного плагина управления пользователями можно удалить или изменить, указав только тот же самый плагин.

Кто может удалить учётную запись пользователя?

• SYSDBA и другие пользователи являющиеся администраторами в базе данных безопасности (с ролью RDB$ADMIN);
• Пользователи вошедшие с ролью или получившие её привилегии (роль назначена по умолчанию), которой назначена системная привилегия USER_MANAGEMENT.

Примеры DROP USER

Example 424. Удаление пользователя.

```
DROP USER bobby;
```

Example 425. Удаление пользователя, созданного с помощью плагина управления пользователями Legacy_UserManager.

```
DROP USER Godzilla USING PLUGIN Legacy_UserManager;
```

См. также:

CREATE USER, ALTER USER.

14.2.5. RECREATE USER

Назначение
Создание новой учётной записи пользователя Firebird или пересоздание существующей.

Доступно в

DSQL

Синтаксис

```sql
RECREATE USER username
  PASSWORD 'password'
  [<user_option> [<user_option> ...]]
  [TAGS ( <tag> [, <tag> ...] )]

<user_option> ::= 
  | FIRSTNAME 'firstname'
  | MIDDLENAME 'middlename'
  | LASTNAME 'lastname'
  | {ACTIVE | INACTIVE}
  | USING PLUGIN pluginname
  | GRANT ADMIN ROLE

<tag> ::= 
  tagname = 'string_value'
```

Описание параметров оператора смотри в CREATE USER.

Оператор RECREATE USER создаёт нового или пересоздаёт существующего пользователя. Если пользователь с таким именем уже существует, то оператор RECREATE TABLE удалить его и создаст нового. Существующие привилегии при этом будут сохранены.

Примеры RECREATE USER

Example 426. Создание или пересоздание пользователя.

```sql
RECREATE USER john PASSWORD 'fYe_3Ksw'
  FIRSTNAME 'John'
  LASTNAME 'Doe'
  INACTIVE;
```

См. также:

CREATE USER, DROP USER.

14.2.6. Получение списка пользователей

Для получения списка пользователей и их атрибутов вы можете воспользоваться виртуальными таблицами SEC$USERS и SEC$USER_ATTRIBUTES.
Example 427. Отображение списка пользователей и их атрибутов

```sql
SELECT
    CAST(U.SECS$USER_NAME AS CHAR(20)) AS LOGIN,
    CAST(A.SECS$KEY AS CHAR(10)) AS TAG,
    CAST(A.SECS$VALUE AS CHAR(20)) AS "VALUE",
    U.SECS$PLUGIN AS "PLUGIN"
FROM SECS$USERS U
LEFT JOIN SECS$USER_ATTRIBUTES A
    ON U.SECS$USER_NAME = A.SECS$USER_NAME
    AND U.SECS$PLUGIN = A.SECS$PLUGIN;
```

<table>
<thead>
<tr>
<th>LOGIN</th>
<th>TAG</th>
<th>VALUE</th>
<th>PLUGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDBA</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>Srp</td>
</tr>
<tr>
<td>ALEX</td>
<td>B</td>
<td>x</td>
<td>Srp</td>
</tr>
<tr>
<td>ALEX</td>
<td>C</td>
<td>sample</td>
<td>Srp</td>
</tr>
<tr>
<td>SYSDBA</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>Legacy_UserManager</td>
</tr>
</tbody>
</table>

Подробнее описание виртуальных таблиц безопасности смотри в: SECS$USERS, SECS$USER_ATTRIBUTES.

14.3. SQL привилегии

Вторым уровнем модели обеспечения безопасности Firebird являются SQL привилегии. После успешного входа в систему (первый уровень), авторизованный пользователь получает доступ к серверу и ко всем базам данных этого сервера, но это не означает, что он имеет доступ к любым объектам в любой базе данных. После создания объекта, только пользователь создавший объект (его владелец) и администраторы имеют доступ к нему. Пользователю необходимы привилегии на каждый объект, к которому он должен получить доступ. Как правило, привилегии должны быть предоставлены явно пользователю владельцем объекта или администратором базы данных.

Привилегия включает тип DML доступа (SELECT, INSERT, UPDATE, DELETE, EXECUTE и REFERENCES), имя объекта базы данных для которого предоставляется привилегия (таблица, представление, процедура или роль) и имя объекта которому предоставляется привилегия (пользователь, процедура, триггер, роль). Существуют различные способы для предоставления нескольких типов доступа на один объект базы данных сразу нескольким пользователям в одном операторе GRANT. Привилегии могут быть отозваны с помощью оператора REVOKE.

Все привилегии по доступу к объектам базы данных хранятся в самой базе, и не могут быть применены к любой другой базе данных.
14.3.1. Владелец объекта базы данных

Пользователь, создавший объект базы данных, становится его владельцем. Только владелец объекта и пользователи с правами администратора в базе данных могут изменить или удалить объект базы данных. Владелец базы данных, то есть пользователь, который создал её, имеет все права на объекты, которые были созданы другими пользователями.

Администраторы или владелец объекта могут выдавать привилегии другим пользователям, в том числе и привилегии на право выдачи привилегий другим пользователям. Собственно сам процесс раздачи и отзыва привилегий на уровне SQL реализуется двумя операторами: GRANT, REVOKE.

14.3.2. Привилегии выполнения SQL кода

Все объекты метаданных содержащие DML или PSQL код могут выполняться в одном из следующих режимов:

• С привилегиями вызывающего пользователя (привилегии CURRENT_USER);
• С привилегиями определяющего пользователя (владельца объекта метаданных).

Исторически сложилось, что все PSQL модули по умолчанию выполняются с привилегиями вызывающего пользователя. Начиная с Firebird 4.0 появилась возможность указывать объектам метаданных с какими привилегиями они будут выполняться: вызывающего или определяющего пользователя. Для этого используется предложение SQL SECURITY, которое можно указать для таблицы, триггера, процедуры, функции или пакета. Если выбрана опция INVOKER, то объект метаданных будет выполняться с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то объект метаданных будет выполняться с привилегиями определяющего пользователя (владельца). Эти привилегии будут дополнены привилегиями выданные самому PSQL модулю с помощью оператора GRANT.

Привилегии выполнения, с которым по умолчанию (не указано у самого модуля) выполняется любой PSQL модуль можно изменить с помощью оператора

ALTER DATABASE SET DEFAULT SQL SECURITY {DEFINER | INVOKER}

Для сохранения обратной совместимости по умолчанию используется опция INVOKER.

Замечания

• Представления (VIEWs) всегда выполняются с привилегиями определяющего пользователя (владельца);
• По умолчанию триггеры наследуют привилегии выполнения которые были указаны у таблицы. Привилегии выполнения могут быть переопределены в самом триггере;
• Процедуры и функции пакета всегда наследуют привилегии выполнения указанный при определении пакета. Привилегии выполнения не могут быть переопределены в самих процедурах и функция пакета;
Анонимные PSQL блоки (EXECUTE BLOCK) всегда выполняются с правами вызывающего пользователя.

В хранимых процедурах, функциях и триггерах вы можете проверить эффективного в настоящий момент пользователя, т.е. пользователя с привилегиями которого выполняется текущий модуль, с помощью системной контекстной переменной EFFECTIVE_USER из пространства имён SYSTEM.

```
select RDB$GET_CONTEXT('SYSTEM', 'EFFECTIVE_USER') from RDB$DATABASE;
```

Один и тот же объект может вызываться в разных контекстах безопасности и требовать различных привилегий. Например, у нас есть:

- хранимая процедура INV с SECURITY INVOKER, которая вставляет записи в таблицу T;
- хранимая процедура DEF с SQL SECURITY DEFINER, которая определена пользователем SYSDBA.

Если пользователь U вызывает процедуру INV, то для доступа к таблице T потребуется привилегия INSERT выданная пользователю U (и конечно привилегия EXECUTE для INV). В этом случае U является эффективным пользователем (EFFECTIVE_USER) во время выполнения INV.

Если пользователь U вызывает процедуру DEF, то для доступа к таблице T потребуется привилегия INSERT выданная пользователю SYSDBA (и EXECUTE на DEF для пользователя U). В этом случае SYSDBA является эффективным пользователем (EFFECTIVE_USER) во время выполнения DEF. Если внутри DEF вызывается процедура INV, то эффективным пользователем по времени выполнения INV будет так же SYSDBA.

Примеры

**Example 428. Создание таблицы с привилегиями определяющего пользователя**

В данном случае пользователю JOE достаточно только привилегии SELECT на таблицу t. Если бы таблица была создана с привилегиями вызывающего пользователя (INVOKER), то ещё потребовалось бы выдать привилегию EXECUTE для таблицы на функцию f.

```
SET TERM ^;
CREATE FUNCTION f() RETURNS INT
AS BEGIN
  RETURN 3;
END^
SET TERM ;^
```
CREATE TABLE t (
  i INTEGER,
  c COMPUTED BY (i + f())
) SQL SECURITY DEFINER;

INSERT INTO t VALUES (2);

GRANT SELECT ON TABLE t TO USER joe;

COMMIT;

CONNECT 'inet://localhost:test' USER joe PASSWORD 'pas';

SELECT * FROM t;

Example 429. Создание процедуры с привилегиями определяющего пользователя

В данном случае пользователю JOE достаточно только привилегии EXECUTE на процедуру p. Если бы процедура была создана с привилегиями вызывающего пользователя (опция INVOKER), то ещё потребовалось бы выдать привилегию INSERT для процедуры p на таблицу t.

CREATE TABLE t (i INTEGER);

SET TERM ^;

CREATE PROCEDURE p (i INTEGER)
SQL SECURITY DEFINER
AS
BEGIN
  INSERT INTO t VALUES (:i);
END^

SET TERM ;^

GRANT EXECUTE ON PROCEDURE p TO USER joe;

COMMIT;

CONNECT 'inet://localhost:test' USER joe PASSWORD 'pas';

EXECUTE PROCEDURE p(1);
例 430. Создание функции с привилегиями определяющего пользователя

В данном случае пользователю JOE достаточно только привилегии EXECUTE на функцию f. Если бы функция была создана с привилегиями вызывающего пользователя (опция INVOKER), то ещё потребовалось бы выдать привилегию SELECT для функции f на таблицу t.

```sql
CREATE TABLE t (i INTEGER PRIMARY KEY, j INTEGER);

INSERT INTO t(i, j) VALUES(1, 2);
INSERT INTO t(i, j) VALUES(2, 5);

COMMIT;

SET TERM ^;

CREATE FUNCTION f (i INTEGER)
SQL SECURITY DEFINER
AS
  DECLARE j INTEGER DEFAULT NULL;
BEGIN
  SELECT j
  FROM t
  WHERE i = :i
  INTO j;

  RETURN COALESCE(j, 0);
END^

SET TERM ;^

GRANT EXECUTE ON FUNCTION f TO USER joe;

COMMIT;

CONNECT 'inet://localhost:test' USER joe PASSWORD 'pas';

SELECT f(1) AS j FROM RDB$DATABASE;
```

例 431. Создание триггера с привилегиями определяющего пользователя

В данном случае пользователю JOE достаточно только привилегии INSERT на таблицу tr. Если бы триггер была создан с привилегиями вызывающего пользователя (опция INVOKER), то ещё потребовалось бы выдать привилегию INSERT для триггера tr_ins на таблицу t.

```sql
CREATE TABLE tr (i INTEGER);
CREATE TABLE t (i INTEGER);
```
SET TERM ^;
CREATE TRIGGER tr_ins FOR tr AFTER INSERT SQL SECURITY DEFINER AS
BEGIN
    INSERT INTO t(i) VALUES(NEW.i);
END^ 
SET TERM ;^ 
GRANT INSERT ON TABLE tr TO USER joe;
COMMIT;

CONNECT 'inet://localhost:test' USER joe PASSWORD 'pas';
INSERT INTO tr(i) VALUES(2);
COMMIT;

Тот же самый результат можно получить указав SQL SECURITY DEFINER для таблицы tr.

CREATE TABLE tr (i INTEGER) SQL SECURITY DEFINER;
CREATE TABLE t (i INTEGER);

SET TERM ^;
CREATE TRIGGER tr_ins FOR tr AFTER INSERT AS
BEGIN
    INSERT INTO t(i) VALUES(NEW.i);
END^ 
SET TERM ;^ 
GRANT INSERT ON TABLE tr TO USER joe;
COMMIT;

CONNECT 'inet://localhost:test' USER joe PASSWORD 'pas';
INSERT INTO tr(i) VALUES(2);
COMMIT;
Example 432. Удаление привилегий выполнения у триггера

Если триггеру явно установлена опция SQL SECURITY, то, для того чтобы наследовать привилегии выполнения у таблицы, необходимо выполнить следующий оператор.

```sql
ALTER TRIGGER tr_ins DROP SQL SECURITY;
```

Example 433. Создание пакета с привилегиями определяющего пользователя

В данном случае пользователю JOE достаточно только привилегии EXECUTE на пакет pk. Если бы пакет была создана с привилегиями вызывающего пользователя (опция INVOKER), то ещё потребовалось бы выдать привилегию INSERT для пакета pk на таблицу t.

```sql
CREATE TABLE t (i INTEGER);

SET TERM ^

CREATE PACKAGE pk
SQL SECURITY DEFINER
AS
BEGIN
  FUNCTION f(i INTEGER) RETURNS INT;
END^

CREATE PACKAGE BODY pk
AS
BEGIN
  FUNCTION f(i INTEGER) RETURNS INT
  AS
  BEGIN
    INSERT INTO t VALUES (:i);
    RETURN i + 1;
  END
END^

SET TERM ;^

GRANT EXECUTE ON PACKAGE pk TO USER joe;

COMMIT;

CONNECT 'inet://localhost:test' USER joe PASSWORD 'pas';

SELECT pk.f(3) FROM rdb$database;
```
### 14.4. Роли

**Роль** (role)—объект базы данных, представляющий набор привилегий. Роли реализуют концепцию управления безопасностью на групповом уровне. Множество привилегий предоставляется роли, а затем роль может быть предоставлена или отозвана у одного или нескольких пользователей.

Пользователь, которому предоставлена роль, должен указать её при входе, для того чтобы получить её привилегии, или же эта роль должна быть гранована с использованием ключевого слова DEFAULT. Любые другие привилегии, предоставленные пользователю, не будут затронуты при его входе в систему с указанной ролью. Вход в систему с несколькими ролами не поддерживается, однако вы можете права нескольких ролей назначенных по умолчанию. Вы можете изменить текущую роль с помощью оператора `SET ROLE`.

Роли могут быть грантованы другим пользователям. При входе с этой ролью пользователь автоматически получит права всех ролей выданных с использованием ключевого слова DEFAULT.

В данном разделе рассматриваются вопросы создания и удаления ролей.

#### 14.4.1. CREATE ROLE

**Назначение**

Создание новой роли.

**Доступно в**

DSQL, ESQL

**Синтаксис**

```sql
CREATE ROLE rolename
[SET SYSTEM PRIVILEGES TO <privileges_list>]

<privileges_list> ::= <privilege> [, <privilege> [, ...]]
```

**Table 258. Параметры оператора CREATE ROLE**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolename</td>
<td>Имя роли. Максимальная длина 63 символа.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>privilege</td>
<td>Системная привилегия.</td>
</tr>
</tbody>
</table>

Оператор `CREATE ROLE` создаёт новую роль. Имя роли должно быть уникальным среди имён ролей.

Желательно также чтобы имя роли было уникальным не только среди имён ролей, но и среди имён пользователей. Если вы создадите роль с тем же именем существующего пользователя, то такой пользователь не сможет подключиться к базе данных.

**Роли с системными привилегиями**

Предложение `SET SYSTEM PRIVILEGES TO` позволяет создать роль с системными привилегиями. Системные привилегии это части привилегий администратора. Таким образом, через делегирование роли с системными привилегиями пользователю можно передавать ему часть прав администратора БД.

Системные привилегии позволяют производить очень тонкую настройку, поэтому иногда вам нужно будет выдать более 1 системной привилегии для выполнения какой-либо задачи. Например, необходимо выдать `IGNORE_DB_TRIGGERS` совместно с `USE_GSTATUtility`, потому что `gstat` должен игнорировать триггера на события БД.

**Доступны следующие системные привилегии:**

**USER_MANAGEMENT**
- Управление пользователями.

**READ_RAW_PAGES**
- Чтение страниц в сыром формате используя `Attachment::getInfo()`

**CREATE_USER_TYPES**
- Создание, изменение и удаление не системных записей в таблице `RDB$USER_TYPES`.

**USE_NBACKUP.Utility**
- Использование `nbackup` для создания резервных копий.

**CHANGE_SHUTDOWN_MODE**
- Закрытие базы данных (shutdown) и возвращение её в online.

**TRACE_ANY_ATTACHMENT**
- Трассировка чужих пользовательских сессий.

**MONITOR_ANY_ATTACHMENT**
- Мониторинг (MON$ таблицы) чужих пользовательских сессий.
**ACCESS_SHUTDOWN_DATABASE**
Доступ к базе данных в режиме shutdown.

**CREATE_DATABASE**
Создание новой базы данных (хранится в базе данных пользователей security.db).

**DROP_DATABASE**
Удаление текущей БД.

**USE_GBAK_UTILITY**
Использование утилиты или сервиса gbak.

**USE_GSTAT_UTILITY**
Использование утилиты или сервиса gstat.

**USE_GFIX_UTILITY**
Использование утилиты или сервиса gfix.

**IGNORE_DB_TRIGGERS**
Разрешает игнорировать триггеры на события БД.

**CHANGE_HEADER_SETTINGS**
Изменение параметров на заголовочной странице БД.

**SELECT_ANY_OBJECT_IN_DATABASE**
Выполнение оператора SELECT из всех селективных объектов (таблиц, представлений, хранимых процедур выбора).

**ACCESS_ANY_OBJECT_IN_DATABASE**
Доступ (любым способом) к любому объекту БД.

**MODIFY_ANY_OBJECT_IN_DATABASE**
Изменение любого объекта БД.

**CHANGE_MAPPING_RULES**
Изменение правил отображения при аутентификации.

**USE_GRANTED_BY_CLAUSE**
Использование GRANTED BY в операторах GRANT и REVOKE.

**GRANT_REVOKE_ON_ANY_OBJECT**
Выполнение операторов GRANT и REVOKE для любого объекта БД.

**GRANT_REVOKE_ANY_DDL_RIGHT**
Выполнение операторов GRANT и REVOKE для выдачи DDL привилегий.

**CREATE_PRIVILEGED_ROLES**
Создание привилегированных ролей (с использованием SET SYSTEM PRIVILEGES).
GET_DBCRYPT_KEY_NAME
Получение имени ключа шифрования.

MODIFY_EXT_CONN_POOL
Управление пулом внешних соединений.

REPLICATE_INTO_DATABASE
Использование API репликации для загрузки наборов изменений в базу данных.

Для проверки имеет ли текущее подключение заданную системную привилегию можно воспользоваться встроенной функцией RDB$SYSTEM_PRIVILEGE().

Кто может создать роль
Выполнить оператор CREATE ROLE могут:

• Администраторы
• Пользователи с привилегией CREATE ROLE.

Примеры CREATE ROLE

Example 435. Создание роли.

```
CREATE ROLE SELLERS;
```

Example 436. Создание роли с системными привилегиями.

```
CREATE ROLE SYS_UTILS
SET SYSTEM PRIVILEGES TO USE_GBAKUTILITY, USE_GSTATUTILITY, IGNORE_DB_TRIGGERS;
```

См. также:
DROP ROLE, GRANT, REVOKE, SET ROLE, RDB$SYSTEM_PRIVILEGE.

14.4.2. ALTER ROLE

Назначение
Изменение системных привилегий роли.

Доступно в
DSQL, ESQL

Синтаксис

```
ALTER ROLE rolename
{ }
```
Синтаксис для роли RDB$ADMIN

```
ALTER ROLE RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING
```

Table 259. Параметры оператора ALTER ROLE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolename</td>
<td>Имя роли.</td>
</tr>
<tr>
<td>privilege</td>
<td>Системная привилегия.</td>
</tr>
</tbody>
</table>

Оператор ALTER ROLE изменяет список системных привилегий роли или удаляет их. При использовании предложения SET SYSTEM PRIVILEGES TO к роли добавляются системные привилегии из списка. Для очистки списка системных привилегий установленных предыдущим оператором используйте оператор ALTER ROLE с предложением DROP SYSTEM PRIVILEGES.

Оператор ALTER ROLE RDB$ADMIN предназначен для включения и отключения возможности администраторам Windows автоматически получать привилегии администраторов при входе.

Эта возможность существует только для одной роли, а именно системной роли RDB$ADMIN, которая существует в любой базе данных с ODS 11.2 и выше. Подробности см. в AUTO ADMIN MAPPING.

В настоящее время является устаревшим и поддерживается для обратной совместимости, вместо него рекомендуется использовать операторы {CREATE | ALTER | DROP} MAPPING.

14.4.3. DROP ROLE

Назначение
Удаление существующей роли.

Доступно в
DSQL, ESQL

Синтаксис

```
DROP ROLE rolename
```

Table 260. Параметры оператора DROP ROLE
Оператор DROP ROLE удаляет существующую роль. При удалении роли все привилегии, предоставленные этой роли, отменяются.

**Кто может удалить роль**

Выполнить оператор DROP ROLE могут:

- Администраторы
- Владелец роли;
- Пользователи с привилегией ALTER ANY ROLE.

**Примеры DROP ROLE**

Example 437. Удаление роли.

```
DROP ROLE SELLERS;
```

См. также: CREATE ROLE, REVOKE.

**14.5. Операторы для предоставления привилегий и назначения ролей**

Оператор GRANT используется для предоставления привилегий и назначения ролей, пользователям и другим объектам базы данных.

**14.5.1. GRANT**

**Назначение**

Предоставление привилегий или назначение ролей.

Доступно в DSQL

**Синтаксис (предоставление привилегий)**

```
GRANT <privileges>
  TO <grantee_list>
  [WITH GRANT OPTION]
  [{GRANTED BY | AS} [USER] grantor]

<privileges> ::= ...
```
<table_privileges> | <execute_privilege> \\
| <usage_privilege> | <ddl_privileges> \\
| <db_ddl_privilege>

table_privileges ::= \\
{ALL [PRIVILEGES] | <table_privilege_list> }  \\
ON [TABLE] {table_name | view_name}

table_privilege_list ::=  \\
<table_privilege> [, <tableprivilege> ...]

table_privilege ::= \\
SELECT | DELETE | INSERT  \\
| UPDATE [(col [, col ...])]  \\
| REFERENCES [(col [, col ...])]

execute_privilege ::= EXECUTE ON {  \\
     PROCEDURE proc_name  \\
   | FUNCTION func_name  \\
   | PACKAGE package_name }

usage_privilege ::= USAGE ON {  \\
      EXCEPTION exception_name  \\
      | {GENERATOR | SEQUENCE} sequence_name }

ddl_privileges ::= {ALL [PRIVILEGES] | <ddl_privilege_list>} <object_type>

ddl_privilege_list ::=  \\
<ddl_privilege> [, <ddl_privilege> ...]

ddl_privilege ::= CREATE | ALTER ANY | DROP ANY

object_type ::= \\
     CHARACTER SET | COLLATION | DOMAIN | EXCEPTION  \\
     | FILTER | FUNCTION | GENERATOR | PACKAGE  \\
     | PROCEDURE | ROLE | SEQUENCE | TABLE | VIEW

db_ddl_privileges ::=  \\
{ALL [PRIVILEGES] | <db_ddl_privilege_list>} {DATABASE | SCHEMA}

db_ddl_privilege_list ::=  \\
<db_ddl_privilege> [, <db_ddl_privilege> ...]

db_ddl_privilege ::= CREATE | ALTER | DROP

grantee_list ::= <grantee> [, <grantee> ...]

grantee ::= \\
     PROCEDURE proc_name | FUNCTION func_name  \\
     | PACKAGE package_name | TRIGGER trig_name
Синтаксис (назначение ролей)

GRANT <role_granted>
TO <role_grantee_list>
[WITH ADMIN OPTION]
[GRANTED BY | AS] [USER] grantor

Table 261. Параметры оператора GRANT

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Имя таблицы, к которой должно быть предоставлена привилегия.</td>
</tr>
<tr>
<td>view_name</td>
<td>Имя представления, к которому должно быть предоставлена привилегия, или которому будут предоставлены привилегии.</td>
</tr>
<tr>
<td>col</td>
<td>Столбец таблицы, к которому должна быть применена привилегия.</td>
</tr>
<tr>
<td>proc_name</td>
<td>Имя хранимой процедуры, для которой должна быть выдана привилегия EXECUTE или которой будут предоставлены привилегии.</td>
</tr>
<tr>
<td>func_name</td>
<td>Имя хранимой функции (или UDF), для которой должна быть выдана привилегия EXECUTE или которой будут даны привилегии.</td>
</tr>
<tr>
<td>package_name</td>
<td>Имя пакета, для которого должна быть выдана привилегия EXECUTE или которому будут даны привилегии.</td>
</tr>
<tr>
<td>exception_name</td>
<td>Имя исключения, для которого должна быть выдана привилегия USAGE.</td>
</tr>
<tr>
<td>generator_name</td>
<td>Имя генератора (последовательности), для которого должна быть выдана привилегия USAGE.</td>
</tr>
<tr>
<td>object_type</td>
<td>Тип объекта метаданных.</td>
</tr>
<tr>
<td>object_list</td>
<td>Список объектов метаданных, которым будут даны привилегии.</td>
</tr>
<tr>
<td>Параметр</td>
<td>Описание</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>trig_name</td>
<td>Имя триггера, которому будут даны привилегии.</td>
</tr>
<tr>
<td>user_list</td>
<td>Список пользователей/ролей, которым будут выданы привилегии.</td>
</tr>
<tr>
<td>username</td>
<td>Имя пользователя, для которого выдаются привилегии или к которому назначается роль.</td>
</tr>
<tr>
<td>rolename</td>
<td>Имя роли.</td>
</tr>
<tr>
<td>Unix_group</td>
<td>Имя группы пользователей в операционных системах семейства UNIX. Только в Firebird Embedded.</td>
</tr>
<tr>
<td>Unix_user</td>
<td>Имя пользователя в операционной системе семейства UNIX. Только в Firebird Embedded.</td>
</tr>
<tr>
<td>sys_privilege</td>
<td>Системная привилегия.</td>
</tr>
<tr>
<td>role_granted</td>
<td>Список ролей, которые будут назначены.</td>
</tr>
<tr>
<td>role_grantee_list</td>
<td>Список пользователей, которым будут назначены роли.</td>
</tr>
<tr>
<td>grantor</td>
<td>Пользователь от имени, которого предоставляются привилегии.</td>
</tr>
</tbody>
</table>

Оператор GRANT предоставляет одну или несколько привилегий для объектов базы данных пользователям, ролям, хранимым процедурам, функциям, пакетам, триггерам и представлениям.

Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие-либо права не будут предоставлены ему явно. При создании объекта только его создатель и SYSDBA имеет привилегии на него и может назначать привилегии другим пользователям, ролям или объектам.

Для различных типов объектов метаданных существует различный набор привилегий. Эти привилегии будут описаны далее отдельно для каждого из типов объектов метаданных.

**Предложение TO**

В предложении TO указывается список пользователей, ролей и объектов базы данных (процедур, функций, пакетов, триггеров и представлений) для которых будут выданы перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить, кому именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным именем, если такой не существует, то привилегии назначаются пользователю. Существование пользователя, которому выдаются права, не проверяется при выполнении оператора GRANT. Если привилегии выдаётся объекту базы данных, то необходимо обязательно указывать тип объекта.

**Рекомендация**

Несмотря на то, что ключевые слова USER и ROLE не обязательные, желательно использовать их, чтобы избежать путаницы.
Пользователь PUBLIC

В SQL существует специальный пользователь PUBLIC, представляющий всех пользователей. Если какая-то операция разрешена пользователю PUBLIC, значит, любой аутентифицированный пользователь может выполнить эту операцию над указанным объектом.

Если привилегии назначены пользователю PUBLIC, то и отозваны они должны быть у пользователя PUBLIC.

Предложение WITH GRANT OPTION

Необязательное предложение WITH Grant OPTION позволяет пользователям, указанным в списке пользователей, передавать другим пользователям привилегии указанные в списке привилегий.

Предложение GRANTED BY

При предоставлении прав в базе данных в качестве лица, предоставившего эти права, обычно записывается текущий пользователь. Используя предложение GRANTED BY можно предоставлять права от имени другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Для облегчения миграции из некоторых других реляционных СУБД нестандартное предложение AS поддерживается как синоним оператора GRANTED BY.

Предложение GRANTED BY может использовать:

- Владелец базы данных;
- SYSDBA;
- Любой пользователь, имеющий права на роль RDB$ADMIN и указавший её при соединении с базой данных;
- При использовании флага AUTO ADMIN MAPPING — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication), даже без указания роли.

Даже владелец роли не может использовать GRANTED BY, если он не находится в вышеупомянутом списке.

Табличные привилегии

Для таблиц и представлений в отличие от других объектов метаданных возможно использовании сразу нескольких привилегий.

Список привилегий для таблиц

SELECT

Разрешает выборку данных (SELECT) из таблицы или представления.
**INSERT**

Разрешает добавлять записи (INSERT) в таблицу или представление.

**UPDATE**

Разрешает изменять записи (UPDATE) в таблице или представлении. Можно указать ограничения, чтобы можно было изменять только указанные столбцы.

**DELETE**

Разрешает удалять записи (DELETE) из таблицы или представления.

**REFERENCES**

Разрешает ссылаться на указанные столбцы внешним ключом. Необходимо указать для столбцов, на которых построен первичный ключ таблицы, если на неё есть ссылка внешним ключом другой таблицы.

**ALL**

Объединяет привилегии SELECT, INSERT, UPDATE, DELETE и REFERENCES.

**Примеры GRANT <privilege> для таблиц**

*Example 438. Пределоставление привилегий для таблиц*

```sql
-- Привилегии SELECT, INSERT пользователю ALEX
GRANT SELECT, INSERT ON TABLE SALES TO USER ALEX;

-- Привилегия SELECT ролям MANAGER, ENGINEER и пользователю IVAN
GRANT SELECT ON TABLE CUSTOMER TO ROLE MANAGER, ROLE ENGINEER, USER IVAN;

-- Все привилегии для роли ADMINISTRATOR
-- с возможностью передачи своих полномочий
GRANT ALL ON TABLE CUSTOMER TO ROLE ADMINISTRATOR WITH GRANT OPTION;

-- Привилегии SELECT и REFERENCE для столбца NAME для всех пользователей
GRANT SELECT, REFERENCES (NAME) ON TABLE COUNTRY TO PUBLIC;

-- Выдача привилегии SELECT для пользователя IVAN от имени пользователя ALEX
GRANT SELECT ON TABLE EMPLOYEE TO USER IVAN GRANTED BY ALEX;

-- Привилегия UPDATE для столбцов FIRST_NAME, LAST_NAME
GRANT UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE TO USER IVAN;

-- Привилегия INSERT для хранимой процедуры ADD_EMP_PROJ
GRANT INSERT ON EMPLOYEE_PROJECT
```
**Привилегия EXECUTE**

Привилегия EXECUTE (выполнение) применима к хранимым процедурам, хранимым функциям, пакетам и унаследованным внешним функциям (UDF), определяемых как DECLARE EXTERNAL FUNCTION.

Для хранимых процедур привилегия EXECUTE позволяет не только выполнять хранимые процедуры, но и делать выборку данных из селективных процедур (с помощью оператора SELECT).

Привилегия может быть назначена только для всего пакета, а не для отдельных его подпрограмм.

**Примеры предоставления привилегии EXECUTE**

*Example 439. Предоставление привилегии EXECUTE*

```
-- Привилегия EXECUTE на хранимую процедуру
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ
TO ROLE MANAGER;

-- Привилегия EXECUTE на хранимую функцию
GRANT EXECUTE ON FUNCTION GET_BEGIN_DATE TO ROLE MANAGER;

-- Привилегия EXECUTE на пакет
GRANT EXECUTE ON PACKAGE APP_VAR TO PUBLIC;

-- Привилегия EXECUTE на функцию выданная пакету
GRANT EXECUTE ON FUNCTION GET_BEGIN_DATE
TO PACKAGE APP_VAR;
```

**Привилегия USAGE**

Для использования объектов метаданных, отличных от таблиц, представлений, хранимых процедур и функций, триггеров и пакетов, в пользовательских запросах необходимо предоставить пользователю привилегию USAGE для этих объектов. Поскольку в Firebird хранимые процедуры и функции, триггеры и подпрограммы пакетов выполняются с привилегиями вызывающего пользователя, то при использовании таких объектов метаданных в них, может потребоваться назначить привилегию USAGE и для них.

В Firebird 3 привилегия USAGE проверяется только для исключений (exception) и генераторов/последовательностей (в gen_id(gen_name, n) или next value for gen_name). Привилегии для других объектов метаданных могут быть включены в следующих релизах, если покажется целесообразным.
Привилегия USAGE даёт права только на приращения генераторов (последовательностей) с помощью функции GEN_ID или конструкции NEXT VALUE FOR. Оператор SET GENERATOR является аналогом оператора ALTER SEQUENCE … RESTART WITH, которые относятся к DDL операторам. По умолчанию права на такие операции имеет только владелец генератора (последовательности). Права на установку начального значения любого генератора (последовательности) можно предоставить с помощью GRANT ALTER ANY SEQUENCE, что не рекомендуется для обычных пользователей.

Примеры предоставления привилегии USAGE

*Example 440. Предоставление привилегии USAGE*

```sql
-- Привилегия USAGE на последовательность выданная роли  
GRANT USAGE ON SEQUENCE GEN_AGE TO ROLE MANAGER;

-- Привилегия USAGE на последовательность выданная триггеру  
GRANT USAGE ON SEQUENCE GEN_AGE TO TRIGGER TR_AGE_BI;

-- Привилегия USAGE на исключение выданная пакету  
GRANT USAGE ON EXCEPTION E_ACCESS_DENIED  
TO PACKAGE PKG_BILL;
```

**DDL привилегии**

По умолчанию создавать новые объекты метаданных могут только Администраторы, а изменять и удалять — администраторы и владельцы этих объектов. Выдача привилегий на создание, изменение или удаление объектов конкретного типа позволяет расширить этот список.

**Список DDL привилегий**

**CREATE**  
Разрешает создание объекта указанного типа метаданных.

**ALTER ANY**  
Разрешает изменение любого объекта указанного типа метаданных.

**DROP ANY**  
Разрешает удаление любого объекта указанного типа метаданных.

**ALL**  
Объединяет привилегии CREATE, ALTER и DROP на указанный тип объекта.

Метаданные триггеров и индексов наследуют привилегии таблиц, которые владеют ими.
Примеры предоставления DDL привилегий

**Example 441. Предоставление привилегий на изменение метаданных**

```
-- Разрешение пользователю Joe создавать таблицы
GRANT CREATE TABLE TO Joe;

-- Разрешение пользователю Joe изменять любые процедуры
GRANT ALTER ANY PROCEDURE TO Joe;
```

**DDL привилегии для базы данных**

Оператор назначения привилегий на создание, удаление и изменение базы данных имеет несколько отличную форму от оператора назначения DDL привилегий на другие объекты метаданных.

**Список DDL привилегий на базу данных**

**CREATE**

- Разрешает создание базы данных.

**ALTER**

- Разрешает изменение текущей базы данных.

**DROP**

- Разрешает удаление текущей базы данных.

**ALL**

- Объединяет привилегии ALTER и DROP на базу данных.

Привилегия `CREATE DATABASE` является особым видом привилегий, поскольку она сохраняется в базе данных безопасности. Список пользователей имеющих привилегию `CREATE DATABASE` можно посмотреть в виртуальной таблице SEC$DB_CREATORS. Привилегию на создание новой базы данных могут выдавать только администраторы в базе данных безопасности.

Привилегии `ALTER DATABASE` и `DROP DATABASE` относятся только к текущей базе данных, тогда как DDL привилегии `ALTER ANY` и `DROP ANY` на другие объекты метаданных относятся ко всем объектам указанного типа внутри текущей базы данных. Привилегии на изменение и удаление текущей базы данных могут выдаваться только администраторы.

**Примеры предоставления DDL привилегий на базу данных**

**Example 442. Разрешение пользователю Superuser создавать базы данных**

```
GRANT CREATE DATABASE TO USER Superuser;
```
Example 443. Разрешение пользователю Joe выполнять оператор ALTER DATABASE для текущей базы данных

```
GRANT ALTER DATABASE TO USER Joe;
```

Example 444. Разрешение пользователю Fedor удалять текущую базу данных

```
GRANT DROP DATABASE TO USER Fedor;
```

Предоставление прав системным привилегиям

Благодаря поддержке системных привилегий в ядре, становится очень удобно предоставлять некоторые дополнительные привилегии пользователям уже имеющим какую-то системную привилегию. Для этих целей существует возможность использовать в качестве грантополучателя одну или несколько системных привилегий.

Примеры предоставления прав системным привилегиям

Следующий оператор назначит все привилегии на представление PLG$SRP_VIEW, используемое в плагине управления пользователями SRP, системной привилегии USER_MANAGEMENT.

```
GRANT ALL ON PLG$SRP_VIEW TO SYSTEM PRIVILEGE USER_MANAGEMENT
```

Описание системных привилегий вы можете посмотреть в CREATE ROLE

Назначение ролей

Синтаксис (выдача ролей)

```
GRANT <role_granted>
   TO <role_grantee_list>
   [WITH ADMIN OPTION]
   [{GRANTED BY | AS} [USER] grantor]

<role_granted> ::= [DEFAULT] role_name [, [DEFAULT] role_name ...]

<role_grantee_list> ::= <role_grantee> [, <role_grantee> ...]

<role_grantee> ::= [USER] username
                 | [ROLE] grantee_role_name
```
Оператор GRANT может быть использован для назначения ролей для списка пользователей или ролей. В этом случае после предложения GRANT следует список ролей, которые будут назначены списку пользователей или ролей, указанному после предложения TO.

**Ключевое слово DEFAULT**

Если используется ключевое слово DEFAULT, то роль (роли) будет использоваться пользователем или ролью каждый раз, даже если она не была указана явно. При подключении пользователь получит привилегии всех ролей, которые были назначены пользователю с использованием ключевого слова DEFAULT. Если пользователь укажет свою роль при подключении, то получит привилегии этой роли (если она была ему назначена) и привилегии всех ролей назначенных ему с использованием ключевого слова DEFAULT.

**Предложение WITH ADMIN OPTION**

Необязательное предложение WITH ADMIN OPTION позволяет пользователям, указанным в списке пользователей, передавать свои роли другому пользователю или роли. Полномочия роли могут быть переданы кумулятивно, только если каждая роль в последовательности ролей назначена с использованием WITH ADMIN OPTION.

**Примеры назначения ролей**

*Example 445. Назначение ролей для пользователей*

```sql
-- Назначение ролей DIRECTOR и MANAGER пользователю IVAN
GRANT DIRECTOR, MANAGER TO USER IVAN;

-- Назначение роли ADMIN пользователю ALEX
-- с возможностью назначить эту другим пользователям
GRANT MANAGER TO USER ALEX WITH ADMIN OPTION;
```

*Example 446. Назначение ролей для пользователей с ключевым словом DEFAULT*

```sql
-- Назначение роли MANAGER пользователю JOHN
-- Привилегии роли будут автоматически назначаться пользователю
-- каждый раз при входе. В этом случае роль выступает в качестве группы.
GRANT DEFAULT MANAGER TO USER JOHN;

-- Теперь при входе пользователь JOHN автоматически получит привилегии
-- ролей MANAGER (см. предыдущий оператор) и DIRECTOR
GRANT DEFAULT DIRECTOR TO USER JOHN;
```

*Example 447. Назначение ролей другим ролям*

```sql
-- Назначение роли MANAGER для роли DIRECTOR
-- с возможностью передачи роли MANAGER другим пользователям или ролям
```
**14.6. Операторы для отзыва привилегий и ролей**

Оператор **REVOKE** используется для отзыва привилегий и ролей, у пользователей и других объектов базы данных.

**14.6.1. REVOKE**

**Назначение**

Отмена привилегий или отбор ролей.

**Доступно в**

DSQL

**Синтаксис (отзыв привилегий)**

```sql
REVOKE [GRANT OPTION FOR] <privileges>
   FROM <grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

- `<privileges>` ::=
  - См. **GRANT** синтаксис

- `<grantee_list>` ::=
  - См. **GRANT** синтаксис

**Синтаксис (отзыв ролей)**

```sql
REVOKE [ADMIN OPTION FOR] <role_granted>
   FROM <role_grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

---

`Grant Manager TO Role Director WITH Admin Option;`

-- Назначение роли ACCOUNTANT роли DIRECTOR
-- при входе в систему с ролью DIRECTOR полномочия роли ACCOUNTANT
-- будут также получены

`Grant Default Accountant TO Role Director;`

-- Пользователь PETROV при входе автоматически получает
-- полномочия роли DIRECTOR. Эти полномочия будут включать также
-- полномочия роли ACCOUNTANT. Для получения полномочий роли MANAGER
-- необходимо указать эту роль при входе в систему или позже с
-- помощью оператора SET ROLE

`Grant Default Role Director TO User Petrov;`

**См. также:**

**REVOKE**

**См. также:**

**14.6. Операторы для отзыва привилегий и ролей**

Оператор **REVOKE** используется для отзыва привилегий и ролей, у пользователей и других объектов базы данных.

**14.6.1. REVOKE**

**Назначение**

Отмена привилегий или отбор ролей.

**Доступно в**

DSQL

**Синтаксис (отзыв привилегий)**

```sql
REVOKE [GRANT OPTION FOR] <privileges>
   FROM <grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

- `<privileges>` ::=
  - См. **GRANT** синтаксис

- `<grantee_list>` ::=
  - См. **GRANT** синтаксис

**Синтаксис (отзыв ролей)**

```sql
REVOKE [ADMIN OPTION FOR] <role_granted>
   FROM <role_grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

---

`Grant Manager TO Role Director WITH Admin Option;`

-- Назначение роли ACCOUNTANT роли DIRECTOR
-- при входе в систему с ролью DIRECTOR полномочия роли ACCOUNTANT
-- будут также получены

`Grant Default Accountant TO Role Director;`

-- Пользователь PETROV при входе автоматически получает
-- полномочия роли DIRECTOR. Эти полномочия будут включать также
-- полномочия роли ACCOUNTANT. Для получения полномочий роли MANAGER
-- необходимо указать эту роль при входе в систему или позже с
-- помощью оператора SET ROLE

`Grant Default Role Director TO User Petrov;`

**См. также:**

**REVOKE**

**См. также:**

**14.6. Операторы для отзыва привилегий и ролей**

Оператор **REVOKE** используется для отзыва привилегий и ролей, у пользователей и других объектов базы данных.

**14.6.1. REVOKE**

**Назначение**

Отмена привилегий или отбор ролей.

**Доступно в**

DSQL

**Синтаксис (отзыв привилегий)**

```sql
REVOKE [GRANT OPTION FOR] <privileges>
   FROM <grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

- `<privileges>` ::=
  - См. **GRANT** синтаксис

- `<grantee_list>` ::=
  - См. **GRANT** синтаксис

**Синтаксис (отзыв ролей)**

```sql
REVOKE [ADMIN OPTION FOR] <role_granted>
   FROM <role_grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

---

`Grant Manager TO Role Director WITH Admin Option;`

-- Назначение роли ACCOUNTANT роли DIRECTOR
-- при входе в систему с ролью DIRECTOR полномочия роли ACCOUNTANT
-- будут также получены

`Grant Default Accountant TO Role Director;`

-- Пользователь PETROV при входе автоматически получает
-- полномочия роли DIRECTOR. Эти полномочия будут включать также
-- полномочия роли ACCOUNTANT. Для получения полномочий роли MANAGER
-- необходимо указать эту роль при входе в систему или позже с
-- помощью оператора SET ROLE

`Grant Default Role Director TO User Petrov;`

**См. также:**

**REVOKE**

**См. также:**

**14.6. Операторы для отзыва привилегий и ролей**

Оператор **REVOKE** используется для отзыва привилегий и ролей, у пользователей и других объектов базы данных.

**14.6.1. REVOKE**

**Назначение**

Отмена привилегий или отбор ролей.

**Доступно в**

DSQL

**Синтаксис (отзыв привилегий)**

```sql
REVOKE [GRANT OPTION FOR] <privileges>
   FROM <grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

- `<privileges>` ::=
  - См. **GRANT** синтаксис

- `<grantee_list>` ::=
  - См. **GRANT** синтаксис

**Синтаксис (отзыв ролей)**

```sql
REVOKE [ADMIN OPTION FOR] <role_granted>
   FROM <role_grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

---

`Grant Manager TO Role Director WITH Admin Option;`

-- Назначение роли ACCOUNTANT роли DIRECTOR
-- при входе в систему с ролью DIRECTOR полномочия роли ACCOUNTANT
-- будут также получены

`Grant Default Accountant TO Role Director;`

-- Пользователь PETROV при входе автоматически получает
-- полномочия роли DIRECTOR. Эти полномочия будут включать также
-- полномочия роли ACCOUNTANT. Для получения полномочий роли MANAGER
-- необходимо указать эту роль при входе в систему или позже с
-- помощью оператора SET ROLE

`Grant Default Role Director TO User Petrov;`

**См. также:**

**REVOKE**

**См. также:**

**14.6. Операторы для отзыва привилегий и ролей**

Оператор **REVOKE** используется для отзыва привилегий и ролей, у пользователей и других объектов базы данных.

**14.6.1. REVOKE**

**Назначение**

Отмена привилегий или отбор ролей.

**Доступно в**

DSQL

**Синтаксис (отзыв привилегий)**

```sql
REVOKE [GRANT OPTION FOR] <privileges>
   FROM <grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```

- `<privileges>` ::=
  - См. **GRANT** синтаксис

- `<grantee_list>` ::=
  - См. **GRANT** синтаксис

**Синтаксис (отзыв ролей)**

```sql
REVOKE [ADMIN OPTION FOR] <role_granted>
   FROM <role_grantee_list>
   [{GRANTED BY | AS} [USER] grantor]
```
<role_granted> ::=  
См. GRANT синтаксис

<role_grantee_list> ::=  
См. GRANT синтаксис

Синтаксис (отзыв всех привилегий)

REVOKE ALL ON ALL FROM <grantee_list>

<grantee_list> ::=  
GRANT синтаксис

Table 262. Параметры оператора REVOKE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>Пользователь от имени, которого отзываются привилегии.</td>
</tr>
</tbody>
</table>

Оператор REVOKE отменяет привилегии для пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений выданные оператором GRANT. Подробное описание различных типов привилегий см. в GRANT.

Предложение FROM

В предложении FROM указывается список пользователей, ролей и объектов базы данных (процедур, функций, пакетов, триггеров и представлений) у которых будут отняты перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить, у кого именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным именем, если такой не существует, то привилегии отбираются у пользователя.

Рекомендация

Несмотря на то, что ключевые слова USER и ROLE не обязательные, желательно использовать их, чтобы избежать путаницы.

Существование пользователя, у которого отбираются права, не проверяется при выполнении оператора REVOKE. Если привилегия отбирается у объекта базы данных, то необходимо обязательно указывать тип объекта.

Если привилегии были назначены специальному пользователю PUBLIC, то отменять привилегии необходимо для пользователя PUBLIC. Специальный пользователь PUBLIC используется, когда необходимо предоставить привилегии сразу всем пользователям. Однако не следует рассматривать PUBLIC как группу пользователей.

Примеры отзыва привилегий
Example 448. Отзыв привилегий на таблицу

-- отзыв привилегий SELECT, INSERT у таблицы
REVOKE SELECT, INSERT ON TABLE SALES FROM USER ALEX

-- отзыв привилегии SELECT у ролей MANAGER и ENGINEER и пользователя IVAN
REVOKE SELECT ON TABLE CUSTOMER
FROM ROLE MANAGER, ROLE ENGINEER, USER IVAN;

-- отзыв привилегий SELECT и REFERENCES у пользователя PUBLIC
REVOKE SELECT, REFERENCES (NAME) ON TABLE COUNTRY
FROM PUBLIC;

-- отзыв привилегии UPDATE для столбцов FIRST_NAME, LAST_NAME
REVOKE UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE
FROM USER IVAN;

-- отзыв привилегии INSERT у хранимой процедуры ADD_EMP_PROJ
REVOKE INSERT ON EMPLOYEE_PROJECT
FROM PROCEDURE ADD_EMP_PROJ;

Example 449. Отзыв привилегии EXECUTE

-- отзыв привилегии EXECUTE на процедуру
REVOKE EXECUTE ON PROCEDURE ADD_EMP_PROJ
FROM USER IVAN;

-- отзыв привилегии EXECUTE на пакет
REVOKE EXECUTE ON PACKAGE DATE_UTILS
FROM USER ALEX;

Example 450. Отзыв привилегии USAGE

-- Отзыв привилегии USAGE на последовательность выданной роли
REVOKE USAGE ON SEQUENCE GEN_AGE FROM ROLE MANAGER;

-- Отзыв привилегии USAGE на последовательность выданной триггеру
REVOKE USAGE ON SEQUENCE GEN_AGE FROM TRIGGER TR_AGE_BI;

-- Отзыв привилегии USAGE на исключение выданной пакету
REVOKE USAGE ON EXCEPTION E_ACCESS_DENIED
FROM PACKAGE PKG_BILL;
Пример 451. Отзыв привилегий на изменение метаданных

```
-- Отзыв у пользователя Joe привилегии на создание таблиц
REVOKE CREATE TABLE FROM Joe;

-- Отзыв у пользователя Joe привилегии на изменение любой процедуры
REVOKE ALTER ANY FUNCTION FROM Joe;

-- Отзыв привилегии пользователю на создание базы данных
-- у пользователя Superuser
REVOKE CREATE DATABASE FROM USER Superuser;
```

Пример 452. Отзыв привилегий у системной привилегии

```
-- Отзыв у системной привилегии USER_MANAGEMENT всех прав
-- на представление PLG$SRP_VIEW
REVOKE ALL ON PLG$SRP_VIEW FROM SYSTEM PRIVILEGE USER_MANAGEMENT;
```

Предложение GRANT OPTION FOR

Необязательное предложение GRANT OPTION FOR отмечает для соответствующего пользователя или роли право предоставления другим пользователям или ролям привилегии к таблицам, представлениям, триггерам, хранимым процедурам.

Отзыв привилегий с использованием GRANT OPTION FOR

Пример 453. Отзыв привилегий с использованием GRANT OPTION FOR

```
-- отмена возможности передавать любую из привилегии на таблицу
-- другим пользователям или ролям у роли ADMINISTRATOR
REVOKE GRANT OPTION FOR ALL ON TABLE CUSTOMER
FROM ROLE ADMINISTRATOR;

-- отзыв привилегии EXECUTE на функцию
-- и лишение права передавать эту привилегию
-- другим пользователям и ролям
REVOKE GRANT OPTION FOR EXECUTE ON FUNCTION GET_BEGIN_DATE
FROM ROLE MANAGER;
```

Отмена назначенных ролей

Другое назначение оператора REVOKE в отзыве назначенных пользователям или ролям ролей оператором GRANT. В этом случае после предложения REVOKE следует список ролей, которые...
будут отозваны у списка пользователей или ролей, указанных после предложения FROM.

В одном операторе могут быть обработаны несколько ролей и/или грантополучателей.

**Предложение ADMIN OPTION FOR**

Необязательное предложение ADMIN OPTION FOR отменяет ранее предоставленную административную опцию (право на передачу предоставленной пользователю роли другим) из грантополучателей, не отменяя прав на роль.

**Примеры отзыва ролей**

*Example 454. Отзыв ролей*

```sql
-- Отзыв ролей DIRECTOR, MANAGER у пользователя IVAN
REVOKE DIRECTOR, MANAGER FROM USER IVAN;

-- Отзыв умолчательной роли MANAGER у пользователя FEDOR
REVOKE DEFAULT MANAGER FROM USER FEDOR;

-- Отзыв роли MANAGER и права назначать её другим пользователям
REVOKE ADMIN OPTION FOR MANAGER FROM USER ALEX;
```

**Предложение GRANTED BY**

При предоставлении прав в базе данных в качестве лица, предоставившего эти права, обычно записывается текущий пользователь. Используя предложение GRANTED BY можно предоставить права от имени другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Для облегчения миграции из некоторых других реляционных СУБД нестандартное предложение AS поддерживается как синоним оператора GRANTED BY.

Предложение GRANTED BY может использовать:

- Владелец базы данных;
- SYSDBA;
- Любой пользователь, имеющий права на роль RDB$ADMIN и указавший её при соединении с базой данных;
- При использовании флага AUTO ADMIN MAPPING — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication), даже без указания роли.

Даже владелец роли не может использовать GRANTED BY, если он не находится в вышеупомянутом списке.
Отзыв привилегий с использованием GRANTED BY

Example 455. Отзыв привилегий на таблицу с использованием GRANTED BY

```sql
-- отмена привилегии SELECT у пользователя IVAN,
-- которая была выдана пользователем ALEX
REVOKE SELECT ON TABLE EMPLOYEE
FROM USER IVAN GRANTED BY ALEX;
```

**REVOKE ALL ON ALL**

Если после ключевого слова REVOKE указано предложение ALL ON ALL, то это позволяет отменить все привилегии (включая роли) на всех объектах от одного или более пользователей и/или ролей. Это быстрый способ “очистить” (отобрать) права, когда пользователю должен быть заблокирован доступ к базе данных.

- Когда оператор REVOKE ALL ON ALL вызывается привилегированным пользователем (владельцем базы данных, SYSDBA или любым пользователем, у которого CURRENT_ROLE — RDB$ADMIN), удаляются все права независимо от того, кто их предоставил. В противном случае удаляются только права, предоставленные текущим пользователем;
- Не поддерживается предложение GRANTED BY;
- Этот оператор не удаляет флаг пользователя, давшего права на хранимые процедуры, триггеры или представлений (права на такие объекты конечно удаляются).

Example 456. Отзыв всех привилегий и ролей у пользователя

```sql
REVOKE ALL ON ALL FROM IVAN;
```

После выполнения этой команды у пользователя IVAN нет вообще никаких прав.

*См. также:*

**GRANT.**

**14.7. Отображение объектов безопасности**

С введением поддержки множества баз данных безопасности в Firebird появились новые проблемы, которые не могли произойти с единой глобальной базой данных безопасности. Кластеры баз данных, использующие одну и ту же базу данных безопасности, были эффективно разделены. Отображения предоставляют средства для достижения той же эффективности, когда существует множество баз данных, использующих свои собственные базу данных безопасности. В некоторых случаях требуется управление для ограничения взаимодействия между такими кластерами. Например:
• когда EXECUTE STATEMENT ON EXTERNAL DATA SOURCE требует обмена данными между кластерами;
• когда общесерверный SYSDBA доступ к базам данных необходим от других кластеров, использующих службы;
• аналогичные проблемы существовали в Firebird 2.1 и 2.5 под Windows, из-за поддержки доверительной аутентификации: существовало два отдельных списка пользователей — один в базе данных безопасности, а другой в Windows, в тех случаях когда было необходимо связать их. Примером может служить получение роли, предоставленной группе Windows, автоматически назначаемой членам этой группы.

Единым решением для всех этих случаев является создание правил отображения информации о пользователе, входящего в систему, на внутренние объекты безопасности — CURRENT_USER и CURRENT_ROLE.

В Firebird имеется одно встроенное глобальное правило, действующее по умолчанию: пользователи прошедшие проверку в базе данных безопасности всегда отображаются в любую базу данных один к одному. Это безопасное правило: для базы данных безопасности не имеет смысла не доверять себе.

14.7.1. CREATE MAPPING

Назначение
Создание отображения объекта безопасности.

Доступно в DSQL

Синтаксис

```
CREATE [GLOBAL] MAPPING name
USING {
    PLUGIN plugin_name [IN database]
    | ANY PLUGIN [IN database | SERVERWIDE]
    | MAPPING [IN database]
    | '*' [IN database] }
FROM { ANY type | type from_name }
TO { USER | ROLE } [to_name]
```

Table 263. Параметры оператора CREATE MAPPING

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Имя отображения. Может содержать до 63 символов.</td>
</tr>
<tr>
<td>plugin_name</td>
<td>Имя плагина аутентификации.</td>
</tr>
<tr>
<td>database</td>
<td>Имя базы данных, в которой прошла аутентификация.</td>
</tr>
<tr>
<td>type</td>
<td>Тип объекта, который будет отображён.</td>
</tr>
</tbody>
</table>
Оператор `CREATE MAPPING` создаёт отображение объектов безопасности (пользователей, групп, ролей) одного или нескольких плагинов аутентификации на внутренние объекты безопасности – `CURRENT_USER` и `CURRENT_ROLE`. Имя отображения должно быть уникальным среди имён отображений.

Если присутствует опция `GLOBAL`, то отображение будет применено не только для текущей базы данных, но и для всех баз данных находящихся в том же кластере, в том числе и базы данных безопасности.

Если существуют одноименные глобальные и локальные отображения, то вам следует знать, что это разные объекты.

Глобальное отображение работает, если в качестве базы данных безопасности используется база данных Firebird 3 или более высокой версии. Если вы планируете использовать другую базу данных, например, для целей использования собственного поставщика, то вам необходимо создать таблицу в ней и назвать её `RDB$MAP` с той же структурой, что и `RDB$MAP` в базе данных Firebird 3 и дать доступ на запись только для SYSDBA.

Предложение `USING` описывает источник отображения. Оно имеет весьма сложный набор опций:

- явное указание имени плагина (опция `PLUGIN plugin_name`) означает, что оно будет работать только с этим плагином;
- оно может использовать любой доступный плагин (опция `ANY PLUGIN`), даже если источник является продуктом предыдущего отображения;
- оно может быть сделано так, чтобы работать только с обще серверными плагинами (опция `SERVERWIDE`);
- оно может быть сделано так, чтобы работать только с результатами предыдущего отображения (опция `MAPPING`);
- вы можете опустить использование любого из методов, используя звёздочку (*) в качестве аргумента;
- оно может содержать имя базы данных (опция `IN`), из которой происходит отображение объекта `FROM`.

Этот аргумент не является допустимым для отображения с общесерверной аутентификацией.

Предложение `FROM` описывает отображаемый объект. Оно принимает обязательный аргумент — тип объекта.
Особенности:

- при отображении имён из плагинов, тип определяется плагином;
- при отображении продукта предыдущего отображения, типом может быть только USER и ROLE;
- если имя объекта будет указано явно, то оно будет учитываться при отображении;
- при использовании ключевого слова ANY будут отображены объекты с любыми именами данного типа.

Сочетание источник (предложение USING) и объект отображения (предложение FROM) должно быть уникальным, иначе будет сгенерирована ошибка. Это позволяет только если одно отображение является глобальным, а второе локальным.

В предложении TO указывается пользователь или роль, на которого будет произведено отображение. to_name является не обязательным аргументом. Если он не указан, то в качестве имени объекта будет использовано оригинальное имя из отображаемого объекта.

Локальное отображение перекрывает глобальное отображение с одинаковым сочетанием источника и объекта отображения. Это действует приблизительно так же как с настройками: настройки уровня базы данных (databases.conf) перекрывают глобальные настройки (firebird.conf).

Кто может создать отображение

- SYSDBA;
- Владелец базы данных (если отображение локальное);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux.

Примеры CREATE MAPPING

**Example 457.** Включение использования доверительной аутентификации Windows во всех базах данных, которые используют текущую базу данных безопасности.

```sql
CREATE GLOBAL MAPPING TRUSTED_AUTH
USING PLUGIN WIN_SSPI
FROM ANY USER
TO USER;
```

**Example 458.** Включение SYSDBA-подобного доступа для администраторов Windows в текущей базе данных.

```sql
CREATE MAPPING WIN_ADMIN
USING PLUGIN WIN_SSPI
```
GROUP Domain_Any_RID_Admins не существует в Windows, но такое имя будет добавлено плагином win_sspi для обеспечения точной обратной совместимости.

**Example 459. Включение доступа определённому пользователю из другой базы данных к текущей базе данных под другим именем.**

```sql
CREATE MAPPING FROM RT
USING PLUGIN SRP IN "rt"
FROM USER U1 TO USER U2;
```

Пользователь U1 прошедший аутентификацию в базе данных rt будет отображён на пользователя с именем U2.

**Example 460. Включение общесерверного SYSDBA (от основной базы данных безопасности) для доступа к текущей базе данных.**

Предположим, что база данных использует базу данных безопасности не по умолчанию.

```sql
CREATE MAPPING DEF_SYSDBA
USING PLUGIN SRP IN "security.db"
FROM USER SYSDBA TO USER;
```

**Example 461. Создание ограничения прав для пользователей, которые подключаются унаследованным плагином аутентификации.**

```sql
CREATE MAPPING LEGACY_2_GUEST
USING PLUGIN legacy_auth
FROM ANY USER TO USER GUEST;
```

См. также:
**14.7.2. ALTER MAPPING**

**Назначение**
Изменение отображения объекта безопасности.

**Доступно в**
DSQL.

**Синтаксис**

```
ALTER [GLOBAL] MAPPING name
  USING {
    PLUGIN plugin_name [IN database]
    | ANY PLUGIN [IN database | SERVERWIDE]
    | MAPPING [IN database]
    | '*' [IN database] }
  FROM { ANY type | type from_name }
  TO { USER | ROLE } [to_name]
```

Описание параметров оператора смотри в **CREATE MAPPING**.

Оператор ALTER MAPPING позволяет изменять любые опции существующего отображения.

⚠️ Одноименные глобальное и локальное отображение — это разные объекты.

**Кто может изменить отображение**

*Выполнить ALTER MAPPING могут:*

- SYSDBA;
- Владелец базы данных (если отображение локальное);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux.

**Примеры ALTER MAPPING**

*Example 462. Изменение отображения.*

```sql
ALTER MAPPING FROM_RT
  USING PLUGIN SRP IN "rt"
  FROM USER U1 TO USER U3;
```

*См. также:*
CREATE MAPPING, CREATE OR ALTER MAPPING, DROP MAPPING.
14.7.3. **CREATE OR ALTER MAPPING**

**Назначение**

Создание или изменение отображения объекта безопасности.

**Доступно в**

DSQL

**Синтаксис**

```sql
CREATE OR ALTER [GLOBAL] MAPPING name
    USING {
        PLUGIN plugin_name [IN database] 
        | ANY PLUGIN [IN database | SERVERWIDE] 
        | MAPPING [IN database] 
        | '*' [IN database] 
    } 
    FROM { ANY type | type from_name } 
    TO { USER | ROLE } [to_name]
```

Описание параметров оператора смотри в **CREATE MAPPING**.

Оператор CREATE OR ALTER MAPPING создаёт новое или изменяет существующее отображение. Если отображение не существует, то оно будет создано с использованием оператора CREATE MAPPING.

⚠️ Одноименные глобальное и локальное отображение — это разные объекты.

**Примеры CREATE OR ALTER MAPPING**

*Example 463. Создание нового или изменение существующего отображения.*

```sql
CREATE OR ALTER MAPPING FROM_RT
USING PLUGIN SRP IN "rt"
FROM USER U1 TO USER U4;
```

*См. также:*

CREATE MAPPING, ALTER MAPPING, DROP MAPPING.

14.7.4. **DROP MAPPING**

**Назначение**

Удаление отображения объекта безопасности.

**Доступно в**

DSQL
Синтаксис

```
DROP [GLOBAL] MAPPING name
```

### Table 264. Параметры оператора DROP MAPPING

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Имя отображения.</td>
</tr>
</tbody>
</table>

Оператор `DROP MAPPING` удаляет существующее отображение. Если указана опция `GLOBAL`, то будет удалено глобальное отображение.

⚠️ Одноименные глобальное и локальное отображение — это разные объекты.

**Кто может удалить отображение**

Выполнить `DROP MAPPING` могут:

- SYSDBA;
- Владелец базы данных (если отображение локальное);
- Любой пользователь, вошедший с ролью `RDB$ADMIN`;
- Пользователь root операционной системы Linux.

**Примеры DROP MAPPING**

**Example 464. Удаление отображения.**

```
DROP MAPPING FROM_RT;
```

**См. также:**

`CREATE MAPPING`.

## 14.8. Шифрование базы данных

В Firebird существует возможность зашифровать данные хранимые в базе данных. Не весь файл базы данных шифруется: только страницы данных, индексов и blob.

Для того чтобы сделать шифрование базы данных возможным, необходимо получить или написать плагин шифрования базы данных.

⚠️ Пример плагина шифрования в `examples/dbcrypt` не производит реального шифрования, это просто пример того, как можно написать этот плагин.

Основная проблема с шифрованием базы данных состоит в том, как хранить секретный ключ. Firebird предоставляет помощника для передачи этого ключа от клиента, но это вовсе не означает, что хранение ключей на клиенте является лучшим способом: это не более чем...
одна возможных альтернатив. Хранение ключей на том же диске, что и база данных, является очень плохим вариантом.

Для эффективного разделения шифрования и доступа к ключу, плагин шифрования базы данных разделён на две части: само шифрование и держатель секретного ключа. Это может быть эффективным подходом, когда вы хотите использовать некоторый хороший алгоритм шифрования, но у вас есть собственный секретный способ хранение ключей.

После того как вы определитесь с плагином и ключом, вы можете включить процесс шифрования.

**Синтаксис**

```
ALTER DATABASE ENCRYPT WITH plugin_name [KEY key_name]
```

*Table 265. Параметры оператора ALTER DATABASE ENCRYPT*

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>plugin_name</td>
<td>Имя плагина шифрования.</td>
</tr>
<tr>
<td>key_name</td>
<td>Имя ключа шифрования.</td>
</tr>
</tbody>
</table>

Шифрование начинается сразу после этого оператора и будет выполняться в фоновом режиме. Нормальная работа с базами данных не нарушается во время шифрования.

Процесс шифрования может быть проконтролирован с помощью поля `MON$CRYPT_PAGE` в псевдотаблице `MON$DATABASE` или смотреть страницу заголовка базы данных с помощью `gstat -e`.

`gstat -h` также будет предоставлять ограниченную информацию о состоянии шифрования.

Например, следующий запрос

```
select MON$CRYPT_PAGE * 100 / MON$PAGES from MON$DATABASE
```

будет отображать процент завершения процесса шифрования.

Необязательное предложение `KEY` позволяет передать имя ключа для плагина шифрования. Что делать с этим именем ключа решает плагин.

Для дешифрования базы данных выполните:

```
ALTER DATABASE DECRIPT
```

Для Linux пример плагина с именем `libDbCrypt_example.so` можно найти в поддиректории `/plugins/`. 
Chapter 15. Управляющие операторы

Начиная с Firebird 3.0, в лексиконе SQL Firebird появился новый класс операторов DSQL, обычно для администрирования аспектов взаимодействия среды клиент/сервер. Обычно такие утверждения начинаются с глагола SET, но могут начинаться и с ключевого слова ALTER.

Инструмент isql также имеет набор команд SET. Эти команды не являются частью лексикона SQL Firebird.

Большинство управляющих операторов влияют только на текущее соединение (сеанс) и не требуют какой-либо дополнительных привилегий от текущего пользователя.

Данные SQL операторы работают вне механизма управления транзакциями, изменения выполненные ими вступают в силу немедленно.

Управляющие операторы доступны, в том числе и в PSQL коде. Это особенно полезно в ON CONNECT триггерах.

Управляющие операторы разбиты на следующие группы:

- управление тайм-аутами;
- управление пулом внешних соединений;
- изменение текущей роли;
- управление поведением типов данных;
- изменение часового пояса сеанса;
- сброс сессионного окружения.

15.1. Поведение типов данных

15.1.1. SET BIND

Назначение
Изменение привязки типа. Обеспечение совместимости со старыми клиентами.

Синтаксис

```sql
SET BIND
  OF {<type-from> | TIME ZONE}
  TO { <type-to> | LEGACY | EXTENDED | NATIVE }
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>type-from</td>
<td>Тип данных для которого задаётся правило преобразования.</td>
</tr>
</tbody>
</table>
Данный оператор позволяет задать правила описания типов возвращаемых клиенту нестандартным способом — тип type-from автоматически преобразуется к типу type-to.

Если используется неполное определение типа (например CHAR вместо CHAR(n)) в левой части SET BIND OF приведения, то преобразование будет осуществляться для всех CHAR столбцов, а не только для CHAR(1).

Специальный неполный тип TIME-ZONE обозначает все типы, а именно {TIME | TIMESTAMP} WITH TIME-ZONE. Когда неполное определение типа используется в правой части оператора (часть TO), сервер автоматически определит недостающие детали этого типа на основе исходного столбца.

Изменение связывания любого NUMERIC и DECIMAL типа не влияет на соответствующий базовый целочисленный тип. Напротив, изменение привязки целочисленного типа данных также влияет на соответствующие NUMERIC и DECIMAL.

Ключевое слово LEGACY в части TO используется, когда тип данных, отсутствующий в предыдущей версии Firebird, должен быть представлен способом понятным для старого клиентского программного обеспечения (возможна некоторая потеря данных). Существуют следующие преобразования в LEGACY типы:

*Table 267. Преобразования в legacy типы*

<table>
<thead>
<tr>
<th>Native тип</th>
<th>Legacy тип</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>CHAR(5)</td>
</tr>
<tr>
<td>DECIML</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>INT128</td>
<td>BIGINT</td>
</tr>
<tr>
<td>TIME WITH TIME-ZONE</td>
<td>TIME WITHOUT TIME-ZONE</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME-ZONE</td>
<td>TIMESTAMP WITHOUT TIME-ZONE</td>
</tr>
</tbody>
</table>

Использование EXTENDED в части TO заставляет Firebird использовать расширенную форму типа в части FROM. В настоящее время он работает только для {TIME | TIMESTAMP} WITH TIME-ZONE — они принудительно приводятся к EXTENDED {TIME | TIMESTAMP} WITH TIME-ZONE.

Установка NATIVE означает, что тип будет использоваться так, как если бы для него не было предыдущих правил преобразования.

Ту же функциональность можно получить используя тэг isc_dpb_set_bind в DPB. Кроме того, преобразование типов в legacy типы доступные в предыдущих версиях Firebird можно установить с помощью параметра DataTypeCompatibility в firebird.conf или databases.conf. Чем позже введено правило (.conf → DPB → SQL), тем выше его приоритет.

692
Example 465. Использование SET BIND OF

```
SELECT CAST('123.45' AS DECFLOAT(16)) FROM RDB$DATABASE;  --native
```

```
CAST
===============
123.45
```

```
SET BIND OF DECFLOAT TO DOUBLE PRECISION;
SELECT CAST('123.45' AS DECFLOAT(16)) FROM RDB$DATABASE;  --double
```

```
CAST
===============
123.4500000000000
```

```
SET BIND OF DECFLOAT(34) TO CHAR;
SELECT CAST('123.45' AS DECFLOAT(16)) FROM RDB$DATABASE;  --всё ещё double
```

```
CAST
===============
123.4500000000000
```

```
SELECT CAST('123.45' AS DECFLOAT(34)) FROM RDB$DATABASE;  --text
```

```
CAST
===============
123.45
```

Example 466. Использование SET BIND OF TIME ZONE TO EXTENDED

Если на стороне клиента отсутствует библиотека ICU, то результат следующего запроса будет таким:

```
SELECT CURRENT_TIMESTAMP FROM RDB$DATABASE;
```

```
CURRENT_TIMESTAMP
===============
```

693
Для того чтобы получить значение смещения времени относительно GMT, выполните следующее:

```sql
SET BIND OF TIME ZONE TO EXTENDED;
SELECT CURRENT_TIMESTAMP FROM RDB$DATABASE;
```

```
CURRENT_TIMESTAMP
=================================================================================================
2020-02-21 19:26:55.6820 +03:00
```

### 15.1.2. SET DECFLOAT

**Назначение**

Изменение режима округления и поведения при ошибках для типа DECFLOAT.

```sql
SET DECFLOAT
{ ROUND <round_mode>
 | TRAPS TO [<trap_opt> [, <trap_opt> ...]] }

<round_mode> ::=  
    CEILING | UP | HALF_UP | HALF_EVEN  
    | HALF_DOWN | DOWN | FLOOR | REROUND

<trap_opt> ::=  
    DIVISION_BY_ZERO | INEXACT | INVALID_OPERATION  
    | OVERFLOW | UNDERFLOW
```

**SET DECFLOAT ROUND**

Оператор SET DECFLOAT ROUND изменяет режим округления для текущей сессии. Поддерживаются следующие режимы округления совместимые со стандартом IEEE:

**CEILING**

Округление сверху. Если все отбрасываемые цифры равны нулю или знак числа отрицателен, последняя не отбрасываемая цифра не меняется. В противном случае последняя не отбрасываемая цифра инкрементируется на единицу (округляется в большую сторону).

**UP**

Округление по направлению от нуля (усечение с приращением). Отбрасываемые значения игнорируются.
HALF_UP
Округление к ближайшему значению. Используется по умолчанию. Если результат равноудаленный, выполняется округление в большую сторону. Если отбрасываемые значения больше чем или равны половине (0,5) единицы в следующей левой позиции, последняя не отбрасываемая цифра инкрементируется на единицу (округляется в большую сторону). В противном случае отбрасываемые значения игнорируются.

HALF_EVEN
Округление к ближайшему значению. Если результат равноудаленный, выполняется округление так, чтобы последняя цифра была четной. Если отбрасываемые значения больше половины (0,5) единицы в следующей левой позиции, последняя не отбрасываемая цифра инкрементируется на единицу (округляется в большую сторону). Если они меньше половины, результат не корректируется (то есть отбрасываемые знаки игнорируются). В противном случае, когда отбрасываемые значения точно равны половине, последняя не отбрасываемая цифра не меняется, если она является четной и инкрементируется на единицу (округляется в большую сторону) в противном случае (чтобы получить четную цифру). Этот режим округления называется также банковским округлением и дает ощущение справедливого округления.

HALF_DOWN
Округление к ближайшему значению. Если результат равноудаленный, выполняется округление в меньшую сторону. Если отбрасываемые значения больше чем или равны половине (0,5) единицы в следующей левой позиции, последняя не отбрасываемая цифра декрементируется на единицу (округляется в меньшую сторону). В противном случае отбрасываемые значения игнорируются.

DOWN
Округление по направлению к нулю (усечение). Отбрасываемые значения игнорируются.

FLOOR
Округление снизу. Если все отбрасываемые цифры равны нулю или знак положителен, последняя не отбрасываемая цифра не меняется. В противном случае (знак отрицателен) последняя не отбрасываемая цифра инкрементируется на единицу.

REROUND
Округление к большему значению, если округляется 0 или 5, в противном случае округление происходит к меньшему значению.

Example 467. Изменение режима округления

```sql
SET DECFLOAT ROUND HALF_DOWN;
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CEILING</td>
<td>12.35</td>
<td>12.35</td>
<td>12.35</td>
<td>12.36</td>
<td>12.41</td>
<td>-12.34</td>
</tr>
<tr>
<td>UP</td>
<td>12.35</td>
<td>12.35</td>
<td>12.35</td>
<td>12.36</td>
<td>12.41</td>
<td>-12.35</td>
</tr>
</tbody>
</table>


### Режимы округления

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HALF_UP</td>
<td>12.34</td>
<td>12.35</td>
<td>12.35</td>
<td>12.36</td>
<td>12.41</td>
<td>-12.35</td>
</tr>
<tr>
<td>HALF_EVEN</td>
<td>12.34</td>
<td>12.34</td>
<td>12.35</td>
<td>12.36</td>
<td>12.40</td>
<td>-12.34</td>
</tr>
<tr>
<td>HALF_DOWN</td>
<td>12.34</td>
<td>12.34</td>
<td>12.35</td>
<td>12.35</td>
<td>12.40</td>
<td>-12.34</td>
</tr>
<tr>
<td>DOWN</td>
<td>12.34</td>
<td>12.34</td>
<td>12.34</td>
<td>12.35</td>
<td>12.40</td>
<td>-12.34</td>
</tr>
<tr>
<td>FLOOR</td>
<td>12.34</td>
<td>12.34</td>
<td>12.34</td>
<td>12.35</td>
<td>12.40</td>
<td>-12.35</td>
</tr>
<tr>
<td>REROUND</td>
<td>12.34</td>
<td>12.34</td>
<td>12.34</td>
<td>12.36</td>
<td>12.41</td>
<td>-12.34</td>
</tr>
</tbody>
</table>

**SET DECFLOAT TRAPS**

Оператор `SET DECFLOAT TRAPS` изменяет поведение ошибок при операциях с типом `DECFLOAT`.

По умолчанию исключения генерируются для следующих ситуаций: `DIVISION_BY_ZERO, INVALID_OPERATION, OVERFLOW`; это значение по умолчанию соответствует поведению, определенному в стандарте SQL: 2016 для `DECFLOAT`. Этот оператор контролирует, приводят ли определенные исключительные условия к ошибке ("ловушка" или "trap") или альтернативной обработке (например, потеря значимости возвращает 0, если не установлена, либо переполнение возвращает бесконечность). Первоначальная конфигурация соединения также может быть указана с помощью тега `isc_dpb_decfloat_traps` с желаемыми значениями `trap_opt`, разделенными запятыми, в виде строкового значения.

Допустимые варианты ловушек (исключительных условий):

- `Division_by_zero` (по умолчанию)
- `Inexact`
- `Invalid_operation` (по умолчанию)
- `Overflow` (по умолчанию)
- `Underflow`  

**Example 468. Установка ситуаций для которых будет генерироваться исключение**

```sql
SET DECFLOAT TRAPS TO Division_by_zero, Inexact, Invalid_operation, Overflow, Underflow;
```

### 15.2. Тайм-ауты

В Firebird существует два вида тайм-аута:

- тайм-аут простого соединения;
- тайм-аут выполнения SQL оператора.
15.2.1. Тайм-аут выполнения SQL оператора

Данная функциональность позволяет автоматически прекратить выполнение SQL оператора, если он выполняется дольше заданного значения тайм-аута.

Данная функция может быть полезна для:

• Администраторов баз данных. Они получают инструмент для ограничения времени выполнения тяжёлых запросов, которые потребляют много ресурсов;
• Разработчиков приложений. Они могут использовать тайм-ауты SQL операторов при написании и отладке сложных запросов с заранее неизвестным временем выполнения;
• Тестеров, которые могут использовать тайм-ауты SQL операторов для обнаружения долго выполняющихся запросов и обеспечения конечного времени выполнения набора тестов.

Эта функциональность работает следующим образом. Когда начинается выполнение оператора (или открывается курсор) Firebird запускает специальный таймер. Выборка записей (fetch) не сбрасывает таймер. Таймер останавливается если выполнение SQL оператора закончено или извлечена (fetch) последняя запись.

По истечении тайм-аута:

• Если выполнение SQL оператора активно, оно останавливается в заданный момент.
• Если SQL оператор не активен в данный момент (например между выборками (fetch)), то он будет помечен как отменённый, следующая выборка (fetch) прервет выполнение и будет возвращена ошибка.

Значение тайм-аута может быть установлено:

• На уровне базы данных. Значение параметра StatementTimeout может быть установлено в firebird.conf (или databases.conf) администратором базы данных. Область действия все операторы во всех соединениях. Параметр StatementTimeout устанавливает тайм-аут в секундах, по истечении которого выполнение SQL операторов будет отменено. Ноль означает, что тайм-аут не установлен. Значение по умолчанию равно 0.
• На уровне соединения. Может быть установлен с использованием API (в миллисекундах) или с помощью SQL оператора SET STATEMENT TIMEOUT. Область действия текущее подключение.
• На уровне оператора. Может быть установлен с использованием API (в миллисекундах). Область действия текущий SQL оператор.

Эффективное значение тайм-аута SQL оператора вычисляется каждый раз, когда запускается SQL оператор (открывается курсор), следующим образом:

• если тайм-аут не установлен на уровне оператора, будет использовано значение тайм-аута уровня соединения;
• если тайм-аут не установлен на уровне соединения, будет использовано значение тайм-аута уровня базы данных;
значение тайм-аута не может быть больше, чем значение установленное на уровне базы данных. Таким образом, значение тайм-аута может перекрываться разработчиком приложения в более низких областях, но оно не может выйти за пределы установленные DBA в конфигурации.

Нулевой тайм-аут не обозначает отсутствие тайм-аута, просто в этом случае таймер выполнения оператора не запускается.

Несмотря на то, что тайм-аут выполнения SQL оператора может быть установлен в миллисекундах, абсолютная точность не гарантируется. При высокой нагрузке он может быть менее точным. Единственная гарантия которую может дать Firebird это то, что тайм-аут не сработает раньше указанного момента. Клиентское приложение может ждать больше времени, чем установленное значение тайм-аута если движку Firebird необходимо отменить множество действий связанных с отменой оператора.

Тайм-аут выполнения оператора игнорируется для всех внутренних запросов, которые используется движком Firebird. Кроме того, тайм-аут игнорируется для DDL операторов.

**SET STATEMENT TIMEOUT**

Назначение
Установка тайм-аута выполнения SQL операторов на уровне соединения.

Доступно в
DSQL

Синтаксис:

```sql
SET STATEMENT TIMEOUT value [HOUR | MINUTE | SECOND | MILLISECOND]
```

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Значение тайм-аута выполнения SQL операторов в указанных единицах измерения времени. Если единица измерения времени не указана, то по умолчанию значение тайм-аута измеряется в секундах.</td>
</tr>
</tbody>
</table>

Устанавливает значение тайм-аута выполнения SQL операторов на уровне текущего соединения. Если единица времени не указана, то по умолчанию тайм-аут будет учитываться в секундах.

Данный SQL оператор работает вне механизма управления транзакциями и вступают в силу немедленно.

*Example 469. Установка тайм-аута выполнения SQL оператора*

```
SET STATEMENT TIMEOUT 2 MINUTE
```
Интерактивный инструмент isql дополнительно поддерживает команду:

```
SET LOCAL_TIMEOUT int
```

Эта команда позволяет установить тайм-аут выполнения оператора (в миллисекундах) для следующего оператора. После выполнения SQL оператора он автоматически сбрасывается в ноль.

### 15.2.2. Тайм-аут простого соединения

Данная функциональность позволяет автоматически закрывать пользовательские подключения после периода бездействия. Она может быть использована администраторами баз данных, чтобы принудительно закрывать старые неактивные соединения и освобождать связанные с ними ресурсы. Приложения и инструменты разработчика могут использовать её как замену самодельного контроля за временем жизни подключения.

Рекомендуется (но не обязательно) устанавливать тайм-аут простого в разумное большое значение, например, несколько часов. По умолчанию эта функция отключена.

Эта функциональность работает следующим образом. Когда пользовательский вызов API покидает движок, запускается специальный таймер связанный с текущим подключением. Как только пользовательский вызов входит в движок, таймер ожидания останавливается. Если тайм-аут простого истечёт движок закроет соединение так как будто произошло асинхронная отмена подключения:

- все активные операторы и курсоры закрываются;
- все активные транзакции откатываются;
- сетевые соединения не закрываются в данный момент. Это позволяет клиентскому приложению получить точный код ошибки при следующем вызове API. Сетевое соединение будет закрыто на стороне сервера после того, как ошибка сообщена, или если клиентская сторона отключится по истечению тайм-аута сети.

Тайм-аут простого соединения может быть установлен:

- На уровне базы данных. Значение параметра `ConnectionIdleTimeout` может быть установлено в `firebird.conf` (или `databases.conf`) администратором базы данных. Область действия все пользовательские подключения, исключая системные подключения (`garbage collector`, `cache writer`, и др.). Параметр `ConnectionIdleTimeout` устанавливает тайм-аут в минутах, по истечении которого неактивное соединение будет разорвано движком. Ноль означает, что тайм-аут не установлен. Значение по умолчанию равно 0.

- На уровне подключения. Может быть установлен с использованием API (в секундах) или с помощью SQL оператора `SET SESSION IDLE TIMEOUT`. Область действия все операторы в текущем подключении.

Эффективное значение тайм-аута простого вычисляется каждый раз, когда пользовательский вызов API покидает движок, следующим образом:
• если тайм-аут не установлен на уровне подключения, будет использовано значение уровня базы данных;

• значение тайм-аута не может быть больше, чем значение установленное на уровне базы данных. Таким образом, значение тайм-аута просто может перекрываться разработчиком приложения для заданного подключения, но оно не может выйти за пределы установленные DBA в конфигурации.

Нулевой тайм-аут не обозначает отсутствие тайм-аута, просто в этом случае таймер ожидания не запускается.

Несмотря на то, что тайм-аут простоя может быть установлен в секундах, абсолютная точность не гарантируется. При высокой нагрузке он может быть менее точным. Единственная гарантия которую может дать Firebird это то, что тайм-аут не сработает раньше указанного момента.

**SET SESSION IDLE TIMEOUT**

**Назначение**
Установка тайм-аута простоя соединения на уровне соединения.

**Доступно в**
DSQL.

**Синтаксис**

```
SET SESSION IDLE TIMEOUT value [HOUR | MINUTE | SECOND]
```

**Table 269. Параметры оператора SET SESSION IDLE TIMEOUT**

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Значение тайм-аута простоя в указанных единицах измерения времени. Если единица измерения времени не указано, то по умолчанию значение тайм-аута измеряется в минутах.</td>
</tr>
</tbody>
</table>

Устанавливает значение тайм-аута простоя на уровне текущего соединения. Если единица времени не указана, то по умолчанию тайм-аут будет учитываться в минутах.

Информация

Данный SQL оператор работает вне механизма управления транзакциями и вступают в силу немедленно.

**Example 470. Установка тайм-аута простоя соединения**

```
SET SESSION IDLE TIMEOUT 8 HOUR
```
15.3. Пул внешних соединений

Каждое внешнее соединение (созданное оператором EXECUTE STATEMENT … ON EXTERNAL) при создании связывается с пулом соединений (подробнее см. Пул внешних подключений). Данная группа операторов позволяет управлять пулом внешних соединений. При его подготовке они описываются как DDL операторы, но имеют немедленный эффект: то есть они выполняются немедленно и полностью, не дожидаясь фиксации транзакции. Изменения применяются к экземпляру пула в памяти в текущем процессе Firebird. Поэтому изменение в одном классическом процессе не влияет на другие классические процессы. Изменения не являются постоянными и после перезапуска Firebird будет использовать настройки пула из firebird.conf.

Для выполнения операторов данной группы требуется системная привилегия MODIFY_EXT_CONN_POOL. Подробнее о системных привилегиях см. CREATE ROLE.

15.3.1. ALTER EXTERNAL CONNECTIONS POOL SET SIZE

Назначение
Устанавливает максимальное количество бездействующих соединений.

Синтаксис

```
ALTER EXTERNAL CONNECTIONS POOL SET SIZE size
```

Таблица 270. Параметры оператора ALTER EXTERNAL CONNECTIONS POOL SET SIZE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Размер пула внешних соединений. Допустимые значения от 0 до 1000.</td>
</tr>
</tbody>
</table>

Оператор ALTER EXTERNAL CONNECTIONS POOL SET SIZE устанавливает максимальное количество бездействующих соединений в пуле внешних соединений. Допустимые значения от 0 до 1000. Нулевое значение обозначает что пул выключен. Значение по умолчанию определяется в firebird.conf (параметр ExtConnPoolSize).

15.3.2. ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME

Назначение
Устанавливает время жизни бездействующих соединений.

Синтаксис

```
ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME value <time_part>
```

```
<time_part> ::= SECOND | MINUTE | HOUR
```

Таблица 271. Параметры оператора ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME
### Параметр | Описание
--- | ---
value | Время жизни бездействующих соединений.

Оператор `ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME` устанавливает время жизни бездействующих соединений в пуле внешних соединений. Допустимые значения от 1 секунды до 24 часов. Значение по умолчанию определяется в `firebird.conf` (параметр `ExtConnPoolLifeTime` в секундах).

#### 15.3.3. `ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL`  
**Назначение**  
Закрывает все бездействующие соединения.  
**Синтаксис**

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL
```

Оператор `ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL` закрывает все бездействующие соединения в пуле внешних соединений. Все активные соединения будут отсоединены от пула (такие соединения будут немедленно закрыты, когда они не будут использоваться).

#### 15.3.4. `ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST`  
**Назначение**  
Закрывает бездействующие соединения у которых истекло время жизни.  
**Синтаксис**

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST
```

Оператор `ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST` закрывает бездействующие соединения в пуле у которых истекло время жизни.

### 15.4. Изменение текущей роли

#### 15.4.1. `SET ROLE`  
**Назначение**  
Изменение текущей роли.  
**Доступно в**  
DSQL.  
**Синтаксис**

```
SET ROLE rolename
```
Согласно стандарту SQL-2008 оператор SET ROLE позволяет установить контекстной переменной CURRENT_ROLE одну из назначенных ролей для пользователя CURRENT_USER или роль, полученную в результате доверительной аутентификации (в этом случае оператор принимает вид SET TRUSTED ROLE).

Example 471. Изменение текущей роли

```
SET ROLE manager;
SELECT current_role FROM rdb$database;
```

ROLE
=======================
MANAGER

15.4.2. SET TRUSTED ROLE

Назначение
Установка доверенной роли.

Доступно в
DSQL

Синтаксис

```
SET TRUSTED ROLE
```

Оператор SET TRUSTED ROLE включает доступ доверенной роли, при условии, что CURRENT_USER получен с помощью доверительной аутентификации и роль доступна.

Идея отдельной команды SET TRUSTED ROLE состоит в том, чтобы при подключении доверенного пользователя не указывать никакой дополнительной информации о роли. SET TRUSTED ROLE делает доверенную роль (если таковая существует) текущей ролью без дополнительной деятельности, связанной с установкой параметров DBP.

Доверенная роль это не специальный тип роли, ей может быть любая роль, созданная с помощью оператора CREATE ROLE или предопределённая системная роль RDB$ADMIN. Она становится доверенной ролью для подключения, когда подсистема отображения объектов безопасности (security objects mapping subsystem) находит соответствие между результатом аутентификации, полученным от плагина и локальным или глобальным отображением (mapping) для текущей базы данных. Роль даже может быть той, которая не предоставлена явно этому доверенному пользователю.

Table 272. Параметры оператора SET ROLE

<table>
<thead>
<tr>
<th>Параметр</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolename</td>
<td>Имя устанавливаемой роли.</td>
</tr>
</tbody>
</table>
Доверенная роль не назначается при подключении по умолчанию. Можно изменить это поведение, используя соответствующий плагин аутентификации и операторы (CREATE | ALTER) MAPPING.

Примером использования доверенной роли является назначение системной роли RDB$ADMIN для администраторов Windows, когда используется доверительная аутентификация Windows.

### 15.5. Управление часовым поясом сеанса

#### 15.5.1. SET TIME ZONE

**Назначение**

Изменение часового пояса сеанса.

**Синтаксис**

```sql
SET TIME ZONE { <time-zone-string> | LOCAL }

<time-zone-string> ::= 
  '<time-zone>'

<time-zone> ::= 
  <time-zone-region> 
  | [+-] <hour-displacement> [: <minute-displacement>]
```

Немедленно изменяет часовой пояс сеанса (текущего подключения).

Указание LOCAL вернет к начальному часовому поясу сеанса (либо по умолчанию, либо как указано в свойстве соединения isc_dbp_session_time_zone).

Получить текущий часовой пояс сеанса можно с использованием функции RDB$GET_CONTEXT с аргументами 'SYSTEM' для пространства имён и 'SESSION_TIMEZONE' в качестве имени переменной.

**Example 472. Изменение часового пояса сеанса**

```sql
set time zone '-02:00';
select rdb$get_context('SYSTEM', 'SESSION_TIMEZONE') from rdb$database;
-- returns -02:00

set time zone 'America/Sao_Paulo';
select rdb$get_context('SYSTEM', 'SESSION_TIMEZONE') from rdb$database;
```
15.6. ALTER SESSION RESET

Назначение
Сброс сессионного окружения.

Доступно в
DSQL

Синтаксис

```
ALTER SESSION RESET
```

Сбрасывает сеансовое окружение (подключения) к исходному состоянию. Эта функциональность полезна если сеанс используется повторно, вместо того чтобы производить отключение/подключение.

Данный оператор делает следующее:

- генерируется ошибка (isc_ses_reset_err), если в текущем соединении существует какая-либо открытая транзакция, кроме текущей транзакции и подготовленных транзакций 2PC, которые разрешены и игнорируются этой проверкой;
- системная переменная RESETTING устанавливается в TRUE;
- запускаются триггеры базы данных на событие ON DISCONNECT, если они присутствуют и разрешены для текущего соединения;
- текущая пользовательская транзакция откатывается (ROLLBACK), если она есть. Если в текущей активной транзакции были произведены изменения, то будет выдано предупреждение;
- сбрасывает установленные параметры DECIMAL (BIND, TRAP и ROUND) в значения по умолчанию;
- сбрасывает тайм-ауты сессии и оператора в 0;
- удаляет все контекстные переменные из пространства имён USER_SESSION;
- сбрасывает роль в значение переданное в DPB (указанное при подключении) и очищает кеш привилегий (если роль была изменена с помощью оператора SET ROLE);
- очищает содержимое всех используемых глобальных таблиц уровня соединения (GLOBAL TEMPORARY TABLE ... ON COMMIT PRESERVE ROWS);
- запускаются триггеры базы данных на событие ON CONNECT, если они присутствуют и разрешены для текущего соединения;
- начинает новую транзакцию с теми же свойствами, что и транзакция, которая была
отменена (если транзакция присутствовала до сброса);
• системная переменная `RESETTING` устанавливается в `FALSE`.

15.6.1. Обработка ошибок

Ошибка, возникшая в триггере `ON DISCONNECT`, прерывает сброс сеанса и оставляет состояние сеанса неизменным. Такие ошибки сообщаются с кодом основной ошибки `isc_session_reset_err` и текстом ошибки “Cannot reset user session”.

Ошибки, возникающие после того, как триггеры `ON DISCONNECT` выполнены, прерывают выполнение оператора сброса сеанса и само соединение. Такие ошибки сообщались с кодом основной ошибки `isc_session_reset_failed` и текстом ошибки “Reset of user session failed. Connection is shut down”. Последующие операции по подключению (кроме отсоединения) завершатся ошибкой `isc_att_shutdown`.

*См. также:* `RESETTING`.  

Chapter 15. Управляющие операторы
Appendix A: Дополнительные статьи

Поле RDB$VALID_BLR

В системных таблицах RDB$PROCEDURES, RDB$FUNCTIONS и RDB$TRIGGERS присутствует поле RDB$VALID_BLR. Оно предназначено для сигнализации о возможной недействительности модуля PSQL (процедуры или триггера) после изменения доменов или столбцов таблиц, от которых он зависит. При возникновении описанной выше ситуации поле RDB$VALID_BLR устанавливается в 0 для процедур или триггеров, код которых возможно является недействительным.

Как работает инвалидация

В триггерах, процедурах и функциях зависимости возникают от столбцов таблицы, к которой они обращаются, а так же от любого параметра или переменной, определенных в модуле с использованием предложения TYPE OF.

После того как ядро Firebird изменило любой домен, включая неявные домены, создаваемые внутри при определении столбцов или параметров, Firebird производит внутреннюю перекомпиляцию всех зависимостей.

Инвалидация происходит для процедур, функций, пакетов и триггеров, но не для блоков DML операторов, которые выполняются при помощи EXECUTE BLOCK.

Любой модуль, который не удалось перекомпилировать из-за несовместимости, возникающей из-за изменения домена, помечается как недействительный (поле RDB$VALID_BLR устанавливается в 0 в записи соответствующей системной таблице (RDB$PROCEDURES или RDB$TRIGGERS)).

Возобновление (установка RDB$VALID_BLR в 1) происходит когда

1. домен изменён снова и его новое определение совместимо с ранее недействительным определением модуля; или
2. ранее недействительный модуль изменён так, что соответствовать новому определению домена.

Нижеприведённый запрос находит процедуры и триггеры, зависящие от определённого домена (в примере это домен 'MYDOMAIN'), и выводит информацию о состоянии поля RDB$VALID_BLR:

```sql
WITH VALID_PSQL
     (PSQL_TYPE, ROUTE_NAME, VALID)
AS (SELECT 'Procedure',
```
RDB$PROCEDURE_NAME,
RDB$VALID_BLR
FROM
RDB$PROCEDURES
WHERE
RDB$PROCEDURES.RDB$PACKAGE_NAME IS NULL
UNION ALL
SELECT
'Function',
RDB$FUNCTION_NAME,
RDB$VALID_BLR
FROM
RDB$FUNCTIONS
WHERE
RDB$FUNCTIONS.RDB$PACKAGE_NAME IS NULL
UNION ALL
SELECT
'Package',
RDB$PACKAGE_NAME,
RDB$VALID_BODY_FLAG
FROM
RDB$PACKAGES
UNION ALL
SELECT
'Trigger',
RDB$TRIGGER_NAME,
RDB$VALID_BLR
FROM
RDB$TRIGGERS
WHERE
RDB$TRIGGERS.RDB$SYSTEM_FLAG = 0
SELECT
PSQL_TYPE,
ROUTE_NAME,
VALID
FROM
VALID_PSQL
WHERE
EXISTS(SELECT *
FROM
RDB$DEPENDENCIES
WHERE
RDB$DEPENDENT_NAME = VALID_PSQL.ROUTE_NAME
AND RDB$DEPENDED_ON_NAME = 'MYDOMAIN');

/*
Замените MYDOMAIN фактическим именем проверяемого домена. Используйте заглавные буквы, если
dомен создавался нечувствительным к регистру – в противном случае используйте точное написание
Следующий запрос находит процедуры и триггеры, зависящие от определённого столбца таблицы (в примере это столбец 'MYCOLUMN' таблицы 'MYTABLE'), и выводит информацию о состоянии поля RDB$VALID_BLR:

WITH VALID_PSQL (PSQL_TYPE, ROUTE_NAME, VALID) AS (
    SELECT 'Procedure', RDB$PROCEDURE_NAME, RDB$VALID_BLR
    FROM RDB$PROCEDURES
    WHERE RDB$PROCEDURES.RDB$PACKAGE_NAME IS NULL
    UNION ALL
    SELECT 'Function', RDB$FUNCTION_NAME, RDB$VALID_BLR
    FROM RDB$FUNCTIONS
    WHERE RDB$FUNCTIONS.RDB$PACKAGE_NAME IS NULL
    UNION ALL
    SELECT 'Package', RDB$PACKAGE_NAME, RDB$VALID_BODY_FLAG
    FROM RDB$PACKAGES
    UNION ALL
    SELECT 'Trigger', RDB$TRIGGER_NAME, RDB$VALID_BLR
    FROM RDB$TRIGGERS
    WHERE RDB$TRIGGERS.RDB$SYSTEM_FLAG = 0)

SELECT PSQL_TYPE, ROUTE_NAME, VALID
FROM VALID_PSQL
WHERE EXISTS(SELECT *
 FROM RDB$DEPENDENCIES D
 WHERE D.RDB$DEPENDENT_NAME = VALID_PSQL.ROUTE_NAME
 AND D.RDB$DEPENDED_ON_NAME = 'MYTABLE'
 AND D.RDB$FIELD_NAME = 'MYCOLUMN');

/*
Замените MYTABLE и MYCOLUMN фактическими именами проверяемой таблицы и её столбца.
Используйте заглавные буквы, если таблица и её столбец создавались нечувствительными к регистру — в противном случае используйте точное написание имени таблицы и её столбца с учётом регистра
*/

Все случаи возникновения недействительных модулей, вызванных изменениями доменов/столбцов, отражаются в поле RDB$VALID_BLR. Тем не менее, другие виды изменения, таких как изменения количества входных или выходных параметров процедур и так далее, не влияют на поле проверки, даже если потенциально они могут привести к недействительности модуля. Типичные сценарии могут быть следующими:

1. Процедура (B) определена так, что она вызывает другую процедуру (A) и считывает выходные параметры из неё. В этом случае зависимость будет зарегистрирована в RDB$DEPENDENCIES. В последствии вызываемая процедура (A) может быть изменена для изменения или удаления одного и более выходных параметров. Оператор ALTER PROCEDURE A приведёт к ошибке при выполнении фиксации транзакции.

2. Процедура (B) вызывает процедуру (A), передавая ей значения в качестве входных параметров. Никаких зависимостей не будет зарегистрировано в RDB$DEPENDENCIES. Последующие модификации входных параметров процедуры A будут позволены. Отказ произойдет во время выполнения, когда B вызовет A с несогласованным набором входных параметров.

Другие замечания

• Для модулей PSQL, наследованных от более ранних версий Firebird (включая многие системные триггеры, даже если база данных создавалась под версией Firebird 2.1 или выше), поле RDB$VALID_BLR имеет значение NULL. Это не означает, что их BLR является недействительным.

• Команды утилиты командной строки isql SHOW PROCEDURES, SHOW FUNCTIONS и SHOW TRIGGERS при выводе информации отмечают звёздочкой модули, у которых поле RDB$VALID_BLR равно 0. Команды SHOW PROCEDURE procname, SHOW FUNCTION funcname и SHOW TRIGGER trigname, выводящие на экран код PSQL...
Замечание о равенстве

Это замечание об операторах равенства и неравенства применяется повсюду в СУБД Firebird.

Оператор “=”, который используется во многих условиях, сравнивает только значения со значениями. В соответствии со стандартом SQL, NULL не является значением и, следовательно, два значения NULL не равны ни неравны друг с другом. Если необходимо, чтобы значения NULL соответствовали друг другу при объединении, используйте оператор IS NOT DISTINCT FROM. Этот оператор возвращает истину, если операнда имеет то же значение, или, если оба они равны NULL.

```
SELECT *
FROM A
JOIN B ON A.id IS NOT DISTINCT FROM B.code
```

Точно так же, если вы хотите чтобы значения NULL отличались от любого значения и два значения NULL считались равными, используйте оператор IS DISTINCT FROM вместо оператора “<>”.

```
SELECT *
FROM A
JOIN B ON A.id IS DISTINCT FROM B.code
```
Appendix B: Обработка ошибок, коды и сообщения

Приложение включает:

- Коды ошибок SQLSTATE и их описание
- Коды ошибок GDSCODE их описание, и SQLCODE

Пользовательские исключения

В Firebird DDL существует простой синтаксис для создания пользовательских исключений для использования в PSQL, с текстами сообщений до 1021 символов. Подробности см. CREATE EXCEPTION главы DDL. Подробности использования пользовательских исключений см. EXCEPTION главы PSQL.

В большинстве случаев, коды ошибок SQLCODE не соотносятся с кодами SQLSTATE “один в один”. Коды ошибок SQLSTATE соответствуют SQL стандарту. В то же время SQLCODE использовались много лет и в настоящий момент считаются устаревшими. В следующих версиях поддержка SQLCODE может полностью прекратиться.

Коды ошибок SQLSTATE и их описание

В данной главе приведены коды ошибок для контекстной переменной SQLSTATE и их описания. Коды ошибок SQLSTATE построены следующим образом: пяти символьный код ошибки состоит из SQL класса ошибки (2 символа) и SQL подкласса (3 символа).

Table 273. Коды ошибок SQLSTATE

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Связанное сообщение</th>
<th>Примечание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLCLASS 00</td>
<td>0 Success</td>
<td>Успех</td>
</tr>
<tr>
<td>SQLCLASS 01</td>
<td>General Warning</td>
<td>Общее предупреждение</td>
</tr>
<tr>
<td>01000</td>
<td>Cursor operation conflict</td>
<td>Конфликт при операции с курсором</td>
</tr>
<tr>
<td>01001</td>
<td>Disconnect error</td>
<td>Ошибка, связанная с разъединением</td>
</tr>
<tr>
<td>01002</td>
<td>NULL value eliminated in set function</td>
<td>Значение NULL устраняется в определении функции</td>
</tr>
<tr>
<td>01003</td>
<td>String data, right-truncated</td>
<td>Строковые данные, обрезание справа</td>
</tr>
<tr>
<td>01004</td>
<td>Insufficient item descriptor areas</td>
<td>Недостаточно элементов в области дескрипторов</td>
</tr>
<tr>
<td>01005</td>
<td>Privilege not revoked</td>
<td>Привилегии не отозваны</td>
</tr>
<tr>
<td>01006</td>
<td>Privilege not granted</td>
<td>Привилегии не выданы</td>
</tr>
</tbody>
</table>

712
### SQLSTATE Связанное сообщение Примечание

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Связанное сообщение</th>
<th>Примечание</th>
</tr>
</thead>
<tbody>
<tr>
<td>01008</td>
<td>Implicit zero-bit padding</td>
<td>Неявное обрезание нулевого бита</td>
</tr>
<tr>
<td>01100</td>
<td>Statement reset to unprepared</td>
<td>Оператор сброшен в состояние unprepared</td>
</tr>
<tr>
<td>01101</td>
<td>Ongoing transaction has been committed</td>
<td>Текущая транзакция завершена COMMIT</td>
</tr>
<tr>
<td>01102</td>
<td>Ongoing transaction has been rolled back</td>
<td>Текущая транзакция завершена ROLLED BACK</td>
</tr>
</tbody>
</table>

**SQLCLASS 02 (No Data)**

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Сообщение</th>
</tr>
</thead>
<tbody>
<tr>
<td>02000</td>
<td>No data found or no rows affected</td>
</tr>
</tbody>
</table>

**SQLCLASS 07 (Dynamic SQL error)**

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Сообщение</th>
</tr>
</thead>
<tbody>
<tr>
<td>07000</td>
<td>Dynamic SQL error</td>
</tr>
<tr>
<td>07001</td>
<td>Wrong number of input parameters</td>
</tr>
<tr>
<td>07002</td>
<td>Wrong number of output parameters</td>
</tr>
<tr>
<td>07003</td>
<td>Cursor specification cannot be executed</td>
</tr>
<tr>
<td>07004</td>
<td>USING clause required for dynamic parameters</td>
</tr>
<tr>
<td>07005</td>
<td>Prepared statement not a cursor-specification</td>
</tr>
<tr>
<td>07006</td>
<td>Restricted data type attribute violation</td>
</tr>
<tr>
<td>07007</td>
<td>USING clause required for result fields</td>
</tr>
<tr>
<td>07008</td>
<td>Invalid descriptor count</td>
</tr>
<tr>
<td>07009</td>
<td>Invalid descriptor index</td>
</tr>
</tbody>
</table>

**SQLCLASS 08 (Connection Exception)**

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Сообщение</th>
</tr>
</thead>
<tbody>
<tr>
<td>08001</td>
<td>Client unable to establish connection</td>
</tr>
<tr>
<td>08002</td>
<td>Connection name in use</td>
</tr>
<tr>
<td>08003</td>
<td>Connection does not exist</td>
</tr>
<tr>
<td>08004</td>
<td>Server rejected the connection</td>
</tr>
<tr>
<td>08006</td>
<td>Connection failure</td>
</tr>
<tr>
<td>08007</td>
<td>Transaction resolution unknown</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Связанное сообщение</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
</tr>
<tr>
<td>SQLCLASS 0A (Feature Not Supported)</td>
<td>Class ошибок 0A (Возможность не поддерживается)</td>
</tr>
<tr>
<td>0A000</td>
<td>Feature Not Supported</td>
</tr>
<tr>
<td>SQLCLASS 0B (Invalid Transaction Initiation)</td>
<td>Class ошибок 0B (неправильная инициализация транзакции)</td>
</tr>
<tr>
<td>0B000</td>
<td>Invalid transaction initiation</td>
</tr>
<tr>
<td>SQLCLASS 0L (Invalid Grantor)</td>
<td>Неверный грантодатель</td>
</tr>
<tr>
<td>0L000</td>
<td>Invalid grantor</td>
</tr>
<tr>
<td>SQLCLASS 0P (Invalid Role Specification)</td>
<td>Class ошибок 0P (неправильная спецификация роли)</td>
</tr>
<tr>
<td>0P000</td>
<td>Invalid role specification</td>
</tr>
<tr>
<td>SQLCLASS 0U (Attempt to Assign to Non-Updatable Column)</td>
<td>Class ошибок 0U (попытка присвоения не обновляемому столбцу)</td>
</tr>
<tr>
<td>0U000</td>
<td>Attempt to assign to non-updatable column</td>
</tr>
<tr>
<td>SQLCLASS 0V (Attempt to Assign to Ordering Column)</td>
<td>Class ошибок 0V (попытка присвоения сортируемому столбцу)</td>
</tr>
<tr>
<td>0V000</td>
<td>Attempt to assign to Ordering column</td>
</tr>
<tr>
<td>SQLCLASS 20 (Case Not Found For Case Statement)</td>
<td>Class 20 (не обнаружено вариантов для предложения CASE)</td>
</tr>
<tr>
<td>20000</td>
<td>Case not found for case statement</td>
</tr>
<tr>
<td>SQLCLASS 21 (Cardinality Violation)</td>
<td>Class 21 (Нарушение определения)</td>
</tr>
<tr>
<td>21000</td>
<td>Cardinality violation</td>
</tr>
<tr>
<td>21S01</td>
<td>Insert value list does not match column list</td>
</tr>
<tr>
<td>21S02</td>
<td>Degree of derived table does not match column list</td>
</tr>
<tr>
<td>SQLCLASS 22 (Data Exception)</td>
<td>Class ошибок 22 (исключения, вызванные данными)</td>
</tr>
<tr>
<td>22000</td>
<td>Data exception</td>
</tr>
<tr>
<td>22001</td>
<td>String data, right truncation</td>
</tr>
<tr>
<td>22002</td>
<td>Null value, no indicator parameter</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Связанное сообщение</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>22003</td>
<td>Numeric value out of range</td>
</tr>
<tr>
<td>22004</td>
<td>Null value not allowed</td>
</tr>
<tr>
<td>22005</td>
<td>Error in assignment</td>
</tr>
<tr>
<td>22006</td>
<td>Null value in field reference</td>
</tr>
<tr>
<td>22007</td>
<td>Invalid datetime format</td>
</tr>
<tr>
<td>22008</td>
<td>Datetime field overflow</td>
</tr>
<tr>
<td>22009</td>
<td>Invalid time zone displacement value</td>
</tr>
<tr>
<td>2200A</td>
<td>Null value in reference target</td>
</tr>
<tr>
<td>2200B</td>
<td>Escape character conflict</td>
</tr>
<tr>
<td>2200C</td>
<td>Invalid use of escape character</td>
</tr>
<tr>
<td>2200D</td>
<td>Invalid escape octet</td>
</tr>
<tr>
<td>2200E</td>
<td>Null value in array target</td>
</tr>
<tr>
<td>2200F</td>
<td>Zero-length character string</td>
</tr>
<tr>
<td>2200G</td>
<td>Most specific type mismatch</td>
</tr>
<tr>
<td>22010</td>
<td>Invalid indicator parameter value</td>
</tr>
<tr>
<td>22011</td>
<td>Substring error</td>
</tr>
<tr>
<td>22012</td>
<td>Division by zero</td>
</tr>
<tr>
<td>22014</td>
<td>Invalid update value</td>
</tr>
<tr>
<td>22015</td>
<td>Interval field overflow</td>
</tr>
<tr>
<td>22018</td>
<td>Invalid character value for cast</td>
</tr>
<tr>
<td>22019</td>
<td>Invalid escape character</td>
</tr>
<tr>
<td>2201B</td>
<td>Invalid regular expression</td>
</tr>
<tr>
<td>2201C</td>
<td>Null row not permitted in table</td>
</tr>
<tr>
<td>22020</td>
<td>Invalid limit value</td>
</tr>
<tr>
<td>22021</td>
<td>Character not in repertoire</td>
</tr>
<tr>
<td>22022</td>
<td>Indicator overflow</td>
</tr>
<tr>
<td>22023</td>
<td>Invalid parameter value</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Связанное сообщение</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>22024</td>
<td>Character string not properly terminated</td>
</tr>
<tr>
<td>22025</td>
<td>Invalid escape sequence</td>
</tr>
<tr>
<td>22026</td>
<td>String data, length mismatch</td>
</tr>
<tr>
<td>22027</td>
<td>Trim error</td>
</tr>
<tr>
<td>22028</td>
<td>Row already exists</td>
</tr>
<tr>
<td>2202D</td>
<td>Null instance used in mutator function</td>
</tr>
<tr>
<td>2202E</td>
<td>Array element error</td>
</tr>
<tr>
<td>2202F</td>
<td>Array data, right truncation</td>
</tr>
<tr>
<td><strong>SQLCLASS 23</strong> (Integrity Constraint Violation)</td>
<td></td>
</tr>
<tr>
<td>23000</td>
<td>Integrity constraint violation</td>
</tr>
<tr>
<td><strong>SQLCLASS 24</strong> (Invalid Cursor State)</td>
<td></td>
</tr>
<tr>
<td>24000</td>
<td>Invalid cursor state</td>
</tr>
<tr>
<td>24504</td>
<td>The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row</td>
</tr>
<tr>
<td><strong>SQLCLASS 25</strong> (Invalid Transaction State)</td>
<td></td>
</tr>
<tr>
<td>25000</td>
<td>Invalid transaction state</td>
</tr>
<tr>
<td>25S01</td>
<td>Transaction state</td>
</tr>
<tr>
<td>25S02</td>
<td>Transaction is still active</td>
</tr>
<tr>
<td>25S03</td>
<td>Transaction is rolled back</td>
</tr>
<tr>
<td><strong>SQLCLASS 26</strong> (Invalid SQL Statement Name)</td>
<td></td>
</tr>
<tr>
<td>26000</td>
<td>Invalid SQL statement name</td>
</tr>
<tr>
<td><strong>SQLCLASS 27</strong> (Triggered Data Change Violation)</td>
<td></td>
</tr>
<tr>
<td>27000</td>
<td>Triggered data change violation</td>
</tr>
<tr>
<td><strong>SQLCLASS 28</strong> (Invalid Authorization Specification)</td>
<td></td>
</tr>
<tr>
<td>28000</td>
<td>Invalid authorization specification</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Связанное сообщение</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLCLASS 2B</td>
<td>(Dependent Privilege Descriptors Still Exist)</td>
</tr>
<tr>
<td>2B000</td>
<td>Dependent privilege descriptors still exist</td>
</tr>
<tr>
<td>SQLCLASS 2C</td>
<td>(Invalid Character Set Name)</td>
</tr>
<tr>
<td>2C000</td>
<td>Invalid character set name</td>
</tr>
<tr>
<td>SQLCLASS 2D</td>
<td>(Invalid Transaction Termination)</td>
</tr>
<tr>
<td>2D000</td>
<td>Invalid transaction termination</td>
</tr>
<tr>
<td>SQLCLASS 2E</td>
<td>(Invalid Connection Name)</td>
</tr>
<tr>
<td>2E000</td>
<td>Invalid connection name</td>
</tr>
<tr>
<td>SQLCLASS 2F</td>
<td>(SQL Routine Exception)</td>
</tr>
<tr>
<td>2F000</td>
<td>SQL routine exception</td>
</tr>
<tr>
<td>2F002</td>
<td>Modifying SQL-data not permitted</td>
</tr>
<tr>
<td>2F003</td>
<td>Prohibited SQL-statement attempted</td>
</tr>
<tr>
<td>2F004</td>
<td>Reading SQL-data not permitted</td>
</tr>
<tr>
<td>2F005</td>
<td>Function executed no return statement</td>
</tr>
<tr>
<td>SQLCLASS 33</td>
<td>(Invalid SQL Descriptor Name)</td>
</tr>
<tr>
<td>33000</td>
<td>Invalid SQL descriptor name</td>
</tr>
<tr>
<td>SQLCLASS 34</td>
<td>(Invalid Cursor Name)</td>
</tr>
<tr>
<td>34000</td>
<td>Invalid cursor name</td>
</tr>
<tr>
<td>SQLCLASS 35</td>
<td>(Invalid Condition Number)</td>
</tr>
<tr>
<td>35000</td>
<td>Invalid condition number</td>
</tr>
<tr>
<td>SQLCLASS 36</td>
<td>(Cursor Sensitivity Exception)</td>
</tr>
<tr>
<td>36001</td>
<td>Request rejected</td>
</tr>
<tr>
<td>36002</td>
<td>Request failed</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Связанное сообщение</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>SQLCLASS 37 (Invalid Identifier)</td>
<td>Класс ошибок 37 (неверный идентификатор)</td>
</tr>
<tr>
<td>37000</td>
<td>Invalid identifier</td>
</tr>
<tr>
<td>37001</td>
<td>Identifier too long</td>
</tr>
<tr>
<td>SQLCLASS 38 (External Routine Exception)</td>
<td>Класс ошибок 38 (ошибки внешних процедур)</td>
</tr>
<tr>
<td>38000</td>
<td>External routine exception</td>
</tr>
<tr>
<td>SQLCLASS 39 (External Routine Invocation Exception)</td>
<td>Класс ошибок 39 (ошибка вызова внешней процедуры)</td>
</tr>
<tr>
<td>39000</td>
<td>External routine invocation exception</td>
</tr>
<tr>
<td>SQLCLASS 3B (Invalid Save Point)</td>
<td>Класс ошибок 3B (неверная точка сохранения)</td>
</tr>
<tr>
<td>3B000</td>
<td>Invalid save point</td>
</tr>
<tr>
<td>SQLCLASS 3C (Ambiguous Cursor Name)</td>
<td>Класс ошибок 3C (имя курсора неоднозначное)</td>
</tr>
<tr>
<td>3C000</td>
<td>Ambiguous cursor name</td>
</tr>
<tr>
<td>SQLCLASS 3D (Invalid Catalog Name)</td>
<td>Класс ошибок 3D (неверное имя каталога)</td>
</tr>
<tr>
<td>3D000</td>
<td>Invalid catalog name</td>
</tr>
<tr>
<td>3D001</td>
<td>Catalog name not found</td>
</tr>
<tr>
<td>SQLCLASS 3F (Invalid Schema Name)</td>
<td>Класс ошибок 3F (неверное имя схемы)</td>
</tr>
<tr>
<td>3F000</td>
<td>Invalid schema name</td>
</tr>
<tr>
<td>SQLCLASS 40 (Transaction Rollback)</td>
<td>Класс ошибок 40 (откат транзакции)</td>
</tr>
<tr>
<td>40000</td>
<td>Ongoing transaction has been rolled back</td>
</tr>
<tr>
<td>40001</td>
<td>Serialization failure</td>
</tr>
<tr>
<td>40002</td>
<td>Transaction integrity constraint violation</td>
</tr>
<tr>
<td>40003</td>
<td>Statement completion unknown</td>
</tr>
<tr>
<td>SQLCLASS 42 (Syntax Error or Access Violation)</td>
<td>Класс ошибок 42 (синтаксическая ошибка или ошибка доступа)</td>
</tr>
<tr>
<td>42000</td>
<td>Syntax error or access violation</td>
</tr>
<tr>
<td>42702</td>
<td>Ambiguous column reference</td>
</tr>
<tr>
<td>42725</td>
<td>Ambiguous function reference</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>связанные сообщения</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>42818</td>
<td>the operands of an operator or function are not compatible</td>
</tr>
<tr>
<td>42S01</td>
<td>base table or view already exists</td>
</tr>
<tr>
<td>42S02</td>
<td>base table or view not found</td>
</tr>
<tr>
<td>42S11</td>
<td>index already exists</td>
</tr>
<tr>
<td>42S12</td>
<td>index not found</td>
</tr>
<tr>
<td>42S21</td>
<td>column already exists</td>
</tr>
<tr>
<td>42S22</td>
<td>column not found</td>
</tr>
<tr>
<td>SQLCLASS 44 (With Check Option Violation)</td>
<td>класс ошибок 44 (нарушение опции WITH CHECK)</td>
</tr>
<tr>
<td>44000</td>
<td>WITH CHECK OPTION Violation</td>
</tr>
<tr>
<td>SQLCLASS 45 (Unhandled User-defined Exception)</td>
<td>класс ошибок 45 (необработанное исключение определенное пользователем)</td>
</tr>
<tr>
<td>45000</td>
<td>unhandled user-defined exception</td>
</tr>
<tr>
<td>SQLCLASS 54 (Program Limit Exceeded)</td>
<td>класс ошибок 54 (превышены ограничения программы)</td>
</tr>
<tr>
<td>54000</td>
<td>program limit exceeded</td>
</tr>
<tr>
<td>54001</td>
<td>statement too complex</td>
</tr>
<tr>
<td>54011</td>
<td>too many columns</td>
</tr>
<tr>
<td>54023</td>
<td>too many arguments</td>
</tr>
<tr>
<td>SQLCLASS HY (CLI-specific Condition)</td>
<td>класс ошибок HY (условия CLI-specific)</td>
</tr>
<tr>
<td>HY000</td>
<td>CLI-specific condition</td>
</tr>
<tr>
<td>HY001</td>
<td>memory allocation error</td>
</tr>
<tr>
<td>HY003</td>
<td>invalid data type in application descriptor</td>
</tr>
<tr>
<td>HY004</td>
<td>invalid data type</td>
</tr>
<tr>
<td>HY007</td>
<td>associated statement is not prepared</td>
</tr>
<tr>
<td>HY008</td>
<td>operation canceled</td>
</tr>
<tr>
<td>HY009</td>
<td>invalid use of null pointer</td>
</tr>
<tr>
<td>HY010</td>
<td>function sequence error</td>
</tr>
<tr>
<td>HY011</td>
<td>attribute cannot be set now</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Связанное сообщение</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>HY012</td>
<td>Invalid transaction operation code</td>
</tr>
<tr>
<td>HY013</td>
<td>Memory management error</td>
</tr>
<tr>
<td>HY014</td>
<td>Limit on the number of handles exceeded</td>
</tr>
<tr>
<td>HY015</td>
<td>No cursor name available</td>
</tr>
<tr>
<td>HY016</td>
<td>Cannot modify an implementation row descriptor</td>
</tr>
<tr>
<td>HY017</td>
<td>Invalid use of an automatically allocated descriptor handle</td>
</tr>
<tr>
<td>HY018</td>
<td>Server declined the cancellation request</td>
</tr>
<tr>
<td>HY019</td>
<td>Non-string data cannot be sent in pieces</td>
</tr>
<tr>
<td>HY020</td>
<td>Attempt to concatenate a null value</td>
</tr>
<tr>
<td>HY021</td>
<td>Inconsistent descriptor information</td>
</tr>
<tr>
<td>HY024</td>
<td>Invalid attribute value</td>
</tr>
<tr>
<td>HY055</td>
<td>Non-string data cannot be used with string routine</td>
</tr>
<tr>
<td>HY090</td>
<td>Invalid string length or buffer length</td>
</tr>
<tr>
<td>HY091</td>
<td>Invalid descriptor field identifier</td>
</tr>
<tr>
<td>HY092</td>
<td>Invalid attribute identifier</td>
</tr>
<tr>
<td>HY095</td>
<td>Invalid FunctionId specified</td>
</tr>
<tr>
<td>HY096</td>
<td>Invalid information type</td>
</tr>
<tr>
<td>HY097</td>
<td>Column type out of range</td>
</tr>
<tr>
<td>HY098</td>
<td>Scope out of range</td>
</tr>
<tr>
<td>HY099</td>
<td>Nullable type out of range</td>
</tr>
<tr>
<td>HY100</td>
<td>Uniqueness option type out of range</td>
</tr>
<tr>
<td>HY101</td>
<td>Accuracy option type out of range</td>
</tr>
<tr>
<td>HY103</td>
<td>Invalid retrieval code</td>
</tr>
<tr>
<td>HY104</td>
<td>Invalid LengthPrecision value</td>
</tr>
<tr>
<td>HY105</td>
<td>Invalid parameter type</td>
</tr>
</tbody>
</table>
SQLSTATE Связанное сообщение Примечание
HY106 Invalid fetch orientation Неверное направление для fetch
HY107 Row value out of range Значение строки вне диапазона
HY109 Invalid cursor position Неверная позиция курсора
HY110 Invalid driver completion Неверный код завершения драйвера
HY111 Invalid bookmark value Неверное значение метки bookmark
HYC00 Optional feature not implemented Опциональная функция не реализована
HYT00 Timeout expired Достигнут тайм-аут
HYT01 Connection timeout expired Достигнут тайм-аут соединения

SQLCLASS XX (Internal Error) SQLCLASS XX (внутренние ошибки)
XX000 Internal error Внутренняя ошибка
XX001 Data corrupted Данные разрушены
XX002 Index corrupted Индекс разрушен

Коды ошибок GDSCODE их описание, и SQLCODE

Таблица ошибок содержит числовое и символьное значения GDSCODE, текст сообщения об ошибке и описание ошибки. Также приводится SQLCODE ошибки.

В настоящее время SQLCODE считаются устаревшим. В следующих версиях поддержка SQLCODE может полностью прекратиться.

Table 274. Коды ошибок GDSCODE, SQLCODE и их описание

<table>
<thead>
<tr>
<th>SQLCODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>501</td>
<td>335544802</td>
<td>dialect_reset_warning</td>
<td>Database dialect being changed from 3 to 1</td>
</tr>
<tr>
<td>304</td>
<td>335545266</td>
<td>truncate_warn</td>
<td>String truncated warning due to the following reason</td>
</tr>
<tr>
<td>304</td>
<td>335545267</td>
<td>truncate_monitor</td>
<td>Monitoring data does not fit into the field</td>
</tr>
<tr>
<td>304</td>
<td>335545268</td>
<td>truncate_context</td>
<td>Engine data does not fit into return value of system function</td>
</tr>
<tr>
<td>301</td>
<td>335544808</td>
<td>dtype_renamed</td>
<td>DATE data type is now called TIMESTAMP</td>
</tr>
<tr>
<td>301</td>
<td>336003076</td>
<td>dsq_dialect_warning_expr</td>
<td>Use of @1 expression that returns different results in dialect 1 and dialect 3</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>301</td>
<td>336003080</td>
<td>dsq1_warning_number_ambiguous</td>
<td>WARNING: Numeric literal @1 is interpreted as a floating-point value in SQL dialect 1, but as an exact numeric value in SQL dialect 3.</td>
</tr>
<tr>
<td>301</td>
<td>336003081</td>
<td>dsq1_warning_number_ambiguous1</td>
<td>WARNING: NUMERIC and DECIMAL fields with precision 10 or greater are stored as approximate floating-point values in SQL dialect 1, but as 64-bit integers in SQL dialect 3.</td>
</tr>
<tr>
<td>300</td>
<td>335544807</td>
<td>sqlwarn</td>
<td>SQL warning code = @1</td>
</tr>
<tr>
<td>106</td>
<td>336068855</td>
<td>dyn_miss_priv_warning</td>
<td>Warning: @1 on @2 is not granted to @3.</td>
</tr>
<tr>
<td>101</td>
<td>335544366</td>
<td>segment</td>
<td>segment buffer length shorter than expected</td>
</tr>
<tr>
<td>100</td>
<td>335544338</td>
<td>from_no_match</td>
<td>no match for first value expression</td>
</tr>
<tr>
<td>100</td>
<td>335544354</td>
<td>no_record</td>
<td>invalid database key</td>
</tr>
<tr>
<td>100</td>
<td>335544367</td>
<td>segstr_eof</td>
<td>attempted retrieval of more segments than exist</td>
</tr>
<tr>
<td>0</td>
<td>335544875</td>
<td>bad_debug_format</td>
<td>Bad debug info format</td>
</tr>
<tr>
<td>0</td>
<td>335544931</td>
<td>montabexh</td>
<td>Monitoring table space exhausted</td>
</tr>
<tr>
<td>-84</td>
<td>335544554</td>
<td>nonsql_security_rel</td>
<td>object has non-SQL security class defined</td>
</tr>
<tr>
<td>-84</td>
<td>335544555</td>
<td>nonsql_security_fld</td>
<td>column has non-SQL security class defined</td>
</tr>
<tr>
<td>-84</td>
<td>335544668</td>
<td>dsq1_procedure_use_err</td>
<td>procedure @1 does not return any values</td>
</tr>
<tr>
<td>-85</td>
<td>335544747</td>
<td>username_too_long</td>
<td>The username entered is too long. Maximum length is 31 bytes.</td>
</tr>
<tr>
<td>-85</td>
<td>335544748</td>
<td>password_too_long</td>
<td>The password specified is too long. Maximum length is 8 bytes.</td>
</tr>
<tr>
<td>-85</td>
<td>335544749</td>
<td>username_required</td>
<td>A username is required for this operation.</td>
</tr>
<tr>
<td>-85</td>
<td>335544750</td>
<td>password_required</td>
<td>A password is required for this operation</td>
</tr>
<tr>
<td>-85</td>
<td>335544751</td>
<td>bad_protocol</td>
<td>The network protocol specified is invalid</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-85</td>
<td>335544752</td>
<td>dup_usname_found</td>
<td>A duplicate user name was found in the security database</td>
</tr>
<tr>
<td>-85</td>
<td>335544753</td>
<td>usname_not_found</td>
<td>The user name specified was not found in the security database</td>
</tr>
<tr>
<td>-85</td>
<td>335544754</td>
<td>error_adding_sec_record</td>
<td>An error occurred while attempting to add the user.</td>
</tr>
<tr>
<td>-85</td>
<td>335544755</td>
<td>error_modifying_sec_record</td>
<td>An error occurred while attempting to modify the user record.</td>
</tr>
<tr>
<td>-85</td>
<td>335544756</td>
<td>error_deleting_sec_record</td>
<td>An error occurred while attempting to delete the user record.</td>
</tr>
<tr>
<td>-85</td>
<td>335544757</td>
<td>error_updating_sec_db</td>
<td>An error occurred while updating the security database.</td>
</tr>
<tr>
<td>-103</td>
<td>335544571</td>
<td>dsq1_constant_err</td>
<td>Data type for constant unknown</td>
</tr>
<tr>
<td>-104</td>
<td>335544343</td>
<td>invalid_blr</td>
<td>invalid request BLR at offset @1</td>
</tr>
<tr>
<td>-104</td>
<td>335544390</td>
<td>syntaxerr</td>
<td>BLR syntax error: expected @1 at offset @2, encountered @3</td>
</tr>
<tr>
<td>-104</td>
<td>335544425</td>
<td>ctxinuse</td>
<td>context already in use (BLR error)</td>
</tr>
<tr>
<td>-104</td>
<td>335544426</td>
<td>ctxnotdef</td>
<td>context not defined (BLR error)</td>
</tr>
<tr>
<td>-104</td>
<td>335544429</td>
<td>badparnum</td>
<td>undefined parameter number</td>
</tr>
<tr>
<td>-104</td>
<td>335544440</td>
<td>bad_msg_vec</td>
<td></td>
</tr>
<tr>
<td>-104</td>
<td>335544456</td>
<td>invalid_sdl</td>
<td>invalid slice description language at offset @1</td>
</tr>
<tr>
<td>-104</td>
<td>335544460</td>
<td>dsq1_command_err</td>
<td>Invalid command</td>
</tr>
<tr>
<td>-104</td>
<td>335544459</td>
<td>dsq1_internal_err</td>
<td>Internal error</td>
</tr>
<tr>
<td>-104</td>
<td>335544458</td>
<td>dsq1_dup_option</td>
<td>Option specified more than once</td>
</tr>
<tr>
<td>-104</td>
<td>335544459</td>
<td>dsq1_tran_err</td>
<td>Unknown transaction option</td>
</tr>
<tr>
<td>-104</td>
<td>335544460</td>
<td>dsq1_invalid_array</td>
<td>Invalid array reference</td>
</tr>
<tr>
<td>-104</td>
<td>335544461</td>
<td>command_end_err</td>
<td>Unexpected end of command</td>
</tr>
<tr>
<td>-104</td>
<td>335544462</td>
<td>token_err</td>
<td>Token unknown</td>
</tr>
<tr>
<td>-104</td>
<td>335544463</td>
<td>dsq1_token_unk_err</td>
<td>Token unknown - line @1, column @2</td>
</tr>
<tr>
<td>-104</td>
<td>335544475</td>
<td>dsq1_agg_ref_err</td>
<td>Invalid aggregate reference</td>
</tr>
<tr>
<td>-104</td>
<td>335544476</td>
<td>invalid_array_id</td>
<td>invalid blob id</td>
</tr>
<tr>
<td>-104</td>
<td>335544477</td>
<td>cse_not_supported</td>
<td>Client/Server Express not supported in this release</td>
</tr>
<tr>
<td>-104</td>
<td>335544478</td>
<td>token_too_long</td>
<td>token size exceeds limit</td>
</tr>
<tr>
<td>SQL- CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-----------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>-104</td>
<td>335544763</td>
<td>invalid_string_constant</td>
<td>a string constant is delimited by double quotes</td>
</tr>
<tr>
<td>-104</td>
<td>335544764</td>
<td>transitional_date</td>
<td>DATE must be changed to TIMESTAMP</td>
</tr>
<tr>
<td>-104</td>
<td>335544796</td>
<td>sql_dialect_datatype_unsupport</td>
<td>Client SQL dialect @1 does not support reference to @2 datatype</td>
</tr>
<tr>
<td>-104</td>
<td>335544798</td>
<td>depend_on_uncommitted_rel</td>
<td>You created an indirect dependency on uncommitted metadata. You must roll back the current transaction.</td>
</tr>
<tr>
<td>-104</td>
<td>335544821</td>
<td>dsqll_column_pos_err</td>
<td>Invalid column position used in the @1 clause</td>
</tr>
<tr>
<td>-104</td>
<td>335544822</td>
<td>dsqll_agg_where_err</td>
<td>Cannot use an aggregate or window function in a WHERE clause, use HAVING (for aggregate only) instead</td>
</tr>
<tr>
<td>-104</td>
<td>335544823</td>
<td>dsqll_agg_group_err</td>
<td>Cannot use an aggregate or window function in a GROUP BY clause</td>
</tr>
<tr>
<td>-104</td>
<td>335544824</td>
<td>dsqll_agg_column_err</td>
<td>Invalid expression in the @1 (not contained in either an aggregate function or the GROUP BY clause)</td>
</tr>
<tr>
<td>-104</td>
<td>335544825</td>
<td>dsqll_agg_having_err</td>
<td>Invalid expression in the @1 (neither an aggregate function nor a part of the GROUP BY clause)</td>
</tr>
<tr>
<td>-104</td>
<td>335544826</td>
<td>dsqll_agg_nested_err</td>
<td>Nested aggregate and window functions are not allowed</td>
</tr>
<tr>
<td>-104</td>
<td>335544849</td>
<td>malformed_string</td>
<td>Malformed string</td>
</tr>
<tr>
<td>-104</td>
<td>335544851</td>
<td>command_end_err2</td>
<td>Unexpected end of command - line @1, column @2</td>
</tr>
<tr>
<td>-104</td>
<td>335544930</td>
<td>too_big_blr</td>
<td>BLR stream length @1 exceeds implementation limit @2</td>
</tr>
<tr>
<td>-104</td>
<td>335544980</td>
<td>internal_rejected_params</td>
<td>Incorrect parameters provided to internal function @1</td>
</tr>
<tr>
<td>-104</td>
<td>335545022</td>
<td>cannot_copy_stmt</td>
<td>Cannot copy statement @1</td>
</tr>
<tr>
<td>-104</td>
<td>335545023</td>
<td>invalid_boolean_usage</td>
<td>Invalid usage of boolean expression</td>
</tr>
<tr>
<td>-104</td>
<td>335545035</td>
<td>svc_no_stdin</td>
<td>No isc_info_svc_stdin in user request, but service thread requested stdin data</td>
</tr>
<tr>
<td>-104</td>
<td>335545037</td>
<td>svc_no_switches</td>
<td>All services except for getting server log require switches</td>
</tr>
<tr>
<td>-104</td>
<td>335545038</td>
<td>svc_bad_size</td>
<td>Size of stdin data is more than was requested from client</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-104</td>
<td>335545039</td>
<td>no_crypt_plugin</td>
<td>Crypt plugin @1 failed to load</td>
</tr>
<tr>
<td>-104</td>
<td>335545040</td>
<td>cp_name_too_long</td>
<td>Length of crypt plugin name should not exceed @1 bytes</td>
</tr>
<tr>
<td>-104</td>
<td>335545045</td>
<td>null_spb</td>
<td>NULL data with non-zero SPB length</td>
</tr>
<tr>
<td>-104</td>
<td>335545116</td>
<td>dsq_window_incompat_frames</td>
<td>If &lt;window frame bound 1&gt; specifies @1, then &lt;window frame bound 2&gt; shall not specify @2</td>
</tr>
<tr>
<td>-104</td>
<td>335545117</td>
<td>dsq_window_range_multi_key</td>
<td>RANGE based window with &lt;expr&gt; {PRECEDING</td>
</tr>
<tr>
<td>-104</td>
<td>335545118</td>
<td>dsq_window_range_inv_key_type</td>
<td>RANGE based window must have an ORDER BY key of numerical, date, time or timestamp types</td>
</tr>
<tr>
<td>-104</td>
<td>335545119</td>
<td>dsq_window_frame_value_inv_type</td>
<td>Window RANGE/ROWS PRECEDING/FOLLOWING value must be of a numerical type</td>
</tr>
<tr>
<td>-104</td>
<td>335545205</td>
<td>no_keyholder_plugin</td>
<td>Key holder plugin @1 failed to load</td>
</tr>
<tr>
<td>-104</td>
<td>336003075</td>
<td>dsq_transitional_numeric</td>
<td>Precision 10 to 18 changed from DOUBLE PRECISION in SQL dialect 1 to 64-bit scaled integer in SQL dialect 3</td>
</tr>
<tr>
<td>-104</td>
<td>336003077</td>
<td>sql_db_dialect_dtype_unsupport</td>
<td>Database SQL dialect @1 does not support reference to @2 datatype</td>
</tr>
<tr>
<td>-104</td>
<td>336003087</td>
<td>dsq_invalid_label</td>
<td>Label @1 @2 in the current scope</td>
</tr>
<tr>
<td>-104</td>
<td>336003088</td>
<td>dsq_datatypes_notComparable</td>
<td>Datatypes @1 are not comparable in expression @2</td>
</tr>
<tr>
<td>-104</td>
<td>336397215</td>
<td>dsq_max_sort_items</td>
<td>cannot sort on more than 255 items</td>
</tr>
<tr>
<td>-104</td>
<td>336397216</td>
<td>dsq_max_group_items</td>
<td>cannot group on more than 255 items</td>
</tr>
<tr>
<td>-104</td>
<td>336397217</td>
<td>dsq_conflicting_sort_field</td>
<td>Cannot include the same field (@1.@2) twice in the ORDER BY clause with conflicting sorting options</td>
</tr>
<tr>
<td>-104</td>
<td>336397218</td>
<td>dsq_derived_table_more_columns</td>
<td>column list from derived table @1 has more columns than the number of items in its SELECT statement</td>
</tr>
<tr>
<td>-104</td>
<td>336397219</td>
<td>dsq_derived_table_less_columns</td>
<td>column list from derived table @1 has less columns than the number of items in its SELECT statement</td>
</tr>
</tbody>
</table>

Appendix B: Обработка ошибок, коды и сообщения
<table>
<thead>
<tr>
<th>SQL-CODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>-104</td>
<td>336397220</td>
<td>dsqld-derived-field_unnamed</td>
<td>no column name specified for column number @1 in derived table @2</td>
</tr>
<tr>
<td>-104</td>
<td>336397221</td>
<td>dsqld-derived-field_dup_name</td>
<td>column @1 was specified multiple times for derived table @2</td>
</tr>
<tr>
<td>-104</td>
<td>336397222</td>
<td>dsqld-derived-alias_select</td>
<td>Internal dsqld error: alias type expected by pass1-expand-select-node</td>
</tr>
<tr>
<td>-104</td>
<td>336397223</td>
<td>dsqld-derived-alias_field</td>
<td>Internal dsqld error: alias type expected by pass1-field</td>
</tr>
<tr>
<td>-104</td>
<td>336397224</td>
<td>dsqld-auto-field_bad_pos</td>
<td>Internal dsqld error: column position out of range in pass1-union-auto_cast</td>
</tr>
<tr>
<td>-104</td>
<td>336397225</td>
<td>dsqld-cte-wrong-reference</td>
<td>Recursive CTE member (@1) can refer itself only in FROM clause</td>
</tr>
<tr>
<td>-104</td>
<td>336397226</td>
<td>dsqld-cte-cycle</td>
<td>CTE '@1' has cyclic dependencies</td>
</tr>
<tr>
<td>-104</td>
<td>336397227</td>
<td>dsqld-cte-outer-join</td>
<td>Recursive member of CTE can't be member of an outer join</td>
</tr>
<tr>
<td>-104</td>
<td>336397228</td>
<td>dsqld-cte-mult-references</td>
<td>Recursive member of CTE can't reference itself more than once</td>
</tr>
<tr>
<td>-104</td>
<td>336397229</td>
<td>dsqld-cte-not-a-union</td>
<td>Recursive CTE (@1) must be an UNION</td>
</tr>
<tr>
<td>-104</td>
<td>336397230</td>
<td>dsqld-cte-nonrecurs-after-recurs</td>
<td>CTE '@1' defined non-recursive member after recursive</td>
</tr>
<tr>
<td>-104</td>
<td>336397231</td>
<td>dsqld-cte-wrong-clause</td>
<td>Recursive member of CTE '@1' has @2 clause</td>
</tr>
<tr>
<td>-104</td>
<td>336397232</td>
<td>dsqld-cte-union-all</td>
<td>Recursive members of CTE (@1) must be linked with another members via UNION ALL</td>
</tr>
<tr>
<td>-104</td>
<td>336397233</td>
<td>dsqld-cte-miss-nonrecursive</td>
<td>Non-recursive member is missing in CTE '@1'</td>
</tr>
<tr>
<td>-104</td>
<td>336397234</td>
<td>dsqld-cte-nested-with</td>
<td>WITH clause can't be nested</td>
</tr>
<tr>
<td>-104</td>
<td>336397235</td>
<td>dsqld-col-more-than-once-using</td>
<td>column @1 appears more than once in USING clause</td>
</tr>
<tr>
<td>-104</td>
<td>336397237</td>
<td>dsqld-cte-not-used</td>
<td>CTE &quot;@1&quot; is not used in query</td>
</tr>
<tr>
<td>-104</td>
<td>336397238</td>
<td>dsqld-col-more-than-once-view</td>
<td>column @1 appears more than once in ALTER VIEW</td>
</tr>
<tr>
<td>-104</td>
<td>336397257</td>
<td>dsqld-max-distinct-items</td>
<td>Cannot have more than 255 items in DISTINCT / UNION DISTINCT list</td>
</tr>
<tr>
<td>-104</td>
<td>336397321</td>
<td>dsqld-cte-recursive-aggregate</td>
<td>Recursive member of CTE cannot use aggregate or window function</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-104</td>
<td>336397326</td>
<td>dsq1_wlock_simple</td>
<td>WITH LOCK can be used only with a single physical table</td>
</tr>
<tr>
<td>-104</td>
<td>336397327</td>
<td>dsq1_firstskip_rows</td>
<td>FIRST/SKIP cannot be used with OFFSET/FETCH or ROWS</td>
</tr>
<tr>
<td>-104</td>
<td>336397328</td>
<td>dsq1_wlock_aggregates</td>
<td>WITH LOCK cannot be used with aggregates</td>
</tr>
<tr>
<td>-104</td>
<td>336397329</td>
<td>dsq1_wlock_conflict</td>
<td>WITH LOCK cannot be used with @1</td>
</tr>
<tr>
<td>-105</td>
<td>335544702</td>
<td>escape_invalid</td>
<td>Invalid ESCAPE sequence</td>
</tr>
<tr>
<td>-105</td>
<td>335544789</td>
<td>extract_input_mismatch</td>
<td>Specified EXTRACT part does not exist in input datatype</td>
</tr>
<tr>
<td>-105</td>
<td>335544884</td>
<td>invalid_similar_pattern</td>
<td>Invalid SIMILAR TO pattern</td>
</tr>
<tr>
<td>-150</td>
<td>335544360</td>
<td>read_only_rel</td>
<td>attempted update of read-only table</td>
</tr>
<tr>
<td>-150</td>
<td>335544362</td>
<td>read_only_view</td>
<td>cannot update read-only view @1</td>
</tr>
<tr>
<td>-150</td>
<td>335544446</td>
<td>non_updatable</td>
<td>not updatable</td>
</tr>
<tr>
<td>-150</td>
<td>335544546</td>
<td>constaint_on_view</td>
<td>Cannot define constraints on views</td>
</tr>
<tr>
<td>-151</td>
<td>335544359</td>
<td>read_only_field</td>
<td>attempted update of read-only column @1</td>
</tr>
<tr>
<td>-155</td>
<td>335544658</td>
<td>dsq1_base_table</td>
<td>@1 is not a valid base table of the specified view</td>
</tr>
<tr>
<td>-157</td>
<td>335544598</td>
<td>specify_field_err</td>
<td>must specify column name for view select expression</td>
</tr>
<tr>
<td>-158</td>
<td>335544599</td>
<td>num_field_err</td>
<td>number of columns does not match select list</td>
</tr>
<tr>
<td>-162</td>
<td>335544685</td>
<td>no_dbkey</td>
<td>dbkey not available for multi-table views</td>
</tr>
<tr>
<td>-170</td>
<td>335544512</td>
<td>prcmismat</td>
<td>Input parameter mismatch for procedure @1</td>
</tr>
<tr>
<td>-170</td>
<td>335544619</td>
<td>extern_func_err</td>
<td>External functions cannot have more than 10 parameters</td>
</tr>
<tr>
<td>-170</td>
<td>335544850</td>
<td>prc_out_param_mismatch</td>
<td>Output parameter mismatch for procedure @1</td>
</tr>
<tr>
<td>-170</td>
<td>335545101</td>
<td>fun_param_mismatch</td>
<td>Input parameter mismatch for function @1</td>
</tr>
<tr>
<td>-171</td>
<td>335544439</td>
<td>funmismat</td>
<td>function @1 could not be matched</td>
</tr>
<tr>
<td>-171</td>
<td>335544458</td>
<td>invalid_dimension</td>
<td>column not array or invalid dimensions (expected @1, encountered @2)</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-171</td>
<td>335544618</td>
<td>return_mode_err</td>
<td>Return mode by value not allowed for this data type</td>
</tr>
<tr>
<td>-171</td>
<td>335544873</td>
<td>array_max_dimensions</td>
<td>Array data type can use up to @1 dimensions</td>
</tr>
<tr>
<td>-172</td>
<td>335544438</td>
<td>funnotdef</td>
<td>function @1 is not defined</td>
</tr>
<tr>
<td>-172</td>
<td>335544932</td>
<td>modnotfound</td>
<td>module name or entrypoint could not be found</td>
</tr>
<tr>
<td>-203</td>
<td>335544708</td>
<td>dyn_fld_ambiguous</td>
<td>Ambiguous column reference.</td>
</tr>
<tr>
<td>-204</td>
<td>335544463</td>
<td>gennotdef</td>
<td>generator @1 is not defined</td>
</tr>
<tr>
<td>-204</td>
<td>335544502</td>
<td>stream_not_defined</td>
<td>reference to invalid stream number</td>
</tr>
<tr>
<td>-204</td>
<td>335544509</td>
<td>charset_not_found</td>
<td>CHARACTER SET @1 is not defined</td>
</tr>
<tr>
<td>-204</td>
<td>335544511</td>
<td>prcnotdef</td>
<td>procedure @1 is not defined</td>
</tr>
<tr>
<td>-204</td>
<td>335544515</td>
<td>codnotdef</td>
<td>status code @1 unknown</td>
</tr>
<tr>
<td>-204</td>
<td>335544516</td>
<td>xcpnotdef</td>
<td>exception @1 not defined</td>
</tr>
<tr>
<td>-204</td>
<td>335544532</td>
<td>ref_cnstrnt_notfound</td>
<td>Name of Referential Constraint not defined in constraints table.</td>
</tr>
<tr>
<td>-204</td>
<td>335544551</td>
<td>grant_obj_notfound</td>
<td>could not find object for GRANT</td>
</tr>
<tr>
<td>-204</td>
<td>335544568</td>
<td>text_subtype</td>
<td>Implementation of text subtype @1 not located.</td>
</tr>
<tr>
<td>-204</td>
<td>335544573</td>
<td>dsqll_datatype_err</td>
<td>Data type unknown</td>
</tr>
<tr>
<td>-204</td>
<td>335544580</td>
<td>dsqll_relation_err</td>
<td>Table unknown</td>
</tr>
<tr>
<td>-204</td>
<td>335544581</td>
<td>dsqll_procedure_err</td>
<td>Procedure unknown</td>
</tr>
<tr>
<td>-204</td>
<td>335544588</td>
<td>collation_not_found</td>
<td>COLLATION @1 for CHARACTER SET @2 is not defined</td>
</tr>
<tr>
<td>-204</td>
<td>335544589</td>
<td>collation_not_for_charset</td>
<td>COLLATION @1 is not valid for specified CHARACTER SET</td>
</tr>
<tr>
<td>-204</td>
<td>335544595</td>
<td>dsqll_trigger_err</td>
<td>Trigger unknown</td>
</tr>
<tr>
<td>-204</td>
<td>335544620</td>
<td>alias_conflict_err</td>
<td>alias @1 conflicts with an alias in the same statement</td>
</tr>
<tr>
<td>-204</td>
<td>335544621</td>
<td>procedure_conflict_error</td>
<td>alias @1 conflicts with a procedure in the same statement</td>
</tr>
<tr>
<td>-204</td>
<td>335544622</td>
<td>relation_conflict_err</td>
<td>alias @1 conflicts with a table in the same statement</td>
</tr>
<tr>
<td>-204</td>
<td>335544635</td>
<td>dsqll_no_relation_alias</td>
<td>there is no alias or table named @1 at this scope level</td>
</tr>
<tr>
<td>-204</td>
<td>335544636</td>
<td>indexname</td>
<td>there is no index @1 for table @2</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-204</td>
<td>335544640</td>
<td>collation_requires_text</td>
<td>Invalid use of CHARACTER SET or COLLATE</td>
</tr>
<tr>
<td>-204</td>
<td>335544662</td>
<td>dsq_blob_type_unknown</td>
<td>BLOB SUB_TYPE @1 is not defined</td>
</tr>
<tr>
<td>-204</td>
<td>335544759</td>
<td>bad_default_value</td>
<td>can not define a not null column with NULL as default value</td>
</tr>
<tr>
<td>-204</td>
<td>335544760</td>
<td>invalid_clause</td>
<td>invalid clause --- ' @1'</td>
</tr>
<tr>
<td>-204</td>
<td>335544800</td>
<td>too_many_contexts</td>
<td>Too many Contexts of Relation/Procedure/Views. Maximum allowed is 256</td>
</tr>
<tr>
<td>-204</td>
<td>335544817</td>
<td>bad_limit_param</td>
<td>Invalid parameter to FETCH or FIRST. Only integers &gt;= 0 are allowed.</td>
</tr>
<tr>
<td>-204</td>
<td>335544818</td>
<td>bad_skip_param</td>
<td>Invalid parameter to OFFSET or SKIP. Only integers &gt;= 0 are allowed.</td>
</tr>
<tr>
<td>-204</td>
<td>335544837</td>
<td>bad_substring_offset</td>
<td>Invalid offset parameter @1 to SUBSTRING. Only positive integers are allowed.</td>
</tr>
<tr>
<td>-204</td>
<td>335544853</td>
<td>bad_substring_length</td>
<td>Invalid length parameter @1 to SUBSTRING. Negative integers are not allowed.</td>
</tr>
<tr>
<td>-204</td>
<td>335544854</td>
<td>charset_not_installed</td>
<td>CHARACTER SET @1 is not installed</td>
</tr>
<tr>
<td>-204</td>
<td>335544855</td>
<td>collation_not_installed</td>
<td>COLLATION @1 for CHARACTER SET @2 is not installed</td>
</tr>
<tr>
<td>-204</td>
<td>335544867</td>
<td>subtype_for_internal_use</td>
<td>Blob sub_types bigger than 1 (text) are for internal use only</td>
</tr>
<tr>
<td>-204</td>
<td>335545104</td>
<td>invalid_attachment_charset</td>
<td>CHARACTER SET @1 cannot be used as a attachment character set</td>
</tr>
<tr>
<td>-204</td>
<td>336003085</td>
<td>dsq_ambiguous_field_name</td>
<td>Ambiguous field name between @1 and @2</td>
</tr>
<tr>
<td>-205</td>
<td>335544396</td>
<td>fldnotdef</td>
<td>column @1 is not defined in table @2</td>
</tr>
<tr>
<td>-205</td>
<td>335544552</td>
<td>grant_fld_notfound</td>
<td>could not find column for GRANT</td>
</tr>
<tr>
<td>-205</td>
<td>335544883</td>
<td>fldnotdef2</td>
<td>column @1 is not defined in procedure @2</td>
</tr>
<tr>
<td>-206</td>
<td>335544578</td>
<td>dsq_field_err</td>
<td>Column unknown</td>
</tr>
<tr>
<td>-206</td>
<td>335544587</td>
<td>dsq_blob_err</td>
<td>Column is not a BLOB</td>
</tr>
<tr>
<td>-206</td>
<td>335544596</td>
<td>dsq_subselect_err</td>
<td>Subselect illegal in this context</td>
</tr>
<tr>
<td>-206</td>
<td>336397208</td>
<td>dsq_line_col_error</td>
<td>At line @1, column @2</td>
</tr>
<tr>
<td>-206</td>
<td>336397209</td>
<td>dsq_unknown_pos</td>
<td>At unknown line and column</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>-206</td>
<td>336397210</td>
<td>dsql_no_dup_name</td>
<td>Column @1 cannot be repeated in @2 statement</td>
</tr>
<tr>
<td>-208</td>
<td>335544617</td>
<td>order_by_err</td>
<td>invalid ORDER BY clause</td>
</tr>
<tr>
<td>-219</td>
<td>335544395</td>
<td>relnotdef</td>
<td>table @1 is not defined</td>
</tr>
<tr>
<td>-219</td>
<td>335544872</td>
<td>domnotdef</td>
<td>domain @1 is not defined</td>
</tr>
<tr>
<td>-230</td>
<td>335544487</td>
<td>walw_err</td>
<td>WAL Writer error</td>
</tr>
<tr>
<td>-231</td>
<td>335544488</td>
<td>logh_small</td>
<td>Log file header of @1 too small</td>
</tr>
<tr>
<td>-232</td>
<td>335544489</td>
<td>logh_inv_version</td>
<td>Invalid version of log file @1</td>
</tr>
<tr>
<td>-233</td>
<td>335544490</td>
<td>logh_open_flag</td>
<td>Log file @1 not latest in the chain but open flag still set</td>
</tr>
<tr>
<td>-234</td>
<td>335544491</td>
<td>logh_open_flag2</td>
<td>Log file @1 not closed properly; database recovery may be required</td>
</tr>
<tr>
<td>-235</td>
<td>335544492</td>
<td>logh_diff dbname</td>
<td>Database name in the log file @1 is different</td>
</tr>
<tr>
<td>-236</td>
<td>335544493</td>
<td>logf_unexpected_eof</td>
<td>Unexpected end of log file @1 at offset @2</td>
</tr>
<tr>
<td>-237</td>
<td>335544494</td>
<td>logr_incomplete</td>
<td>Incomplete log record at offset @1 in log file @2</td>
</tr>
<tr>
<td>-238</td>
<td>335544495</td>
<td>logr_header_small</td>
<td>Log record header too small at offset @1 in log file @2</td>
</tr>
<tr>
<td>-239</td>
<td>335544496</td>
<td>logb_small</td>
<td>Log block too small at offset @1 in log file @2</td>
</tr>
<tr>
<td>-239</td>
<td>335544691</td>
<td>cache_too_small</td>
<td>Insufficient memory to allocate page buffer cache</td>
</tr>
<tr>
<td>-239</td>
<td>335544693</td>
<td>logr_too_small</td>
<td>Log size too small</td>
</tr>
<tr>
<td>-239</td>
<td>335544694</td>
<td>partition_too_small</td>
<td>Log partition size too small</td>
</tr>
<tr>
<td>-240</td>
<td>335544497</td>
<td>wal_illegal_attach</td>
<td>Illegal attempt to attach to an uninitialized WAL segment for @1</td>
</tr>
<tr>
<td>-241</td>
<td>335544498</td>
<td>wal_invalid_wpb</td>
<td>Invalid WAL parameter block option @1</td>
</tr>
<tr>
<td>-242</td>
<td>335544499</td>
<td>wal_err_rollover</td>
<td>Cannot roll over to the next log file @1</td>
</tr>
<tr>
<td>-243</td>
<td>335544500</td>
<td>no_wal</td>
<td>database does not use Write-ahead Log</td>
</tr>
<tr>
<td>-244</td>
<td>335544503</td>
<td>wal_subsys_error</td>
<td>WAL subsystem encountered error</td>
</tr>
<tr>
<td>-245</td>
<td>335544504</td>
<td>wal_subsys_corrupt</td>
<td>WAL subsystem corrupted</td>
</tr>
<tr>
<td>-246</td>
<td>335544513</td>
<td>wal_bugcheck</td>
<td>Database @1: WAL subsystem bug for pid @2 @3</td>
</tr>
<tr>
<td>SQL-Code</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>-247</td>
<td>335544514</td>
<td>wal_cant_expand</td>
<td>Could not expand the WAL segment for database @1</td>
</tr>
<tr>
<td>-248</td>
<td>335544521</td>
<td>wal_err_rollover2</td>
<td>Unable to roll over please see Firebird log.</td>
</tr>
<tr>
<td>-249</td>
<td>335544522</td>
<td>wal_err_logwrite</td>
<td>WAL I/O error. Please see Firebird log.</td>
</tr>
<tr>
<td>-250</td>
<td>335544523</td>
<td>wal_err_jrn_comm</td>
<td>WAL writer - Journal server communication error. Please see Firebird log.</td>
</tr>
<tr>
<td>-251</td>
<td>335544524</td>
<td>wal_err_expansion</td>
<td>WAL buffers cannot be increased. Please see Firebird log.</td>
</tr>
<tr>
<td>-252</td>
<td>335544525</td>
<td>wal_err_setup</td>
<td>WAL setup error. Please see Firebird log.</td>
</tr>
<tr>
<td>-253</td>
<td>335544526</td>
<td>wal_err_ww_sync</td>
<td>obsolete</td>
</tr>
<tr>
<td>-254</td>
<td>335544527</td>
<td>wal_err_ww_start</td>
<td>Cannot start WAL writer for the database @1</td>
</tr>
<tr>
<td>-255</td>
<td>335544556</td>
<td>wal_cache_err</td>
<td>Write-ahead Log without shared cache configuration not allowed</td>
</tr>
<tr>
<td>-257</td>
<td>335544566</td>
<td>start_cm_for_wal</td>
<td>WAL defined; Cache Manager must be started first</td>
</tr>
<tr>
<td>-258</td>
<td>335544567</td>
<td>wal_ovflow_log_required</td>
<td>Overflow log specification required for round-robin log</td>
</tr>
<tr>
<td>-259</td>
<td>335544629</td>
<td>wal_shadow_err</td>
<td>Write-ahead Log with shadowing configuration not allowed</td>
</tr>
<tr>
<td>-260</td>
<td>335544690</td>
<td>cache_redef</td>
<td>Cache redefined</td>
</tr>
<tr>
<td>-260</td>
<td>335544692</td>
<td>log_redef</td>
<td>Log redefined</td>
</tr>
<tr>
<td>-261</td>
<td>335544695</td>
<td>partition_not_supp</td>
<td>Partitions not supported in series of log file specification</td>
</tr>
<tr>
<td>-261</td>
<td>335544696</td>
<td>log_length_spec</td>
<td>Total length of a partitioned log must be specified</td>
</tr>
<tr>
<td>-281</td>
<td>335544637</td>
<td>no_stream_plan</td>
<td>table @1 is not referenced in plan</td>
</tr>
<tr>
<td>-282</td>
<td>335544638</td>
<td>stream_twice</td>
<td>table @1 is referenced more than once in plan; use aliases to distinguish</td>
</tr>
<tr>
<td>-282</td>
<td>335544643</td>
<td>dsqsl_self_join</td>
<td>the table @1 is referenced twice; use aliases to differentiate</td>
</tr>
<tr>
<td>-282</td>
<td>335544659</td>
<td>duplicate_base_table</td>
<td>table @1 is referenced twice in view; use an alias to distinguish</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-282</td>
<td>335544660</td>
<td>view_alias</td>
<td>view @1 has more than one base table; use aliases to distinguish</td>
</tr>
<tr>
<td>-282</td>
<td>335544710</td>
<td>complex_view</td>
<td>navigational stream @1 references a view with more than one base table</td>
</tr>
<tr>
<td>-283</td>
<td>335544639</td>
<td>stream_not_found</td>
<td>table @1 is referenced in the plan but not the from list</td>
</tr>
<tr>
<td>-284</td>
<td>335544642</td>
<td>index_unused</td>
<td>index @1 cannot be used in the specified plan</td>
</tr>
<tr>
<td>-291</td>
<td>335544531</td>
<td>primary_key_notnull</td>
<td>Column used in a PRIMARY constraint must be NOT NULL.</td>
</tr>
<tr>
<td>-291</td>
<td>335545103</td>
<td>domain_primary_key_notnull</td>
<td>Domain used in the PRIMARY KEY constraint of table @1 must be NOT NULL.</td>
</tr>
<tr>
<td>-292</td>
<td>335544534</td>
<td>ref_cnstrnt_update</td>
<td>Cannot update constraints (RDB$REF_CONSTRAINTS).</td>
</tr>
<tr>
<td>-293</td>
<td>335544535</td>
<td>check_cnstrnt_update</td>
<td>Cannot update constraints (RDB$CHECK_CONSTRAINTS).</td>
</tr>
<tr>
<td>-294</td>
<td>335544536</td>
<td>check_cnstrnt_del</td>
<td>Cannot delete CHECK constraint entry (RDB$CHECK_CONSTRAINTS)</td>
</tr>
<tr>
<td>-295</td>
<td>335544545</td>
<td>rel_cnstrnt_update</td>
<td>Cannot update constraints (RDB$RELATION_CONSTRAINTS).</td>
</tr>
<tr>
<td>-296</td>
<td>335544547</td>
<td>invld_cnstrnt_type</td>
<td>internal Firebird consistency check (invalid RDB$CONSTRAINT_TYPE)</td>
</tr>
<tr>
<td>-297</td>
<td>335544558</td>
<td>check_constraint</td>
<td>Operation violates CHECK constraint @1 on view or table @2</td>
</tr>
<tr>
<td>-313</td>
<td>335544669</td>
<td>dsql_count_mismatch</td>
<td>count of column list and variable list do not match</td>
</tr>
<tr>
<td>-313</td>
<td>336003099</td>
<td>upd_ins_doesnt_match_pk</td>
<td>UPDATE OR INSERT field list does not match primary key of table @1</td>
</tr>
<tr>
<td>-313</td>
<td>336003100</td>
<td>upd_ins_doesnt_match_matching</td>
<td>UPDATE OR INSERT field list does not match MATCHING clause</td>
</tr>
<tr>
<td>-313</td>
<td>336003111</td>
<td>dsql_wrong_param_num</td>
<td>Wrong number of parameters (expected @1, got @2)</td>
</tr>
<tr>
<td>-313</td>
<td>336003113</td>
<td>upd_ins_cannot_default</td>
<td>UPDATE OR INSERT value for field @1, part of the implicit or explicit MATCHING clause, cannot be DEFAULT</td>
</tr>
<tr>
<td>-314</td>
<td>335544565</td>
<td>transliteration_failed</td>
<td>Cannot transliterate character between character sets</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-315</td>
<td>336068815</td>
<td>dyn_dtype_invalid</td>
<td>Cannot change datatype for column @1. Changing datatype is not supported for BLOB or ARRAY columns.</td>
</tr>
<tr>
<td>-383</td>
<td>336068814</td>
<td>dyn_dependency_exists</td>
<td>Column @1 from table @2 is referenced in @3</td>
</tr>
<tr>
<td>-401</td>
<td>335544647</td>
<td>invalid_operator</td>
<td>invalid comparison operator for find operation</td>
</tr>
<tr>
<td>-402</td>
<td>335544368</td>
<td>segstr_no_op</td>
<td>attempted invalid operation on a BLOB</td>
</tr>
<tr>
<td>-402</td>
<td>335544414</td>
<td>blobnotsup</td>
<td>BLOB and array data types are not supported for @1 operation</td>
</tr>
<tr>
<td>-402</td>
<td>335544427</td>
<td>datnotsup</td>
<td>data operation not supported</td>
</tr>
<tr>
<td>-402</td>
<td>335545262</td>
<td>cannot_update_old_blob</td>
<td>cannot update old BLOB</td>
</tr>
<tr>
<td>-402</td>
<td>335545263</td>
<td>cannot_read_new_blob</td>
<td>cannot read from new BLOB</td>
</tr>
<tr>
<td>-406</td>
<td>335544457</td>
<td>out_of_bounds</td>
<td>subscript out of bounds</td>
</tr>
<tr>
<td>-406</td>
<td>335545028</td>
<td>ss_out_of_bounds</td>
<td>Subscript @1 out of bounds [@2, @3]</td>
</tr>
<tr>
<td>-407</td>
<td>335544435</td>
<td>nullsegkey</td>
<td>null segment of UNIQUE KEY</td>
</tr>
<tr>
<td>-413</td>
<td>335544334</td>
<td>convert_error</td>
<td>conversion error from string &quot;@1&quot;</td>
</tr>
<tr>
<td>-413</td>
<td>335544454</td>
<td>nofilter</td>
<td>filter not found to convert type @1 to type @2</td>
</tr>
<tr>
<td>-413</td>
<td>335544860</td>
<td>blob_convert_error</td>
<td>Unsupported conversion to target type BLOB (subtype @1)</td>
</tr>
<tr>
<td>-413</td>
<td>335544861</td>
<td>array_convert_error</td>
<td>Unsupported conversion to target type ARRAY</td>
</tr>
<tr>
<td>-501</td>
<td>335544577</td>
<td>dsq1_cursor_close_err</td>
<td>Attempt to reclose a closed cursor</td>
</tr>
<tr>
<td>-502</td>
<td>335544574</td>
<td>dsq1_decl_err</td>
<td>Invalid cursor declaration</td>
</tr>
<tr>
<td>-502</td>
<td>335544576</td>
<td>dsq1_cursor_open_err</td>
<td>Attempt to reopen an open cursor</td>
</tr>
<tr>
<td>-502</td>
<td>336003090</td>
<td>dsq1 Cursor_redefined</td>
<td>Statement already has a cursor @1 assigned</td>
</tr>
<tr>
<td>-502</td>
<td>336003091</td>
<td>dsq1 Cursor_not_found</td>
<td>Cursor @1 is not found in the current context</td>
</tr>
<tr>
<td>-502</td>
<td>336003092</td>
<td>dsq1 Cursor_exists</td>
<td>Cursor @1 already exists in the current context</td>
</tr>
<tr>
<td>-502</td>
<td>336003093</td>
<td>dsq1 Cursor_rel_ambiguous</td>
<td>Relation @1 is ambiguous in cursor @2</td>
</tr>
<tr>
<td>-502</td>
<td>336003094</td>
<td>dsq1 Cursor_rel_not_found</td>
<td>Relation @1 is not found in cursor @2</td>
</tr>
<tr>
<td>-502</td>
<td>336003095</td>
<td>dsq1 Cursor_not_open</td>
<td>Cursor is not open</td>
</tr>
<tr>
<td>-504</td>
<td>335544572</td>
<td>dsq1 Cursor_err</td>
<td>Invalid cursor reference</td>
</tr>
</tbody>
</table>

Appendix B: Обработка ошибок, коды и сообщения
<table>
<thead>
<tr>
<th>SQL-CODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>-504</td>
<td>336003089</td>
<td>dsq1_cursor_invalid</td>
<td>Empty cursor name is not allowed</td>
</tr>
<tr>
<td>-508</td>
<td>335544348</td>
<td>no_cur_rec</td>
<td>no current record for fetch operation</td>
</tr>
<tr>
<td>-510</td>
<td>335544575</td>
<td>dsq1_cursor_update_err</td>
<td>Cursor @1 is not updatable</td>
</tr>
<tr>
<td>-518</td>
<td>335544582</td>
<td>dsq1_request_err</td>
<td>Request unknown</td>
</tr>
<tr>
<td>-519</td>
<td>335544688</td>
<td>dsq1_open_cursor_request</td>
<td>The prepare statement identifies a prepare statement with an open cursor</td>
</tr>
<tr>
<td>-530</td>
<td>335544466</td>
<td>foreign_key</td>
<td>violation of FOREIGN KEY constraint &quot;@1&quot; on table &quot;@2&quot;</td>
</tr>
<tr>
<td>-530</td>
<td>335544838</td>
<td>foreign_key_target_doesnt_exist</td>
<td>Foreign key reference target does not exist</td>
</tr>
<tr>
<td>-530</td>
<td>335544839</td>
<td>foreign_key_references_present</td>
<td>Foreign key references are present for the record</td>
</tr>
<tr>
<td>-531</td>
<td>335544597</td>
<td>dsq1_crdb_prepare_err</td>
<td>Cannot prepare a CREATE DATABASE/SCHEMA statement</td>
</tr>
<tr>
<td>-532</td>
<td>335544469</td>
<td>trans_invalid</td>
<td>transaction marked invalid and cannot be committed</td>
</tr>
<tr>
<td>-532</td>
<td>335545002</td>
<td>attachment_in_use</td>
<td>Attachment is in use</td>
</tr>
<tr>
<td>-532</td>
<td>335545003</td>
<td>transaction_in_use</td>
<td>Transaction is in use</td>
</tr>
<tr>
<td>-532</td>
<td>335545017</td>
<td>async_active</td>
<td>Asynchronous call is already running for this attachment</td>
</tr>
<tr>
<td>-551</td>
<td>335544352</td>
<td>no_priv</td>
<td>no permission for @1 access to @2 @3</td>
</tr>
<tr>
<td>-551</td>
<td>335544790</td>
<td>insufficient_svc_privileges</td>
<td>Service @1 requires SYSDBA permissions. Reattach to the Service Manager using the SYSDBA account.</td>
</tr>
<tr>
<td>-551</td>
<td>335545033</td>
<td>trunc_limits</td>
<td>expected length @1, actual @2</td>
</tr>
<tr>
<td>-551</td>
<td>335545034</td>
<td>info_access</td>
<td>Wrong info requested in isc_svc_query() for anonymous service</td>
</tr>
<tr>
<td>-551</td>
<td>335545036</td>
<td>svc_start_failed</td>
<td>Start request for anonymous service is impossible</td>
</tr>
<tr>
<td>-551</td>
<td>335545254</td>
<td>effective_user</td>
<td>Effective user is @1</td>
</tr>
<tr>
<td>-552</td>
<td>335544550</td>
<td>not_rel_owner</td>
<td>only the owner of a table may reassign ownership</td>
</tr>
<tr>
<td>-552</td>
<td>335544553</td>
<td>grant_nopriv</td>
<td>user does not have GRANT privileges for operation</td>
</tr>
<tr>
<td>-552</td>
<td>335544707</td>
<td>grant_nopriv_on_base</td>
<td>user does not have GRANT privileges on base table/view for operation</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>----------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>-552</td>
<td>335545058</td>
<td>protect_ownership</td>
<td>Only the owner can change the ownership</td>
</tr>
<tr>
<td>-553</td>
<td>335544529</td>
<td>existing_priv_mod</td>
<td>cannot modify an existing user privilege</td>
</tr>
<tr>
<td>-595</td>
<td>335544645</td>
<td>stream_crack</td>
<td>the current position is on a crack</td>
</tr>
<tr>
<td>-596</td>
<td>335544374</td>
<td>stream_eof</td>
<td>attempt to fetch past the last record in a record stream</td>
</tr>
<tr>
<td>-596</td>
<td>335544644</td>
<td>stream_bof</td>
<td>attempt to fetch before the first record in a record stream</td>
</tr>
<tr>
<td>-596</td>
<td>335545092</td>
<td>cursor_not_positioned</td>
<td>Cursor @1 is not positioned in a valid record</td>
</tr>
<tr>
<td>-597</td>
<td>335544632</td>
<td>dsql_file_length_err</td>
<td>Preceding file did not specify length, so @1 must include starting page number</td>
</tr>
<tr>
<td>-598</td>
<td>335544633</td>
<td>dsql_shadow_number_err</td>
<td>Shadow number must be a positive integer</td>
</tr>
<tr>
<td>-599</td>
<td>335544607</td>
<td>node_err</td>
<td>gen.c: node not supported</td>
</tr>
<tr>
<td>-599</td>
<td>335544625</td>
<td>node_name_err</td>
<td>A node name is not permitted in a secondary, shadow, cache or log file name</td>
</tr>
<tr>
<td>-600</td>
<td>335544680</td>
<td>crrp_data_err</td>
<td>sort error: corruption in data structure</td>
</tr>
<tr>
<td>-601</td>
<td>335544646</td>
<td>db_or_file_exists</td>
<td>database or file exists</td>
</tr>
<tr>
<td>-604</td>
<td>335544593</td>
<td>dsql_max_arr_dim_exceeded</td>
<td>Array declared with too many dimensions</td>
</tr>
<tr>
<td>-604</td>
<td>335544594</td>
<td>dsql_arr_range_error</td>
<td>Illegal array dimension range</td>
</tr>
<tr>
<td>-605</td>
<td>335544682</td>
<td>dsql_field_ref</td>
<td>Inappropriate self-reference of column</td>
</tr>
<tr>
<td>-607</td>
<td>335544351</td>
<td>no_meta_update</td>
<td>unsuccessful metadata update</td>
</tr>
<tr>
<td>-607</td>
<td>335544549</td>
<td>systrig_update</td>
<td>cannot modify or erase a system trigger</td>
</tr>
<tr>
<td>-607</td>
<td>335544657</td>
<td>dsql_no_blob_array</td>
<td>Array/BLOB/DATE data types not allowed in arithmetic</td>
</tr>
<tr>
<td>-607</td>
<td>335544746</td>
<td>reftableRequires_pk</td>
<td>&quot;REFERENCES table&quot; without &quot;(column)&quot; requires PRIMARY KEY on referenced table</td>
</tr>
<tr>
<td>-607</td>
<td>335544815</td>
<td>generator_name</td>
<td>GENERATOR @1</td>
</tr>
<tr>
<td>-607</td>
<td>335544816</td>
<td>udf_name</td>
<td>Function @1</td>
</tr>
<tr>
<td>-607</td>
<td>335544858</td>
<td>must_have_phys_field</td>
<td>Can’t have relation with only computed fields or constraints</td>
</tr>
<tr>
<td>SQL CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-607</td>
<td>336003074</td>
<td>dsql_dbkey_from_non_table</td>
<td>Cannot SELECT RDB$DB_KEY from a stored procedure.</td>
</tr>
<tr>
<td>-607</td>
<td>336003086</td>
<td>dsql_udf_return_pos_err</td>
<td>External function should have return position between 1 and @1</td>
</tr>
<tr>
<td>-607</td>
<td>336003096</td>
<td>dsql_type_not_supp_ext_tab</td>
<td>Data type @1 is not supported for EXTERNAL TABLES. Relation '@2', field '@3'</td>
</tr>
<tr>
<td>-607</td>
<td>336003104</td>
<td>dsql_record_version_table</td>
<td>To be used with RDB$RECORD_VERSION, @1 must be a table or a view of single table</td>
</tr>
<tr>
<td>-607</td>
<td>336068845</td>
<td>dyn_cannot_del_syscoll</td>
<td>Cannot delete system collation</td>
</tr>
<tr>
<td>-607</td>
<td>336068866</td>
<td>dyn_cannot_mod_sysproc</td>
<td>Cannot ALTER or DROP system procedure @1</td>
</tr>
<tr>
<td>-607</td>
<td>336068867</td>
<td>dyn_cannot_mod_systrig</td>
<td>Cannot ALTER or DROP system trigger @1</td>
</tr>
<tr>
<td>-607</td>
<td>336068868</td>
<td>dyn_cannot_mod_sysfunc</td>
<td>Cannot ALTER or DROP system function @1</td>
</tr>
<tr>
<td>-607</td>
<td>336068869</td>
<td>dyn_invalid_ddl_proc</td>
<td>Invalid DDL statement for procedure @1</td>
</tr>
<tr>
<td>-607</td>
<td>336068870</td>
<td>dyn_invalid_ddl_trig</td>
<td>Invalid DDL statement for trigger @1</td>
</tr>
<tr>
<td>-607</td>
<td>336068878</td>
<td>dyn_invalid_ddl_func</td>
<td>Invalid DDL statement for function @1</td>
</tr>
<tr>
<td>-607</td>
<td>336397206</td>
<td>dsql_table_not_found</td>
<td>Table @1 does not exist</td>
</tr>
<tr>
<td>-607</td>
<td>336397207</td>
<td>dsql_view_not_found</td>
<td>View @1 does not exist</td>
</tr>
<tr>
<td>-607</td>
<td>336397212</td>
<td>dsql_no_array_computed</td>
<td>Array and BLOB data types not allowed in computed field</td>
</tr>
<tr>
<td>-607</td>
<td>336397214</td>
<td>dsql_only_can_subscript_array</td>
<td>scalar operator used on field @1 which is not an array</td>
</tr>
<tr>
<td>-612</td>
<td>336068812</td>
<td>dyn_domain_name_exists</td>
<td>Cannot rename domain @1 to @2. A domain with that name already exists.</td>
</tr>
<tr>
<td>-612</td>
<td>336068813</td>
<td>dyn_field_name_exists</td>
<td>Cannot rename column @1 to @2. A column with that name already exists in table @3.</td>
</tr>
<tr>
<td>-615</td>
<td>335544475</td>
<td>relation_lock</td>
<td>lock on table @1 conflicts with existing lock</td>
</tr>
<tr>
<td>-615</td>
<td>335544476</td>
<td>record_lock</td>
<td>requested record lock conflicts with existing lock</td>
</tr>
<tr>
<td>-615</td>
<td>335544501</td>
<td>drop_wal</td>
<td>cannot drop log file when journaling is enabled</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>-615</td>
<td>335544507</td>
<td>range_in_use</td>
<td>refresh range number @1 already in use</td>
</tr>
<tr>
<td>-616</td>
<td>335544530</td>
<td>primary_key_ref</td>
<td>Cannot delete PRIMARY KEY being used in FOREIGN KEY definition.</td>
</tr>
<tr>
<td>-616</td>
<td>335544539</td>
<td>integ_index_del</td>
<td>Cannot delete index used by an Integrity Constraint</td>
</tr>
<tr>
<td>-616</td>
<td>335544540</td>
<td>integ_index_mod</td>
<td>Cannot modify index used by an Integrity Constraint</td>
</tr>
<tr>
<td>-616</td>
<td>335544541</td>
<td>check_trig_del</td>
<td>Cannot delete trigger used by a CHECK Constraint</td>
</tr>
<tr>
<td>-616</td>
<td>335544543</td>
<td>cnstrnt_fld_del</td>
<td>Cannot delete column being used in an Integrity Constraint.</td>
</tr>
<tr>
<td>-616</td>
<td>335544630</td>
<td>dependency</td>
<td>there are @1 dependencies</td>
</tr>
<tr>
<td>-616</td>
<td>335544674</td>
<td>del_last_field</td>
<td>last column in a table cannot be deleted</td>
</tr>
<tr>
<td>-616</td>
<td>335544728</td>
<td>integ_index_deactivate</td>
<td>Cannot deactivate index used by an integrity constraint</td>
</tr>
<tr>
<td>-616</td>
<td>335544729</td>
<td>integ_deactivate_primary</td>
<td>Cannot deactivate index used by a PRIMARY/UNIQUE constraint</td>
</tr>
<tr>
<td>-617</td>
<td>335544542</td>
<td>check_trig_update</td>
<td>Cannot update trigger used by a CHECK Constraint</td>
</tr>
<tr>
<td>-617</td>
<td>335544544</td>
<td>cnstrnt_fld_rename</td>
<td>Cannot rename column being used in an Integrity Constraint.</td>
</tr>
<tr>
<td>-618</td>
<td>335544537</td>
<td>integ_index_seg_del</td>
<td>Cannot delete index segment used by an Integrity Constraint</td>
</tr>
<tr>
<td>-618</td>
<td>335544538</td>
<td>integ_index_seg_mod</td>
<td>Cannot update index segment used by an Integrity Constraint</td>
</tr>
<tr>
<td>-625</td>
<td>335544347</td>
<td>not_valid</td>
<td>validation error for column @1, value &quot;@2&quot;</td>
</tr>
<tr>
<td>-625</td>
<td>335544879</td>
<td>not_valid_for_var</td>
<td>validation error for variable @1, value &quot;@2&quot;</td>
</tr>
<tr>
<td>-625</td>
<td>335544880</td>
<td>not_valid_for</td>
<td>validation error for @1, value &quot;@2&quot;</td>
</tr>
<tr>
<td>-637</td>
<td>335544664</td>
<td>dsql_duplicate_spec</td>
<td>duplicate specification of @1 - not supported</td>
</tr>
<tr>
<td>-637</td>
<td>336397213</td>
<td>dsql_implicit_domain_name</td>
<td>Implicit domain name @1 not allowed in user created domain</td>
</tr>
<tr>
<td>-660</td>
<td>335544533</td>
<td>foreign_key_notfound</td>
<td>Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY.</td>
</tr>
<tr>
<td>-660</td>
<td>335544628</td>
<td>idx_create_err</td>
<td>cannot create index @1</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-660</td>
<td>336003098</td>
<td>primary_key_required</td>
<td>Primary key required on table @1</td>
</tr>
<tr>
<td>-663</td>
<td>335544624</td>
<td>idx_seg_err</td>
<td>segment count of 0 defined for index @1</td>
</tr>
<tr>
<td>-663</td>
<td>335544631</td>
<td>idx_key_err</td>
<td>too many keys defined for index @1</td>
</tr>
<tr>
<td>-663</td>
<td>335544672</td>
<td>key_field_err</td>
<td>too few key columns found for index @1 (incorrect column name?)</td>
</tr>
<tr>
<td>-664</td>
<td>335544434</td>
<td>keytoobig</td>
<td>key size exceeds implementation restriction for index &quot;@1&quot;</td>
</tr>
<tr>
<td>-677</td>
<td>335544445</td>
<td>ext_err</td>
<td>@1 extension error</td>
</tr>
<tr>
<td>-685</td>
<td>335544465</td>
<td>bad_segstr_type</td>
<td>invalid BLOB type for operation</td>
</tr>
<tr>
<td>-685</td>
<td>335544670</td>
<td>blob_idx_err</td>
<td>attempt to index BLOB column in index @1</td>
</tr>
<tr>
<td>-685</td>
<td>335544671</td>
<td>array_idx_err</td>
<td>attempt to index array column in index @1</td>
</tr>
<tr>
<td>-689</td>
<td>335544403</td>
<td>badpagtyp</td>
<td>page @1 is of wrong type (expected @2, found @3)</td>
</tr>
<tr>
<td>-689</td>
<td>335544650</td>
<td>page_type_err</td>
<td>wrong page type</td>
</tr>
<tr>
<td>-690</td>
<td>335544679</td>
<td>no_segments_err</td>
<td>segments not allowed in expression index @1</td>
</tr>
<tr>
<td>-691</td>
<td>335544681</td>
<td>rec_size_err</td>
<td>new record size of @1 bytes is too big</td>
</tr>
<tr>
<td>-692</td>
<td>335544477</td>
<td>max_idx</td>
<td>maximum indexes per table (@1) exceeded</td>
</tr>
<tr>
<td>-693</td>
<td>335544663</td>
<td>req_max_clones_exceeded</td>
<td>Too many concurrent executions of the same request</td>
</tr>
<tr>
<td>-694</td>
<td>335544684</td>
<td>no_field_access</td>
<td>cannot access column @1 in view @2</td>
</tr>
<tr>
<td>-802</td>
<td>335544321</td>
<td>arith_except</td>
<td>arithmetic exception, numeric overflow, or string truncation</td>
</tr>
<tr>
<td>-802</td>
<td>335544836</td>
<td>concat_overflow</td>
<td>Concatenation overflow. Resulting string cannot exceed 32765 bytes in length.</td>
</tr>
<tr>
<td>-802</td>
<td>335544914</td>
<td>string_truncation</td>
<td>string right truncation</td>
</tr>
<tr>
<td>-802</td>
<td>335544915</td>
<td>blob_truncation</td>
<td>blob truncation when converting to a string: length limit exceeded</td>
</tr>
<tr>
<td>-802</td>
<td>335544916</td>
<td>numeric_out_of_range</td>
<td>numeric value is out of range</td>
</tr>
<tr>
<td>-802</td>
<td>336003105</td>
<td>dsql_invalid_sqlda_version</td>
<td>SQLDA version expected between @1 and @2, found @3</td>
</tr>
<tr>
<td>-802</td>
<td>336003106</td>
<td>dsql_sqlvar_index</td>
<td>at SQLVAR index @1</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>---------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>-802</td>
<td>336003107</td>
<td>dsq1_no_sqlind</td>
<td>empty pointer to NULL indicator variable</td>
</tr>
<tr>
<td>-802</td>
<td>336003108</td>
<td>dsq1_no_sqldata</td>
<td>empty pointer to data</td>
</tr>
<tr>
<td>-802</td>
<td>336003109</td>
<td>dsq1_no_input_sqlda</td>
<td>No SQLDA for input values provided</td>
</tr>
<tr>
<td>-802</td>
<td>336003110</td>
<td>dsq1_no_output_sqlda</td>
<td>No SQLDA for output values provided</td>
</tr>
<tr>
<td>-803</td>
<td>335544349</td>
<td>no_dup</td>
<td>attempt to store duplicate value (visible to active transactions) in unique index &quot;@1&quot;</td>
</tr>
<tr>
<td>-803</td>
<td>335544665</td>
<td>unique_key_violation</td>
<td>violation of PRIMARY or UNIQUE KEY constraint &quot;@1&quot; on table &quot;@2&quot;</td>
</tr>
<tr>
<td>-804</td>
<td>335544380</td>
<td>wronumarg</td>
<td>wrong number of arguments on call</td>
</tr>
<tr>
<td>-804</td>
<td>335544583</td>
<td>dsq1_sqlda_err</td>
<td>SQLDA error</td>
</tr>
<tr>
<td>-804</td>
<td>335544584</td>
<td>dsq1_var_count_err</td>
<td>Count of read-write columns does not equal count of values</td>
</tr>
<tr>
<td>-804</td>
<td>335544586</td>
<td>dsq1_function_err</td>
<td>Function unknown</td>
</tr>
<tr>
<td>-804</td>
<td>335544713</td>
<td>dsq1_sqlda_value_err</td>
<td>Incorrect values within SQLDA structure</td>
</tr>
<tr>
<td>-804</td>
<td>335545050</td>
<td>wrong_message_length</td>
<td>Message length passed from user application does not match set of columns</td>
</tr>
<tr>
<td>-804</td>
<td>335545051</td>
<td>no_output_format</td>
<td>Resultset is missing output format information</td>
</tr>
<tr>
<td>-804</td>
<td>335545052</td>
<td>item_finish</td>
<td>Message metadata not ready - item @1 is not finished</td>
</tr>
<tr>
<td>-804</td>
<td>335545100</td>
<td>interface_version_too_old</td>
<td>Interface @3 version too old: expected @1, found @2</td>
</tr>
<tr>
<td>-804</td>
<td>336003097</td>
<td>dsq1_feature_not_supported_ods</td>
<td>Feature not supported on ODS version older than @1,@2</td>
</tr>
<tr>
<td>-804</td>
<td>336397205</td>
<td>dsq1_too_old_ods</td>
<td>ODS versions before ODS@1 are not supported</td>
</tr>
<tr>
<td>-806</td>
<td>335544600</td>
<td>col_name_err</td>
<td>Only simple column names permitted for VIEW WITH CHECK OPTION</td>
</tr>
<tr>
<td>-807</td>
<td>335544601</td>
<td>where_err</td>
<td>No WHERE clause for VIEW WITH CHECK OPTION</td>
</tr>
<tr>
<td>-808</td>
<td>335544602</td>
<td>table_view_err</td>
<td>Only one table allowed for VIEW WITH CHECK OPTION</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-809</td>
<td>335544603</td>
<td>distinct_err</td>
<td>DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION</td>
</tr>
<tr>
<td>-810</td>
<td>335544605</td>
<td>subquery_err</td>
<td>No subqueries permitted for VIEW WITH CHECK OPTION</td>
</tr>
<tr>
<td>-811</td>
<td>335544652</td>
<td>sing_select_err</td>
<td>multiple rows in singleton select</td>
</tr>
<tr>
<td>-811</td>
<td>335545269</td>
<td>merge_dup_update</td>
<td>Multiple source records cannot match the same target during MERGE</td>
</tr>
<tr>
<td>-816</td>
<td>335544651</td>
<td>ext_readonly_err</td>
<td>Cannot insert because the file is readonly or is on a read only medium.</td>
</tr>
<tr>
<td>-816</td>
<td>335544715</td>
<td>extfile_uns_op</td>
<td>Operation not supported for EXTERNAL FILE table @1</td>
</tr>
<tr>
<td>-817</td>
<td>335544361</td>
<td>read_only_trans</td>
<td>attempted update during read-only transaction</td>
</tr>
<tr>
<td>-817</td>
<td>335544371</td>
<td>segstr_no_write</td>
<td>attempted write to read-only BLOB</td>
</tr>
<tr>
<td>-817</td>
<td>335544444</td>
<td>read_only</td>
<td>operation not supported</td>
</tr>
<tr>
<td>-817</td>
<td>335544765</td>
<td>read_only_database</td>
<td>attempted update on read-only database</td>
</tr>
<tr>
<td>-817</td>
<td>335544766</td>
<td>must_be_dialect_2_and_up</td>
<td>SQL dialect @1 is not supported in this database</td>
</tr>
<tr>
<td>-817</td>
<td>335544793</td>
<td>ddl_not_allowed_by_db_sql_dial</td>
<td>Metadata update statement is not allowed by the current database SQL dialect @1</td>
</tr>
<tr>
<td>-817</td>
<td>336003079</td>
<td>sql_dialect_conflict_num</td>
<td>DB dialect @1 and client dialect @2 conflict with respect to numeric precision @3.</td>
</tr>
<tr>
<td>-817</td>
<td>336003101</td>
<td>upd_ins_with_complex_view</td>
<td>UPDATE OR INSERT without MATCHING could not be used with views based on more than one table</td>
</tr>
<tr>
<td>-817</td>
<td>336003102</td>
<td>dsql_incompatible_trigger_type</td>
<td>Incompatible trigger type</td>
</tr>
<tr>
<td>-817</td>
<td>336003103</td>
<td>dsqldb_trigger_type_cant_change</td>
<td>Database trigger type can't be changed</td>
</tr>
<tr>
<td>-817</td>
<td>336003112</td>
<td>dsqldb_invalid_drop_ss_clause</td>
<td>Invalid DROP SQL SECURITY clause</td>
</tr>
<tr>
<td>-820</td>
<td>335544356</td>
<td>obsolete_metadata</td>
<td>metadata is obsolete</td>
</tr>
<tr>
<td>-820</td>
<td>335544379</td>
<td>wrong_ods</td>
<td>unsupported on-disk structure for file @1; found @2.@3, support @4.@5</td>
</tr>
<tr>
<td>-820</td>
<td>335544437</td>
<td>wrodynver</td>
<td>wrong DYN version</td>
</tr>
<tr>
<td>-820</td>
<td>335544467</td>
<td>high_minor</td>
<td>minor version too high found @1 expected @2</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-820</td>
<td>335544881</td>
<td>need_difference</td>
<td>Difference file name should be set explicitly for database on raw device</td>
</tr>
<tr>
<td>-823</td>
<td>335544473</td>
<td>invalid_bookmark</td>
<td>invalid bookmark handle</td>
</tr>
<tr>
<td>-824</td>
<td>335544474</td>
<td>bad_lock_level</td>
<td>invalid lock level @1</td>
</tr>
<tr>
<td>-825</td>
<td>335544519</td>
<td>bad_lock_handle</td>
<td>invalid lock handle</td>
</tr>
<tr>
<td>-826</td>
<td>335544585</td>
<td>dsql_stmt_handle</td>
<td>Invalid statement handle</td>
</tr>
<tr>
<td>-827</td>
<td>335544655</td>
<td>invalid_direction</td>
<td>invalid direction for find operation</td>
</tr>
<tr>
<td>-827</td>
<td>335544718</td>
<td>invalid_key</td>
<td>Invalid key for find operation</td>
</tr>
<tr>
<td>-828</td>
<td>335544678</td>
<td>inval_key_posn</td>
<td>invalid key position</td>
</tr>
<tr>
<td>-829</td>
<td>335544616</td>
<td>field_ref_err</td>
<td>invalid column reference</td>
</tr>
<tr>
<td>-829</td>
<td>336068816</td>
<td>dyn_char_fld_too_small</td>
<td>New size specified for column @1 must be at least @2 characters.</td>
</tr>
<tr>
<td>-829</td>
<td>336068817</td>
<td>dyn_invalid_dtype_conversion</td>
<td>Cannot change datatype for @1. Conversion from base type @2 to @3 is not supported.</td>
</tr>
<tr>
<td>-829</td>
<td>336068818</td>
<td>dyn_dtype_conv_invalid</td>
<td>Cannot change datatype for column @1 from a character type to a non-character type.</td>
</tr>
<tr>
<td>-829</td>
<td>336068829</td>
<td>max_coll_per_charset</td>
<td>Maximum number of collations per character set exceeded</td>
</tr>
<tr>
<td>-829</td>
<td>336068830</td>
<td>invalid_coll_attr</td>
<td>Invalid collation attributes</td>
</tr>
<tr>
<td>-829</td>
<td>336068852</td>
<td>dyn_scale_too_big</td>
<td>New scale specified for column @1 must be at most @2.</td>
</tr>
<tr>
<td>-829</td>
<td>336068853</td>
<td>dyn_precision_too_small</td>
<td>New precision specified for column @1 must be at least @2.</td>
</tr>
<tr>
<td>-829</td>
<td>336068857</td>
<td>dyn_cannot_addrem_computed</td>
<td>Cannot add or remove COMPUTED from column @1</td>
</tr>
<tr>
<td>-830</td>
<td>335544615</td>
<td>field_aggregate_err</td>
<td>column used with aggregate</td>
</tr>
<tr>
<td>-831</td>
<td>335544548</td>
<td>primary_key_exists</td>
<td>Attempt to define a second PRIMARY KEY for the same table</td>
</tr>
<tr>
<td>-832</td>
<td>335544604</td>
<td>key_field_count_err</td>
<td>FOREIGN KEY column count does not match PRIMARY KEY</td>
</tr>
<tr>
<td>-833</td>
<td>335544606</td>
<td>expression_eval_err</td>
<td>expression evaluation not supported</td>
</tr>
<tr>
<td>-833</td>
<td>335544810</td>
<td>date_range_exceeded</td>
<td>value exceeds the range for valid dates</td>
</tr>
<tr>
<td>-833</td>
<td>335544912</td>
<td>time_range_exceeded</td>
<td>value exceeds the range for a valid time</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-833</td>
<td>335544913</td>
<td><code>datetime_range_exceeded</code></td>
<td>value exceeds the range for valid timestamps</td>
</tr>
<tr>
<td>-833</td>
<td>335544937</td>
<td><code>invalid_type_datetime_op</code></td>
<td>Invalid data type in DATE/TIME/TIMESTAMP addition or subtraction in add_datetime()</td>
</tr>
<tr>
<td>-833</td>
<td>335544938</td>
<td><code>onlycan_add_timetodate</code></td>
<td>Only a TIME value can be added to a DATE value</td>
</tr>
<tr>
<td>-833</td>
<td>335544939</td>
<td><code>onlycan_add_datetotime</code></td>
<td>Only a DATE value can be added to a TIME value</td>
</tr>
<tr>
<td>-833</td>
<td>335544940</td>
<td><code>onlycansub_tstampfromtstamp</code></td>
<td>TIMESTAMP values can be subtracted only from another TIMESTAMP value</td>
</tr>
<tr>
<td>-833</td>
<td>335544941</td>
<td><code>onlyoneop_mustbe_tstamp</code></td>
<td>Only one operand can be of type TIMESTAMP</td>
</tr>
<tr>
<td>-833</td>
<td>335544942</td>
<td><code>invalid_extractpart_time</code></td>
<td>Only HOUR, MINUTE, SECOND and MILLISECOND can be extracted from TIME values</td>
</tr>
<tr>
<td>-833</td>
<td>335544943</td>
<td><code>invalid_extractpart_date</code></td>
<td>HOUR, MINUTE, SECOND and MILLISECOND cannot be extracted from DATE values</td>
</tr>
<tr>
<td>-833</td>
<td>335544944</td>
<td><code>invalidarg_extract</code></td>
<td>Invalid argument for EXTRACT() not being of DATE/TIME/TIMESTAMP type</td>
</tr>
<tr>
<td>-833</td>
<td>335544945</td>
<td><code>sysf_argmustbe_exact</code></td>
<td>Arguments for @1 must be integral types or NUMERIC/DECIMAL without scale</td>
</tr>
<tr>
<td>-833</td>
<td>335544946</td>
<td><code>sysf_argmustbe_exact_or_fp</code></td>
<td>First argument for @1 must be integral type or floating point type</td>
</tr>
<tr>
<td>-833</td>
<td>335544947</td>
<td><code>sysf_argviolates_uuidtype</code></td>
<td>Human readable UUID argument for @1 must be of string type</td>
</tr>
<tr>
<td>-833</td>
<td>335544948</td>
<td><code>sysf_argviolates_uuidlen</code></td>
<td>Human readable UUID argument for @2 must be of exact length @1</td>
</tr>
<tr>
<td>-833</td>
<td>335544949</td>
<td><code>sysf_argviolates_uiddfmt</code></td>
<td>Human readable UUID argument for @3 must have &quot;-&quot; at position @2 instead of &quot;@1&quot;</td>
</tr>
<tr>
<td>-833</td>
<td>335544950</td>
<td><code>sysf_argviolates_guidigits</code></td>
<td>Human readable UUID argument for @3 must have hex digit at position @2 instead of &quot;@1&quot;</td>
</tr>
<tr>
<td>-833</td>
<td>335544951</td>
<td><code>sysf_invalid_addpart_time</code></td>
<td>Only HOUR, MINUTE, SECOND and MILLISECOND can be added to TIME values in @1</td>
</tr>
<tr>
<td>SQL-Code</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-833</td>
<td>335544952</td>
<td>sysf_invalid_add_datetime</td>
<td>Invalid data type in addition of part to DATE/TIME/TIMESTAMP in @1</td>
</tr>
<tr>
<td>-833</td>
<td>335544953</td>
<td>sysf_invalid_addpart_dtime</td>
<td>Invalid part @1 to be added to a DATE/TIME/TIMESTAMP value in @2</td>
</tr>
<tr>
<td>-833</td>
<td>335544954</td>
<td>sysf_invalid_add_dtime_rc</td>
<td>Expected DATE/TIME/TIMESTAMP type in evlDateAdd() result</td>
</tr>
<tr>
<td>-833</td>
<td>335544955</td>
<td>sysf_invalid_diff_dtime</td>
<td>Expected DATE/TIME/TIMESTAMP type as first and second argument to @1</td>
</tr>
<tr>
<td>-833</td>
<td>335544956</td>
<td>sysf_invalid_timediff</td>
<td>The result of TIME-&lt;value&gt; in @1 cannot be expressed in YEAR, MONTH, DAY or WEEK</td>
</tr>
<tr>
<td>-833</td>
<td>335544957</td>
<td>sysf_invalid_tstamp timediff</td>
<td>The result of TIME-TIMESTAMP or TIMESTAMP-TIME in @1 cannot be expressed in HOUR, MINUTE, SECOND or MILLISECOND</td>
</tr>
<tr>
<td>-833</td>
<td>335544958</td>
<td>sysf_invalid_datetimediff</td>
<td>The result of DATE-TIME or TIME-DATE in @1 cannot be expressed in HOUR, MINUTE, SECOND and MILLISECOND</td>
</tr>
<tr>
<td>-833</td>
<td>335544959</td>
<td>sysf_invalid_diffpart</td>
<td>Invalid part @1 to express the difference between two DATE/TIME/TIMESTAMP values in @2</td>
</tr>
<tr>
<td>-833</td>
<td>335544960</td>
<td>sysf_argmustbe_positive</td>
<td>Argument for @1 must be positive</td>
</tr>
<tr>
<td>-833</td>
<td>335544961</td>
<td>sysf_base mustbe_positive</td>
<td>Base for @1 must be positive</td>
</tr>
<tr>
<td>-833</td>
<td>335544962</td>
<td>sysf_argnmustbe_nonneg</td>
<td>Argument #@1 for @2 must be zero or positive</td>
</tr>
<tr>
<td>-833</td>
<td>335544963</td>
<td>sysf_argnmustbe_positive</td>
<td>Argument #@1 for @2 must be positive</td>
</tr>
<tr>
<td>-833</td>
<td>335544964</td>
<td>sysf_invalid_zeropowneg</td>
<td>Base for @1 cannot be zero if exponent is negative</td>
</tr>
<tr>
<td>-833</td>
<td>335544965</td>
<td>sysf_invalid_negpowfp</td>
<td>Base for @1 cannot be negative if exponent is not an integral value</td>
</tr>
<tr>
<td>-833</td>
<td>335544966</td>
<td>sysf_invalid_scale</td>
<td>The numeric scale must be between -128 and 127 in @1</td>
</tr>
<tr>
<td>-833</td>
<td>335544967</td>
<td>sysf_argmustbe_nonneg</td>
<td>Argument for @1 must be zero or positive</td>
</tr>
<tr>
<td>-833</td>
<td>335544968</td>
<td>sysf_binuuuid mustbe_str</td>
<td>Binary UUID argument for @1 must be of string type</td>
</tr>
<tr>
<td>-833</td>
<td>335544969</td>
<td>sysf_binuuuid wrongsize</td>
<td>Binary UUID argument for @2 must use @1 bytes</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>-833</td>
<td>335544976</td>
<td>sysf_argmustbe_nonzero</td>
<td>Argument for @1 must be different than zero</td>
</tr>
<tr>
<td>-833</td>
<td>335544977</td>
<td>sysf_argmustbe_range_inc1_1</td>
<td>Argument for @1 must be in the range [-1, 1]</td>
</tr>
<tr>
<td>-833</td>
<td>335544978</td>
<td>sysf_argmustbe_gteq_one</td>
<td>Argument for @1 must be greater or equal than one</td>
</tr>
<tr>
<td>-833</td>
<td>335544979</td>
<td>sysf_argmustbe_range_exc1_1</td>
<td>Argument for @1 must be in the range [-1, 1[</td>
</tr>
<tr>
<td>-833</td>
<td>335544981</td>
<td>sys_f_p_overflow</td>
<td>Floating point overflow in built-in function @1</td>
</tr>
<tr>
<td>-833</td>
<td>335545009</td>
<td>sysf_invalid_trig_namespace</td>
<td>Invalid usage of context namespace DDL_TRIGGER</td>
</tr>
<tr>
<td>-833</td>
<td>335545024</td>
<td>sysf_argscant_both_be_zero</td>
<td>Arguments for @1 cannot both be zero</td>
</tr>
<tr>
<td>-833</td>
<td>335545046</td>
<td>max_args_exceeded</td>
<td>Maximum (@1) number of arguments exceeded for function @2</td>
</tr>
<tr>
<td>-833</td>
<td>335545120</td>
<td>window_frame_value_invalid</td>
<td>Invalid PRECEDING or FOLLOWING offset in window function: cannot be negative</td>
</tr>
<tr>
<td>-833</td>
<td>335545121</td>
<td>dsql_window_not_found</td>
<td>Window @1 not found</td>
</tr>
<tr>
<td>-833</td>
<td>335545122</td>
<td>dsql_window_cant_overr_part</td>
<td>Cannot use PARTITION BY clause while overriding the window @1</td>
</tr>
<tr>
<td>-833</td>
<td>335545123</td>
<td>dsql_window_cant_overr_order</td>
<td>Cannot use ORDER BY clause while overriding the window @1 which already has an ORDER BY clause</td>
</tr>
<tr>
<td>-833</td>
<td>335545124</td>
<td>dsql_window_cant_overr_frame</td>
<td>Cannot override the window @1 because it has a frame clause. Tip: it can be used without parenthesis in OVER</td>
</tr>
<tr>
<td>-833</td>
<td>335545125</td>
<td>dsql_window_duplicate</td>
<td>Duplicate window definition for @1</td>
</tr>
<tr>
<td>-833</td>
<td>335545156</td>
<td>sysf_invalid_first_last_part</td>
<td>Invalid part @1 to calculate the @1 of a DATE/TIMESTAMP</td>
</tr>
<tr>
<td>-833</td>
<td>335545157</td>
<td>sysf_invalid_date_timestamp</td>
<td>Expected DATE/TIMESTAMP value in @1</td>
</tr>
<tr>
<td>-833</td>
<td>336397240</td>
<td>dsql_eval_unknode</td>
<td>Unknown node type @1 in dsq/GEN_expr</td>
</tr>
<tr>
<td>-833</td>
<td>336397241</td>
<td>dsql_agg_wrongarg</td>
<td>Argument for @1 in dialect 1 must be string or numeric</td>
</tr>
<tr>
<td>-833</td>
<td>336397242</td>
<td>dsql_agg2_wrongarg</td>
<td>Argument for @1 in dialect 3 must be numeric</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-833</td>
<td>336397243</td>
<td>dsql_nodateortime_pm_string</td>
<td>Strings cannot be added to or subtracted from DATE or TIME types</td>
</tr>
<tr>
<td>-833</td>
<td>336397244</td>
<td>dsql_invalid_datetime_subtract</td>
<td>Invalid data type for subtraction involving DATE, TIME or TIMESTAMP types</td>
</tr>
<tr>
<td>-833</td>
<td>336397245</td>
<td>dsql_invalid_dateortime_add</td>
<td>Adding two DATE values or two TIME values is not allowed</td>
</tr>
<tr>
<td>-833</td>
<td>336397246</td>
<td>dsql_invalid_type_minus_date</td>
<td>DATE value cannot be subtracted from the provided data type</td>
</tr>
<tr>
<td>-833</td>
<td>336397247</td>
<td>dsql_nostring_addsub_dial3</td>
<td>Strings cannot be added or subtracted in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397248</td>
<td>dsql_invalid_type_addsub_dial3</td>
<td>Invalid data type for addition or subtraction in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397249</td>
<td>dsql_invalid_type_multip_dial1</td>
<td>Invalid data type for multiplication in dialect 1</td>
</tr>
<tr>
<td>-833</td>
<td>336397250</td>
<td>dsql_nostring_multip_dial3</td>
<td>Strings cannot be multiplied in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397251</td>
<td>dsql_invalid_type_multip_dial3</td>
<td>Invalid data type for multiplication in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397252</td>
<td>dsql_mustuse_numeric_div_dial1</td>
<td>Division in dialect 1 must be between numeric data types</td>
</tr>
<tr>
<td>-833</td>
<td>336397253</td>
<td>dsql_nostring_div_dial3</td>
<td>Strings cannot be divided in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397254</td>
<td>dsql_invalid_type_div_dial3</td>
<td>Invalid data type for division in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397255</td>
<td>dsql_nostring_neg_dial3</td>
<td>Strings cannot be negated (applied the minus operator) in dialect 3</td>
</tr>
<tr>
<td>-833</td>
<td>336397256</td>
<td>dsql_invalid_type_neg</td>
<td>Invalid data type for negation (minus operator)</td>
</tr>
<tr>
<td>-834</td>
<td>335544508</td>
<td>range_not_found</td>
<td>refresh range number @1 not found</td>
</tr>
<tr>
<td>-835</td>
<td>335544649</td>
<td>bad_checksum</td>
<td>bad checksum</td>
</tr>
<tr>
<td>-836</td>
<td>335544517</td>
<td>except</td>
<td>exception @1</td>
</tr>
<tr>
<td>-836</td>
<td>335544848</td>
<td>except2</td>
<td>exception @1</td>
</tr>
<tr>
<td>-836</td>
<td>335545016</td>
<td>formatted_exception</td>
<td>@1</td>
</tr>
<tr>
<td>-837</td>
<td>335544518</td>
<td>cache_restart</td>
<td>restart shared cache manager</td>
</tr>
<tr>
<td>-838</td>
<td>335544560</td>
<td>shutwarn</td>
<td>database @1 shutdown in @2 seconds</td>
</tr>
<tr>
<td>-839</td>
<td>335544686</td>
<td>jrn_format_err</td>
<td>journal file wrong format</td>
</tr>
<tr>
<td>-840</td>
<td>335544687</td>
<td>jrn_file_full</td>
<td>intermediate journal file full</td>
</tr>
<tr>
<td>-841</td>
<td>335544677</td>
<td>version_err</td>
<td>too many versions</td>
</tr>
</tbody>
</table>
## Appendix B: Обработка ошибок, коды и сообщения

<table>
<thead>
<tr>
<th>SQL-CODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>-842</td>
<td>335544697</td>
<td>precision_err</td>
<td>Precision must be from 1 to 18</td>
</tr>
<tr>
<td>-842</td>
<td>335544698</td>
<td>scale_nogt</td>
<td>Scale must be between zero and precision</td>
</tr>
<tr>
<td>-842</td>
<td>335544699</td>
<td>expec_short</td>
<td>Short integer expected</td>
</tr>
<tr>
<td>-842</td>
<td>335544700</td>
<td>expec_long</td>
<td>Long integer expected</td>
</tr>
<tr>
<td>-842</td>
<td>335544701</td>
<td>expec_ushort</td>
<td>Unsigned short integer expected</td>
</tr>
<tr>
<td>-842</td>
<td>335544712</td>
<td>expec_positive</td>
<td>Positive value expected</td>
</tr>
<tr>
<td>-842</td>
<td>335545138</td>
<td>decprecision_err</td>
<td>DecFloat precision must be 16 or 34</td>
</tr>
<tr>
<td>-842</td>
<td>335545158</td>
<td>precision_err2</td>
<td>Precision must be from @1 to @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544322</td>
<td>bad_dbkey</td>
<td>invalid database key</td>
</tr>
<tr>
<td>-901</td>
<td>335544326</td>
<td>bad_dpb_form</td>
<td>unrecognized database parameter block</td>
</tr>
<tr>
<td>-901</td>
<td>335544327</td>
<td>bad_req_handle</td>
<td>invalid request handle</td>
</tr>
<tr>
<td>-901</td>
<td>335544328</td>
<td>bad_segstr_handle</td>
<td>invalid BLOB handle</td>
</tr>
<tr>
<td>-901</td>
<td>335544329</td>
<td>bad_segstr_id</td>
<td>invalid BLOB ID</td>
</tr>
<tr>
<td>-901</td>
<td>335544330</td>
<td>bad_tpb_content</td>
<td>invalid parameter in transaction parameter block</td>
</tr>
<tr>
<td>-901</td>
<td>335544331</td>
<td>bad_tpb_form</td>
<td>invalid format for transaction parameter block</td>
</tr>
<tr>
<td>-901</td>
<td>335544332</td>
<td>bad_trans_handle</td>
<td>invalid transaction handle (expecting explicit transaction start)</td>
</tr>
<tr>
<td>-901</td>
<td>335544337</td>
<td>excess_trans</td>
<td>attempt to start more than @1 transactions</td>
</tr>
<tr>
<td>-901</td>
<td>335544339</td>
<td>infinap</td>
<td>information type inappropriate for object specified</td>
</tr>
<tr>
<td>-901</td>
<td>335544340</td>
<td>infona</td>
<td>no information of this type available for object specified</td>
</tr>
<tr>
<td>-901</td>
<td>335544341</td>
<td>infunk</td>
<td>unknown information item</td>
</tr>
<tr>
<td>-901</td>
<td>335544342</td>
<td>integ_fail</td>
<td>action cancelled by trigger (@1) to preserve data integrity</td>
</tr>
<tr>
<td>-901</td>
<td>335544345</td>
<td>lock_conflict</td>
<td>lock conflict on no wait transaction</td>
</tr>
<tr>
<td>-901</td>
<td>335544350</td>
<td>no_finish</td>
<td>program attempted to exit without finishing database</td>
</tr>
<tr>
<td>-901</td>
<td>335544353</td>
<td>no_recon</td>
<td>transaction is not in limbo</td>
</tr>
<tr>
<td>-901</td>
<td>335544355</td>
<td>no_segstr_close</td>
<td>BLOB was not closed</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544357</td>
<td>open_trans</td>
<td>cannot disconnect database with open transactions (@1 active)</td>
</tr>
<tr>
<td>-901</td>
<td>335544358</td>
<td>port_len</td>
<td>message length error (encountered @1, expected @2)</td>
</tr>
<tr>
<td>-901</td>
<td>335544363</td>
<td>req_no_trans</td>
<td>no transaction for request</td>
</tr>
<tr>
<td>-901</td>
<td>335544364</td>
<td>req_sync</td>
<td>request synchronization error</td>
</tr>
<tr>
<td>-901</td>
<td>335544365</td>
<td>req_wrong_db</td>
<td>request referenced an unavailable database</td>
</tr>
<tr>
<td>-901</td>
<td>335544369</td>
<td>segstr_no_read</td>
<td>attempted read of a new, open BLOB</td>
</tr>
<tr>
<td>-901</td>
<td>335544370</td>
<td>segstr_no_trans</td>
<td>attempted action on BLOB outside transaction</td>
</tr>
<tr>
<td>-901</td>
<td>335544372</td>
<td>segstr_wrong_db</td>
<td>attempted reference to BLOB in unavailable database</td>
</tr>
<tr>
<td>-901</td>
<td>335544376</td>
<td>unres_rel</td>
<td>table @1 was omitted from the transaction reserving list</td>
</tr>
<tr>
<td>-901</td>
<td>335544377</td>
<td>uns_ext</td>
<td>request includes a DSRI extension not supported in this implementation</td>
</tr>
<tr>
<td>-901</td>
<td>335544378</td>
<td>wish_list</td>
<td>feature is not supported</td>
</tr>
<tr>
<td>-901</td>
<td>335544382</td>
<td>random</td>
<td>@1</td>
</tr>
<tr>
<td>-901</td>
<td>335544383</td>
<td>fatal_conflict</td>
<td>unrecoverable conflict with limbo transaction @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544392</td>
<td>bdbincon</td>
<td>internal error</td>
</tr>
<tr>
<td>-901</td>
<td>335544407</td>
<td>dbbnotzer</td>
<td>database handle not zero</td>
</tr>
<tr>
<td>-901</td>
<td>335544408</td>
<td>tranotzer</td>
<td>transaction handle not zero</td>
</tr>
<tr>
<td>-901</td>
<td>335544418</td>
<td>trainlim</td>
<td>transaction in limbo</td>
</tr>
<tr>
<td>-901</td>
<td>335544419</td>
<td>notinlim</td>
<td>transaction not in limbo</td>
</tr>
<tr>
<td>-901</td>
<td>335544420</td>
<td>traoutsta</td>
<td>transaction outstanding</td>
</tr>
<tr>
<td>-901</td>
<td>335544428</td>
<td>badmsgnum</td>
<td>undefined message number</td>
</tr>
<tr>
<td>-901</td>
<td>335544431</td>
<td>blocking_signal</td>
<td>blocking signal has been received</td>
</tr>
<tr>
<td>-901</td>
<td>335544442</td>
<td>noargacc_read</td>
<td>database system cannot read argument @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544443</td>
<td>noargacc_write</td>
<td>database system cannot write argument @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544450</td>
<td>misc_interpreted</td>
<td>@1</td>
</tr>
<tr>
<td>-901</td>
<td>335544468</td>
<td>tra_state</td>
<td>transaction @1 is @2</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544485</td>
<td>bad_stmt_handle</td>
<td>invalid statement handle</td>
</tr>
<tr>
<td>-901</td>
<td>335544510</td>
<td>lock_timeout</td>
<td>lock time-out on wait transaction</td>
</tr>
<tr>
<td>-901</td>
<td>335544559</td>
<td>bad_svc_handle</td>
<td>invalid service handle</td>
</tr>
<tr>
<td>-901</td>
<td>335544561</td>
<td>wrospbver</td>
<td>wrong version of service parameter block</td>
</tr>
<tr>
<td>-901</td>
<td>335544562</td>
<td>bad_spb_form</td>
<td>unrecognized service parameter block</td>
</tr>
<tr>
<td>-901</td>
<td>335544563</td>
<td>svcnotdef</td>
<td>service @1 is not defined</td>
</tr>
<tr>
<td>-901</td>
<td>335544609</td>
<td>index_name</td>
<td>INDEX @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544610</td>
<td>exception_name</td>
<td>EXCEPTION @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544611</td>
<td>field_name</td>
<td>COLUMN @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544613</td>
<td>union_err</td>
<td>union not supported</td>
</tr>
<tr>
<td>-901</td>
<td>335544614</td>
<td>dsql_construct_err</td>
<td>Unsupported DSQL construct</td>
</tr>
<tr>
<td>-901</td>
<td>335544623</td>
<td>dsql_domain_err</td>
<td>Illegal use of keyword VALUE</td>
</tr>
<tr>
<td>-901</td>
<td>335544626</td>
<td>table_name</td>
<td>TABLE @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544627</td>
<td>proc_name</td>
<td>PROCEDURE @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544641</td>
<td>dsql_domain_not_found</td>
<td>Specified domain or source column @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>335544656</td>
<td>dsql_var_conflict</td>
<td>variable @1 conflicts with parameter in same procedure</td>
</tr>
<tr>
<td>-901</td>
<td>335544666</td>
<td>srvr_version_too_old</td>
<td>server version too old to support all CREATE DATABASE options</td>
</tr>
<tr>
<td>-901</td>
<td>335544673</td>
<td>no_delete</td>
<td>cannot delete</td>
</tr>
<tr>
<td>-901</td>
<td>335544675</td>
<td>sort_err</td>
<td>sort error</td>
</tr>
<tr>
<td>-901</td>
<td>335544703</td>
<td>svcnoexe</td>
<td>service @1 does not have an associated executable</td>
</tr>
<tr>
<td>-901</td>
<td>335544704</td>
<td>net_lookup_err</td>
<td>Failed to locate host machine.</td>
</tr>
<tr>
<td>-901</td>
<td>335544705</td>
<td>service_unknown</td>
<td>Undefined service @1/@2.</td>
</tr>
<tr>
<td>-901</td>
<td>335544706</td>
<td>host_unknown</td>
<td>The specified name was not found in the hosts file or Domain Name Services.</td>
</tr>
<tr>
<td>-901</td>
<td>335544711</td>
<td>unprepared_stmt</td>
<td>Attempt to execute an unprepared dynamic SQL statement.</td>
</tr>
<tr>
<td>-901</td>
<td>335544716</td>
<td>svc_in_use</td>
<td>Service is currently busy: @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544719</td>
<td>net_init_error</td>
<td>Error initializing the network software.</td>
</tr>
<tr>
<td>-901</td>
<td>335544720</td>
<td>loadlib_failure</td>
<td>Unable to load required library @1.</td>
</tr>
<tr>
<td>SQL- CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544731</td>
<td>tra_must_sweep</td>
<td>A fatal exception occurred during the execution of a user defined function.</td>
</tr>
<tr>
<td>-901</td>
<td>335544740</td>
<td>udf_exception</td>
<td>A fatal exception occurred during the execution of a user defined function.</td>
</tr>
<tr>
<td>-901</td>
<td>335544741</td>
<td>lost_db_connection</td>
<td>connection lost to database</td>
</tr>
<tr>
<td>-901</td>
<td>335544742</td>
<td>no_write_user_priv</td>
<td>User cannot write to RDB$USER_PRIVILEGES</td>
</tr>
<tr>
<td>-901</td>
<td>335544767</td>
<td>blob_filter_exception</td>
<td>A fatal exception occurred during the execution of a blob filter.</td>
</tr>
<tr>
<td>-901</td>
<td>335544768</td>
<td>exception_access_violation</td>
<td>Access violation. The code attempted to access a virtual address without privilege to do so.</td>
</tr>
<tr>
<td>-901</td>
<td>335544769</td>
<td>exception_datatype_missalignmen</td>
<td>Datatype misalignment. The attempted to read or write a value that was not stored on a memory boundary.</td>
</tr>
<tr>
<td>-901</td>
<td>335544770</td>
<td>exception_array_bounds_exceeded</td>
<td>Array bounds exceeded. The code attempted to access an array element that is out of bounds.</td>
</tr>
<tr>
<td>-901</td>
<td>335544771</td>
<td>exception_float_denormal_operand</td>
<td>Float denormal operand. One of the floating-point operands is too small to represent a standard float value.</td>
</tr>
<tr>
<td>-901</td>
<td>335544772</td>
<td>exception_float_divide_by_zero</td>
<td>Floating-point divide by zero. The code attempted to divide a floating-point value by zero.</td>
</tr>
<tr>
<td>-901</td>
<td>335544773</td>
<td>exception_float_inexact_result</td>
<td>Floating-point inexact result. The result of a floating-point operation cannot be represented as a decimal fraction.</td>
</tr>
<tr>
<td>-901</td>
<td>335544774</td>
<td>exception_float_invalid_operand</td>
<td>Floating-point invalid operand. An indeterminant error occurred during a floating-point operation.</td>
</tr>
<tr>
<td>-901</td>
<td>335544775</td>
<td>exception_float_overflow</td>
<td>Floating-point overflow. The exponent of a floating-point operation is greater than the magnitude allowed.</td>
</tr>
<tr>
<td>-901</td>
<td>335544776</td>
<td>exception_float_stack_check</td>
<td>Floating-point stack check. The stack overflowed or underflowed as the result of a floating-point operation.</td>
</tr>
<tr>
<td>-901</td>
<td>335544777</td>
<td>exception_float_underflow</td>
<td>Floating-point underflow. The exponent of a floating-point operation is less than the magnitude allowed.</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544778</td>
<td>exception_integer_divide_by_zero</td>
<td>Integer divide by zero. The code attempted to divide an integer value by an integer divisor of zero.</td>
</tr>
<tr>
<td>-901</td>
<td>335544779</td>
<td>exception_integer_overflow</td>
<td>Integer overflow. The result of an integer operation caused the most significant bit of the result to carry.</td>
</tr>
<tr>
<td>-901</td>
<td>335544780</td>
<td>exception_unknown</td>
<td>An exception occurred that does not have a description. Exception number @1.</td>
</tr>
<tr>
<td>-901</td>
<td>335544781</td>
<td>exception_stack_overflow</td>
<td>Stack overflow. The resource requirements of the runtime stack have exceeded the memory available to it.</td>
</tr>
<tr>
<td>-901</td>
<td>335544782</td>
<td>exception_sigsegv</td>
<td>Segmentation Fault. The code attempted to access memory without privileges.</td>
</tr>
<tr>
<td>-901</td>
<td>335544783</td>
<td>exception_sigill</td>
<td>Illegal Instruction. The Code attempted to perform an illegal operation.</td>
</tr>
<tr>
<td>-901</td>
<td>335544784</td>
<td>exception_sigbus</td>
<td>Bus Error. The Code caused a system bus error.</td>
</tr>
<tr>
<td>-901</td>
<td>335544785</td>
<td>exception_sigfpe</td>
<td>Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception.</td>
</tr>
<tr>
<td>-901</td>
<td>335544786</td>
<td>ext_file_delete</td>
<td>Cannot delete rows from external files.</td>
</tr>
<tr>
<td>-901</td>
<td>335544787</td>
<td>ext_file_modify</td>
<td>Cannot update rows in external files.</td>
</tr>
<tr>
<td>-901</td>
<td>335544788</td>
<td>adm_task_denied</td>
<td>Unable to perform operation</td>
</tr>
<tr>
<td>-901</td>
<td>335544794</td>
<td>cancelled</td>
<td>operation was cancelled</td>
</tr>
<tr>
<td>-901</td>
<td>335544797</td>
<td>svcnouser</td>
<td>user name and password are required while attaching to the services manager</td>
</tr>
<tr>
<td>-901</td>
<td>335544801</td>
<td>datatype_notsup</td>
<td>data type not supported for arithmetic</td>
</tr>
<tr>
<td>-901</td>
<td>335544803</td>
<td>dialect_not_changed</td>
<td>Database dialect not changed.</td>
</tr>
<tr>
<td>-901</td>
<td>335544804</td>
<td>database_create_failed</td>
<td>Unable to create database @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544805</td>
<td>inv_dialect_specified</td>
<td>Database dialect @1 is not a valid dialect.</td>
</tr>
<tr>
<td>-901</td>
<td>335544806</td>
<td>valid_db_dialects</td>
<td>Valid database dialects are @1.</td>
</tr>
<tr>
<td>-901</td>
<td>335544811</td>
<td>inv_client_dialect_specified</td>
<td>passed client dialect @1 is not a valid dialect.</td>
</tr>
<tr>
<td>-901</td>
<td>335544812</td>
<td>valid_client_dialects</td>
<td>Valid client dialects are @1.</td>
</tr>
<tr>
<td>SQLCODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544814</td>
<td>service_not_supported</td>
<td>Services functionality will be supported in a later version of the product</td>
</tr>
<tr>
<td>-901</td>
<td>335544820</td>
<td>invalid_savepoint</td>
<td>Unable to find savepoint with name @1 in transaction context</td>
</tr>
<tr>
<td>-901</td>
<td>335544835</td>
<td>bad_shutdown_mode</td>
<td>Target shutdown mode is invalid for database &quot;@1&quot;</td>
</tr>
<tr>
<td>-901</td>
<td>335544840</td>
<td>no_update</td>
<td>cannot update</td>
</tr>
<tr>
<td>-901</td>
<td>335544842</td>
<td>stack_trace</td>
<td>@1</td>
</tr>
<tr>
<td>-901</td>
<td>335544843</td>
<td>ctx_var_not_found</td>
<td>Context variable @1 is not found in namespace @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544844</td>
<td>ctx_namespace_invalid</td>
<td>Invalid namespace name @1 passed to @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544845</td>
<td>ctx_too_big</td>
<td>Too many context variables</td>
</tr>
<tr>
<td>-901</td>
<td>335544846</td>
<td>ctx_bad_argument</td>
<td>Invalid argument passed to @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544847</td>
<td>identifier_too_long</td>
<td>BLR syntax error. Identifier @1... is too long</td>
</tr>
<tr>
<td>-901</td>
<td>335544859</td>
<td>invalid_time_precision</td>
<td>Time precision exceeds allowed range (0-@1)</td>
</tr>
<tr>
<td>-901</td>
<td>335544866</td>
<td>met_wrong_gtt_scope</td>
<td>@1 cannot depend on @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544868</td>
<td>illegal_prc_type</td>
<td>Procedure @1 is not selectable (it does not contain a SUSPEND statement)</td>
</tr>
<tr>
<td>-901</td>
<td>335544869</td>
<td>invalid_sort_datatype</td>
<td>Datatype @1 is not supported for sorting operation</td>
</tr>
<tr>
<td>-901</td>
<td>335544870</td>
<td>collation_name</td>
<td>COLLATION @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544871</td>
<td>domain_name</td>
<td>DOMAIN @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544874</td>
<td>max_db_per_trans_allowed</td>
<td>A multi database transaction cannot span more than @1 databases</td>
</tr>
<tr>
<td>-901</td>
<td>335544876</td>
<td>bad_proc_BLR</td>
<td>Error while parsing procedure @1’s BLR</td>
</tr>
<tr>
<td>-901</td>
<td>335544877</td>
<td>key_too_big</td>
<td>index key too big</td>
</tr>
<tr>
<td>-901</td>
<td>335544885</td>
<td>bad_teb_form</td>
<td>Invalid TEB format</td>
</tr>
<tr>
<td>-901</td>
<td>335544886</td>
<td>tpb_multiple_txn_isolation</td>
<td>Found more than one transaction isolation in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544887</td>
<td>tpb_reserv_before_table</td>
<td>Table reservation lock type @1 requires table name before in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544888</td>
<td>tpb_multiple_spec</td>
<td>Found more than one @1 specification in TPB</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>---------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544889</td>
<td>tpb_option_without_rc</td>
<td>Option @1 requires READ COMMITTED isolation in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544890</td>
<td>tpb_conflicting_options</td>
<td>Option @1 is not valid if @2 was used previously in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544891</td>
<td>tpb_reserv_missing_tlen</td>
<td>Table name length missing after table reservation @1 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544892</td>
<td>tpb_reserv_long_tlen</td>
<td>Table name length @1 is too long after table reservation @2 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544893</td>
<td>tpb_reserv_missing_tname</td>
<td>Table name length @1 without table name after table reservation @2 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544894</td>
<td>tpb_reserv_corrup_tlen</td>
<td>Table name length @1 goes beyond the remaining TPB size after table reservation @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544895</td>
<td>tpb_reserv_null_tlen</td>
<td>Table name length is zero after table reservation @1 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544896</td>
<td>tpb_reserv_relnotfound</td>
<td>Table or view @1 not defined in system tables after table reservation @2 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544897</td>
<td>tpb_reserv_baserelnotfound</td>
<td>Base table or view @1 for view @2 not defined in system tables after table reservation @3 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544898</td>
<td>tpb_missing_len</td>
<td>Option length missing after option @1 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544899</td>
<td>tpb_missing_value</td>
<td>Option length @1 without value after option @2 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544900</td>
<td>tpb_corrupt_len</td>
<td>Option length @1 goes beyond the remaining TPB size after option @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544901</td>
<td>tpb_null_len</td>
<td>Option length is zero after table reservation @1 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544902</td>
<td>tpb_overflow_len</td>
<td>Option length @1 exceeds the range for option @2 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544903</td>
<td>tpb_invalid_value</td>
<td>Option value @1 is invalid for the option @2 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544904</td>
<td>tpb_reserv_stronger_wng</td>
<td>Preserving previous table reservation @1 for table @2, stronger than new @3 in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544905</td>
<td>tpb_reserv_stronger</td>
<td>Table reservation @1 for table @2 already specified and is stronger than new @3 in TPB</td>
</tr>
<tr>
<td>SQL CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544906</td>
<td>tpb_reserv_max_recursion</td>
<td>Table reservation reached maximum recursion of @1 when expanding views in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544907</td>
<td>tpb_reserv_virtualtbl</td>
<td>Table reservation in TPB cannot be applied to @1 because it's a virtual table</td>
</tr>
<tr>
<td>-901</td>
<td>335544908</td>
<td>tpb_reserv_systbl</td>
<td>Table reservation in TPB cannot be applied to @1 because it's a system table</td>
</tr>
<tr>
<td>-901</td>
<td>335544909</td>
<td>tpb_reserv_temptbl</td>
<td>Table reservation @1 or @2 in TPB cannot be applied to @3 because it's a temporary table</td>
</tr>
<tr>
<td>-901</td>
<td>335544910</td>
<td>tpb_readtxn_after_writelock</td>
<td>Cannot set the transaction in read only mode after a table reservation isc_tpb_lock_write in TPB</td>
</tr>
<tr>
<td>-901</td>
<td>335544911</td>
<td>tpb_writelock_after_readtxn</td>
<td>Cannot take a table reservation isc_tpb_lock_write in TPB because the transaction is in read only mode</td>
</tr>
<tr>
<td>-901</td>
<td>335544917</td>
<td>shutdown_timeout</td>
<td>Firebird shutdown is still in progress after the specified timeout</td>
</tr>
<tr>
<td>-901</td>
<td>335544918</td>
<td>att_handle_busy</td>
<td>Attachment handle is busy</td>
</tr>
<tr>
<td>-901</td>
<td>335544919</td>
<td>bad_udf_freeit</td>
<td>Bad written UDF detected: pointer returned in FREE_IT function was not allocated by ib_util_malloc</td>
</tr>
<tr>
<td>-901</td>
<td>335544920</td>
<td>eds_provider_not_found</td>
<td>External Data Source provider '@1' not found</td>
</tr>
<tr>
<td>-901</td>
<td>335544921</td>
<td>eds_connection</td>
<td>Execute statement error at @1 : @2Data source : @3</td>
</tr>
<tr>
<td>-901</td>
<td>335544922</td>
<td>eds_preprocess</td>
<td>Execute statement preprocess SQL error</td>
</tr>
<tr>
<td>-901</td>
<td>335544923</td>
<td>eds_stmt_expected</td>
<td>Statement expected</td>
</tr>
<tr>
<td>-901</td>
<td>335544924</td>
<td>eds_prm_name_expected</td>
<td>Parameter name expected</td>
</tr>
<tr>
<td>-901</td>
<td>335544925</td>
<td>eds_unclosed_comment</td>
<td>Unclosed comment found near '@1'</td>
</tr>
<tr>
<td>-901</td>
<td>335544926</td>
<td>eds_statement</td>
<td>Execute statement error at @1 : @2Statement : @3 Data source : @4</td>
</tr>
<tr>
<td>-901</td>
<td>335544927</td>
<td>eds_input_prm_mismatch</td>
<td>Input parameters mismatch</td>
</tr>
<tr>
<td>-901</td>
<td>335544928</td>
<td>eds_output_prm_mismatch</td>
<td>Output parameters mismatch</td>
</tr>
<tr>
<td>-901</td>
<td>335544929</td>
<td>eds_input_prm_not_set</td>
<td>Input parameter '@1' have no value set</td>
</tr>
<tr>
<td>-901</td>
<td>335544933</td>
<td>nothing_to_cancel</td>
<td>nothing to cancel</td>
</tr>
<tr>
<td>SQL- CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335544934</td>
<td>ibutil_not_loaded</td>
<td>ib_util library has not been loaded to deallocate memory returned by FREE_IT function</td>
</tr>
<tr>
<td>-901</td>
<td>335544973</td>
<td>bad_epb_form</td>
<td>Unrecognized events block</td>
</tr>
<tr>
<td>-901</td>
<td>335544982</td>
<td>udf_fp_overflow</td>
<td>Floating point overflow in result from UDF @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544983</td>
<td>udf_fp_nan</td>
<td>Invalid floating point value returned by UDF @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544985</td>
<td>out_of_temp_space</td>
<td>No free space found in temporary directories</td>
</tr>
<tr>
<td>-901</td>
<td>335544986</td>
<td>eds_expl_trn_ctrl</td>
<td>Explicit transaction control is not allowed</td>
</tr>
<tr>
<td>-901</td>
<td>335544988</td>
<td>package_name</td>
<td>PACKAGE @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544989</td>
<td>cannot_make_not_null</td>
<td>Cannot make field @1 of table @2 NOT NULL because there are NULLs present</td>
</tr>
<tr>
<td>-901</td>
<td>335544990</td>
<td>feature_removed</td>
<td>Feature @1 is not supported anymore</td>
</tr>
<tr>
<td>-901</td>
<td>335544991</td>
<td>view_name</td>
<td>VIEW @1</td>
</tr>
<tr>
<td>-901</td>
<td>335544993</td>
<td>invalid_fetch_option</td>
<td>Fetch option @1 is invalid for a non-scrollable cursor</td>
</tr>
<tr>
<td>-901</td>
<td>335544994</td>
<td>bad_fun_BLR</td>
<td>Error while parsing function @1’s BLR</td>
</tr>
<tr>
<td>-901</td>
<td>335544995</td>
<td>func_pack_not_implemented</td>
<td>Cannot execute function @1 of the unimplemented package @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544996</td>
<td>proc_pack_not_implemented</td>
<td>Cannot execute procedure @1 of the unimplemented package @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544997</td>
<td>eem_func_not_returned</td>
<td>External function @1 not returned by the external engine plugin @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544998</td>
<td>eem_proc_not_returned</td>
<td>External procedure @1 not returned by the external engine plugin @2</td>
</tr>
<tr>
<td>-901</td>
<td>335544999</td>
<td>eem_trig_not_returned</td>
<td>External trigger @1 not returned by the external engine plugin @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545000</td>
<td>eem_bad_plugin_ver</td>
<td>Incompatible plugin version @1 for external engine @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545001</td>
<td>eem_engine_notfound</td>
<td>External engine @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>335545004</td>
<td>pman_cannot_load_plugin</td>
<td>Error loading plugin @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545005</td>
<td>pman_module_notfound</td>
<td>Loadable module @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>335545006</td>
<td>pman_entrypoint_notfound</td>
<td>Standard plugin entrypoint does not exist in module @1</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545007</td>
<td>pman_module_bad</td>
<td>Module @1 exists but can not be loaded</td>
</tr>
<tr>
<td>-901</td>
<td>335545008</td>
<td>pman_plugin_notfound</td>
<td>Module @1 does not contain plugin @2 type @3</td>
</tr>
<tr>
<td>-901</td>
<td>335545010</td>
<td>unexpected_null</td>
<td>Value is NULL but isNull parameter was not informed</td>
</tr>
<tr>
<td>-901</td>
<td>335545011</td>
<td>type_notcompat_blob</td>
<td>Type @1 is incompatible with BLOB</td>
</tr>
<tr>
<td>-901</td>
<td>335545012</td>
<td>invalid_date_val</td>
<td>Invalid date</td>
</tr>
<tr>
<td>-901</td>
<td>335545013</td>
<td>invalid_time_val</td>
<td>Invalid time</td>
</tr>
<tr>
<td>-901</td>
<td>335545014</td>
<td>invalid_timestamp_val</td>
<td>Invalid timestamp</td>
</tr>
<tr>
<td>-901</td>
<td>335545015</td>
<td>invalid_index_val</td>
<td>Invalid index @1 in function @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545018</td>
<td>private_function</td>
<td>Function @1 is private to package @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545019</td>
<td>private_procedure</td>
<td>Procedure @1 is private to package @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545021</td>
<td>bad_events_handle</td>
<td>invalid events id (handle)</td>
</tr>
<tr>
<td>-901</td>
<td>335545025</td>
<td>spb_no_id</td>
<td>missing service ID in spb</td>
</tr>
<tr>
<td>-901</td>
<td>335545026</td>
<td>ee_blr_mismatch_null</td>
<td>External BLR message mismatch: invalid null descriptor at field @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545027</td>
<td>ee_blr_mismatch_length</td>
<td>External BLR message mismatch: length = @1, expected @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545031</td>
<td>libtommath_generic</td>
<td>Libtommath error code @1 in function @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545041</td>
<td>cp_process_active</td>
<td>Crypt failed - already crypting database</td>
</tr>
<tr>
<td>-901</td>
<td>335545042</td>
<td>cp_already_crypted</td>
<td>Crypt failed - database is already in requested state</td>
</tr>
<tr>
<td>-901</td>
<td>335545047</td>
<td>ee_blr_mismatch_names_count</td>
<td>External BLR message mismatch: names count = @1, blr count = @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545048</td>
<td>ee_blr_mismatch_name_not_found</td>
<td>External BLR message mismatch: name @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>335545049</td>
<td>bad_result_set</td>
<td>Invalid resultset interface</td>
</tr>
<tr>
<td>-901</td>
<td>335545059</td>
<td>badvarnum</td>
<td>undefined variable number</td>
</tr>
<tr>
<td>-901</td>
<td>335545071</td>
<td>info_unprepared_stmt</td>
<td>Attempt to get information about an unprepared dynamic SQL statement.</td>
</tr>
<tr>
<td>-901</td>
<td>335545072</td>
<td>idx_key_value</td>
<td>Problematic key value is @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545073</td>
<td>forupdate_virtualtbl</td>
<td>Cannot select virtual table @1 for update WITH LOCK</td>
</tr>
<tr>
<td>-901</td>
<td>335545074</td>
<td>forupdate_systbl</td>
<td>Cannot select system table @1 for update WITH LOCK</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545075</td>
<td>forupdate_temptbl</td>
<td>Cannot select temporary table @1 for update WITH LOCK</td>
</tr>
<tr>
<td>-901</td>
<td>335545076</td>
<td>cant_modify_sysobj</td>
<td>System @1 @2 cannot be modified</td>
</tr>
<tr>
<td>-901</td>
<td>335545077</td>
<td>server_misconfigured</td>
<td>Server misconfigured - contact administrator please</td>
</tr>
<tr>
<td>-901</td>
<td>335545078</td>
<td>alter_role</td>
<td>Deprecated backward compatibility ALTER ROLE ... SET/DROP AUTO ADMIN mapping may be used only for RDB$ADMIN role</td>
</tr>
<tr>
<td>-901</td>
<td>335545079</td>
<td>map_already_exists</td>
<td>Mapping @1 already exists</td>
</tr>
<tr>
<td>-901</td>
<td>335545080</td>
<td>map_not_exists</td>
<td>Mapping @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>335545081</td>
<td>map_load</td>
<td>@1 failed when loading mapping cache</td>
</tr>
<tr>
<td>-901</td>
<td>335545082</td>
<td>map_aster</td>
<td>Invalid name &lt;*&gt; in authentication block</td>
</tr>
<tr>
<td>-901</td>
<td>335545083</td>
<td>map_multi</td>
<td>Multiple maps found for @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545084</td>
<td>map_undefined</td>
<td>Undefined mapping result - more than one different results found</td>
</tr>
<tr>
<td>-901</td>
<td>335545088</td>
<td>map_nodb</td>
<td>Global mapping is not available when database @1 is not present</td>
</tr>
<tr>
<td>-901</td>
<td>335545089</td>
<td>map_notable</td>
<td>Global mapping is not available when table RDB$MAP is not present in database @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545090</td>
<td>miss_trusted_role</td>
<td>Your attachment has no trusted role</td>
</tr>
<tr>
<td>-901</td>
<td>335545091</td>
<td>set_invalid_role</td>
<td>Role @1 is invalid or unavailable</td>
</tr>
<tr>
<td>-901</td>
<td>335545093</td>
<td>dup_attribute</td>
<td>Duplicated user attribute @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545094</td>
<td>dyn_no_priv</td>
<td>There is no privilege for this operation</td>
</tr>
<tr>
<td>-901</td>
<td>335545095</td>
<td>dsql_cant_grant_option</td>
<td>Using GRANT OPTION on @1 not allowed</td>
</tr>
<tr>
<td>-901</td>
<td>335545097</td>
<td>crdb_load</td>
<td>@1 failed when working with CREATE DATABASE grants</td>
</tr>
<tr>
<td>-901</td>
<td>335545098</td>
<td>crdb_nodb</td>
<td>CREATE DATABASE grants check is not possible when database @1 is not present</td>
</tr>
<tr>
<td>-901</td>
<td>335545099</td>
<td>crdb_notable</td>
<td>CREATE DATABASE grants check is not possible when table RDB$DB_CREATORS is not present in database @1</td>
</tr>
</tbody>
</table>

Appendix B: Обработка ошибок, коды и сообщения
<table>
<thead>
<tr>
<th>SQL-CODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>-901</td>
<td>335545102</td>
<td>savepoint_backout_err</td>
<td>Error during savepoint backout - transaction invalidated</td>
</tr>
<tr>
<td>-901</td>
<td>335545105</td>
<td>map_down</td>
<td>Some database(s) were shutdown when trying to read mapping data</td>
</tr>
<tr>
<td>-901</td>
<td>335545109</td>
<td>encrypt_error</td>
<td>Page requires encryption but crypt plugin is missing</td>
</tr>
<tr>
<td>-901</td>
<td>335545111</td>
<td>wrong_prvlg</td>
<td>System privilege @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>335545115</td>
<td>no_cursor</td>
<td>Cannot open cursor for non-SELECT statement</td>
</tr>
<tr>
<td>-901</td>
<td>335545127</td>
<td>cfg_stmt_timeout</td>
<td>Config level timeout expired.</td>
</tr>
<tr>
<td>-901</td>
<td>335545128</td>
<td>att_stmt_timeout</td>
<td>Attachment level timeout expired.</td>
</tr>
<tr>
<td>-901</td>
<td>335545129</td>
<td>req_stmt_timeout</td>
<td>Statement level timeout expired.</td>
</tr>
<tr>
<td>-901</td>
<td>335545139</td>
<td>decfloat_divide_by_zero</td>
<td>Decimal float divide by zero. The code attempted to divide a DECFLOAT value by zero.</td>
</tr>
<tr>
<td>-901</td>
<td>335545140</td>
<td>decfloat_inexact_result</td>
<td>Decimal float inexact result. The result of an operation cannot be represented as a decimal fraction.</td>
</tr>
<tr>
<td>-901</td>
<td>335545141</td>
<td>decfloat_invalid_operation</td>
<td>Decimal float invalid operation. An indeterminant error occurred during an operation.</td>
</tr>
<tr>
<td>-901</td>
<td>335545142</td>
<td>decfloat_overflow</td>
<td>Decimal float overflow. The exponent of a result is greater than the magnitude allowed.</td>
</tr>
<tr>
<td>-901</td>
<td>335545143</td>
<td>decfloat_underflow</td>
<td>Decimal float underflow. The exponent of a result is less than the magnitude allowed.</td>
</tr>
<tr>
<td>-901</td>
<td>335545144</td>
<td>subfunc_notdef</td>
<td>Sub-function @1 has not been defined</td>
</tr>
<tr>
<td>-901</td>
<td>335545145</td>
<td>subproc_notdef</td>
<td>Sub-procedure @1 has not been defined</td>
</tr>
<tr>
<td>-901</td>
<td>335545146</td>
<td>subfunc_signat</td>
<td>Sub-function @1 has a signature mismatch with its forward declaration</td>
</tr>
<tr>
<td>-901</td>
<td>335545147</td>
<td>subproc_signat</td>
<td>Sub-procedure @1 has a signature mismatch with its forward declaration</td>
</tr>
<tr>
<td>-901</td>
<td>335545148</td>
<td>subfunc_defvaldecl</td>
<td>Default values for parameters are not allowed in definition of the previously declared sub-function @1</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545149</td>
<td>subproc_defvaldecl</td>
<td>Default values for parameters are not allowed in definition of the previously declared sub-procedure @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545150</td>
<td>subfunc_not_impl</td>
<td>Sub-function @1 was declared but not implemented</td>
</tr>
<tr>
<td>-901</td>
<td>335545151</td>
<td>subproc_not_impl</td>
<td>Sub-procedure @1 was declared but not implemented</td>
</tr>
<tr>
<td>-901</td>
<td>335545152</td>
<td>sysf_invalid_hash_algorithm</td>
<td>Invalid HASH algorithm @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545153</td>
<td>expression_eval_index</td>
<td>Expression evaluation error for index &quot;@1&quot; on table &quot;@2&quot;</td>
</tr>
<tr>
<td>-901</td>
<td>335545154</td>
<td>invalid_decfloat_trap</td>
<td>Invalid decfloat trap state @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545155</td>
<td>invalid_decfloat_round</td>
<td>Invalid decfloat rounding mode @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545159</td>
<td>bad_batch_handle</td>
<td>invalid batch handle</td>
</tr>
<tr>
<td>-901</td>
<td>335545160</td>
<td>intl_char</td>
<td>Bad international character in tag @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545161</td>
<td>null_block</td>
<td>Null data in parameters block with non-zero length</td>
</tr>
<tr>
<td>-901</td>
<td>335545162</td>
<td>mixed_info</td>
<td>Items working with running service and getting generic server information should not be mixed in single info block</td>
</tr>
<tr>
<td>-901</td>
<td>335545163</td>
<td>unknown_info</td>
<td>Unknown information item, code @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545164</td>
<td>bpb_version</td>
<td>Wrong version of blob parameters block @1, should be @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545165</td>
<td>user_manager</td>
<td>User management plugin is missing or failed to load</td>
</tr>
<tr>
<td>-901</td>
<td>335545168</td>
<td>metadata_name</td>
<td>Name @1 not found in system MetadataBuilder</td>
</tr>
<tr>
<td>-901</td>
<td>335545169</td>
<td>tokens_parse</td>
<td>Parse to tokens error</td>
</tr>
<tr>
<td>-901</td>
<td>335545171</td>
<td>batch_compl_range</td>
<td>Message @1 is out of range, only @2 messages in batch</td>
</tr>
<tr>
<td>-901</td>
<td>335545172</td>
<td>batch_compl_detail</td>
<td>Detailed error info for message @1 is missing in batch</td>
</tr>
<tr>
<td>-901</td>
<td>335545175</td>
<td>big_segment</td>
<td>Segment size (@1) should not exceed 65535 (64K - 1) when using segmented blob</td>
</tr>
<tr>
<td>-901</td>
<td>335545176</td>
<td>batch_policy</td>
<td>Invalid blob policy in the batch for @1() call</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545177</td>
<td>batch_defbpb</td>
<td>Can't change default BPB after adding any data to batch</td>
</tr>
<tr>
<td>-901</td>
<td>335545178</td>
<td>batch_align</td>
<td>Unexpected info buffer structure querying for default blob alignment</td>
</tr>
<tr>
<td>-901</td>
<td>335545179</td>
<td>multi_segment_dup</td>
<td>Duplicated segment @1 in multisegment connect block parameter</td>
</tr>
<tr>
<td>-901</td>
<td>335545181</td>
<td>message_format</td>
<td>Error parsing message format</td>
</tr>
<tr>
<td>-901</td>
<td>335545182</td>
<td>batch_param_version</td>
<td>Wrong version of batch parameters block @1, should be @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545183</td>
<td>batch_msg_long</td>
<td>Message size (@1) in batch exceeds internal buffer size (@2)</td>
</tr>
<tr>
<td>-901</td>
<td>335545184</td>
<td>batch_open</td>
<td>Batch already opened for this statement</td>
</tr>
<tr>
<td>-901</td>
<td>335545185</td>
<td>batch_type</td>
<td>Invalid type of statement used in batch</td>
</tr>
<tr>
<td>-901</td>
<td>335545186</td>
<td>batch_param</td>
<td>Statement used in batch must have parameters</td>
</tr>
<tr>
<td>-901</td>
<td>335545187</td>
<td>batch_blobs</td>
<td>There are no blobs in associated with batch statement</td>
</tr>
<tr>
<td>-901</td>
<td>335545188</td>
<td>batch_blob_append</td>
<td>appendBlobData() is used to append data to last blob but no such blob was added to the batch</td>
</tr>
<tr>
<td>-901</td>
<td>335545189</td>
<td>batch_stream_align</td>
<td>Portions of data, passed as blob stream, should have size multiple to the alignment required for blobs</td>
</tr>
<tr>
<td>-901</td>
<td>335545190</td>
<td>batch_rpt_blob</td>
<td>Repeated blob id @1 in registerBlob()</td>
</tr>
<tr>
<td>-901</td>
<td>335545191</td>
<td>batch_blob_buf</td>
<td>Blob buffer format error</td>
</tr>
<tr>
<td>-901</td>
<td>335545192</td>
<td>batch_small_data</td>
<td>Unusable (too small) data remained in @1 buffer</td>
</tr>
<tr>
<td>-901</td>
<td>335545193</td>
<td>batch_cont_bpib</td>
<td>Blob continuation should not contain BPB</td>
</tr>
<tr>
<td>-901</td>
<td>335545194</td>
<td>batch_big_bpib</td>
<td>Size of BPB (@1) greater than remaining data (@2)</td>
</tr>
<tr>
<td>-901</td>
<td>335545195</td>
<td>batch_big_segment</td>
<td>Size of segment (@1) greater than current BLOB data (@2)</td>
</tr>
<tr>
<td>-901</td>
<td>335545196</td>
<td>batch_big_seg2</td>
<td>Size of segment (@1) greater than available data (@2)</td>
</tr>
<tr>
<td>-901</td>
<td>335545197</td>
<td>batch_blob_id</td>
<td>Unknown blob ID @1 in the batch message</td>
</tr>
<tr>
<td>SQL CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545198</td>
<td>batch_too_big</td>
<td>Internal buffer overflow - batch too big</td>
</tr>
<tr>
<td>-901</td>
<td>335545199</td>
<td>num_literal</td>
<td>Numeric literal too long</td>
</tr>
<tr>
<td>-901</td>
<td>335545202</td>
<td>hdr_overflow</td>
<td>Header page overflow - too many clumplets on it</td>
</tr>
<tr>
<td>-901</td>
<td>335545203</td>
<td>vld_plugins</td>
<td>No matching client/server authentication plugins configured for execute statement in embedded datasource</td>
</tr>
<tr>
<td>-901</td>
<td>335545206</td>
<td>ses_reset_err</td>
<td>Cannot reset user session</td>
</tr>
<tr>
<td>-901</td>
<td>335545207</td>
<td>ses_reset_open_trans</td>
<td>There are open transactions (@1 active)</td>
</tr>
<tr>
<td>-901</td>
<td>335545208</td>
<td>ses_reset_warn</td>
<td>Session was reset with warning(s)</td>
</tr>
<tr>
<td>-901</td>
<td>335545209</td>
<td>ses_reset_tran_rollback</td>
<td>Transaction is rolled back due to session reset, all changes are lost</td>
</tr>
<tr>
<td>-901</td>
<td>335545210</td>
<td>plugin_name</td>
<td>Plugin @1:</td>
</tr>
<tr>
<td>-901</td>
<td>335545211</td>
<td>parameter_name</td>
<td>PARAMETER @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545212</td>
<td>file_starting_page_err</td>
<td>Starting page number for file @1 must be @2 or greater</td>
</tr>
<tr>
<td>-901</td>
<td>335545213</td>
<td>invalid_timezone_offset</td>
<td>Invalid time zone offset: @1 - must use format +/-hours:minutes and be between -14:00 and +14:00</td>
</tr>
<tr>
<td>-901</td>
<td>335545214</td>
<td>invalid_timezone_region</td>
<td>Invalid time zone region: @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545215</td>
<td>invalid_timezone_id</td>
<td>Invalid time zone ID: @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545216</td>
<td>tom_decode64len</td>
<td>Wrong base64 text length @1, should be multiple of 4</td>
</tr>
<tr>
<td>-901</td>
<td>335545217</td>
<td>tom_strblob</td>
<td>Invalid first parameter datatype - need string or blob</td>
</tr>
<tr>
<td>-901</td>
<td>335545218</td>
<td>tom_reg</td>
<td>Error registering @1 - probably bad tomcrypt library</td>
</tr>
<tr>
<td>-901</td>
<td>335545219</td>
<td>tom_algorithm</td>
<td>Unknown crypt algorithm @1 in USING clause</td>
</tr>
<tr>
<td>-901</td>
<td>335545220</td>
<td>tom_mode_miss</td>
<td>Should specify mode parameter for symmetric cipher</td>
</tr>
<tr>
<td>-901</td>
<td>335545221</td>
<td>tom_mode_bad</td>
<td>Unknown symmetric crypt mode specified</td>
</tr>
<tr>
<td>-901</td>
<td>335545222</td>
<td>tom_no_mode</td>
<td>Mode parameter makes no sense for chosen cipher</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545223</td>
<td>tom_iv_miss</td>
<td>Should specify initialization vector (IV) for chosen cipher and/or mode</td>
</tr>
<tr>
<td>-901</td>
<td>335545224</td>
<td>tom_no_iv</td>
<td>Initialization vector (IV) makes no sense for chosen cipher and/or mode</td>
</tr>
<tr>
<td>-901</td>
<td>335545225</td>
<td>tom_ctrtype_bad</td>
<td>Invalid counter endianess @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545226</td>
<td>tom_no_ctrtype</td>
<td>Counter endianess parameter is not used in mode @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545227</td>
<td>tom_ctr_big</td>
<td>Too big counter value @1, maximum @2 can be used</td>
</tr>
<tr>
<td>-901</td>
<td>335545228</td>
<td>tom_no_ctr</td>
<td>Counter length/value parameter is not used with @1 @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545229</td>
<td>tom_iv_length</td>
<td>Invalid initialization vector (IV) length @1, need @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545230</td>
<td>tom_error</td>
<td>TomCrypt library error: @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545231</td>
<td>tom_yarrow_start</td>
<td>Starting PRNG yarrow</td>
</tr>
<tr>
<td>-901</td>
<td>335545232</td>
<td>tom_yarrow_setup</td>
<td>Setting up PRNG yarrow</td>
</tr>
<tr>
<td>-901</td>
<td>335545233</td>
<td>tom_init_mode</td>
<td>Initializing @1 mode</td>
</tr>
<tr>
<td>-901</td>
<td>335545234</td>
<td>tom_crypt_mode</td>
<td>Encrypting in @1 mode</td>
</tr>
<tr>
<td>-901</td>
<td>335545235</td>
<td>tom_decrypt_mode</td>
<td>Decrypting in @1 mode</td>
</tr>
<tr>
<td>-901</td>
<td>335545236</td>
<td>tom_init_cip</td>
<td>Initializing cipher @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545237</td>
<td>tom_crypt_cip</td>
<td>Encrypting using cipher @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545238</td>
<td>tom_decrypt_cip</td>
<td>Decrypting using cipher @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545239</td>
<td>tom_setup_cip</td>
<td>Setting initialization vector (IV) for @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545240</td>
<td>tom_setup_chacha</td>
<td>Invalid initialization vector (IV) length @1, need 8 or 12</td>
</tr>
<tr>
<td>-901</td>
<td>335545241</td>
<td>tom_encode</td>
<td>Encoding @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545242</td>
<td>tom_decode</td>
<td>Decoding @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545243</td>
<td>tom_rsa_import</td>
<td>Importing RSA key</td>
</tr>
<tr>
<td>-901</td>
<td>335545244</td>
<td>tom_oaep</td>
<td>Invalid OAEP packet</td>
</tr>
<tr>
<td>-901</td>
<td>335545245</td>
<td>tom_hash_bad</td>
<td>Unknown hash algorithm @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545246</td>
<td>tom_rsa_make</td>
<td>Making RSA key</td>
</tr>
<tr>
<td>-901</td>
<td>335545247</td>
<td>tom_rsa_export</td>
<td>Exporting @1 RSA key</td>
</tr>
<tr>
<td>-901</td>
<td>335545248</td>
<td>tom_rsa_sign</td>
<td>RSA-signing data</td>
</tr>
<tr>
<td>-901</td>
<td>335545249</td>
<td>tom_rsa_verify</td>
<td>Verifying RSA-signed data</td>
</tr>
<tr>
<td>SQL-Code</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>----------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>335545250</td>
<td>tom_chacha_key</td>
<td>Invalid key length @1, need 16 or 32</td>
</tr>
<tr>
<td>-901</td>
<td>335545251</td>
<td>bad_repl_handle</td>
<td>invalid replicator handle</td>
</tr>
<tr>
<td>-901</td>
<td>335545252</td>
<td>tra_snapshot_does_not_exist</td>
<td>Transaction’s base snapshot number does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>335545253</td>
<td>eds_input_prm_not_used</td>
<td>Input parameter ’@1‘ is not used in SQL query text</td>
</tr>
<tr>
<td>-901</td>
<td>335545255</td>
<td>invalid_time_zone_bind</td>
<td>Invalid time zone bind mode @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545256</td>
<td>invalid_decfloat_bind</td>
<td>Invalid decfloat bind mode @1</td>
</tr>
<tr>
<td>-901</td>
<td>335545257</td>
<td>odd_hex_len</td>
<td>Invalid hex text length @1, should be multiple of 2</td>
</tr>
<tr>
<td>-901</td>
<td>335545258</td>
<td>invalid_hex_digit</td>
<td>Invalid hex digit @1 at position @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545261</td>
<td>bind_convert</td>
<td>Can not convert @1 to @2</td>
</tr>
<tr>
<td>-901</td>
<td>335545264</td>
<td>dyn_no_create_priv</td>
<td>No permission for CREATE @1 operation</td>
</tr>
<tr>
<td>-901</td>
<td>335545265</td>
<td>suspend_without_returns</td>
<td>SUSPEND could not be used without RETURNS clause in PROCEDURE or EXECUTE BLOCK</td>
</tr>
<tr>
<td>-901</td>
<td>335545274</td>
<td>tom_key_length</td>
<td>Invalid key length @1, need &gt;@2</td>
</tr>
<tr>
<td>-901</td>
<td>336068645</td>
<td>dyn_filter_not_found</td>
<td>BLOB Filter @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068649</td>
<td>dyn_func_not_found</td>
<td>Function @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068656</td>
<td>dyn_index_not_found</td>
<td>Index not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068662</td>
<td>dyn_view_not_found</td>
<td>View @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068697</td>
<td>dyn_domain_not_found</td>
<td>Domain not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068717</td>
<td>dyn_cant_modify_auto_trig</td>
<td>Triggers created automatically cannot be modified</td>
</tr>
<tr>
<td>-901</td>
<td>336068740</td>
<td>dyn_dup_table</td>
<td>Table @1 already exists</td>
</tr>
<tr>
<td>-901</td>
<td>336068748</td>
<td>dyn_proc_not_found</td>
<td>Procedure @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068752</td>
<td>dyn_exception_not_found</td>
<td>Exception not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068754</td>
<td>dyn_proc_param_not_found</td>
<td>Parameter @1 in procedure @2 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068755</td>
<td>dyn_trig_not_found</td>
<td>Trigger @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068759</td>
<td>dyn_charset_not_found</td>
<td>Character set @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068760</td>
<td>dyn_collation_not_found</td>
<td>Collation @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068763</td>
<td>dyn_role_not_found</td>
<td>Role @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068767</td>
<td>dyn_name_longer</td>
<td>Name longer than database column size</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>336068784</td>
<td>dyn_column_does_not_exist</td>
<td>column @1 does not exist in table/view @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068796</td>
<td>dyn_role_does_not_exist</td>
<td>SQL role @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068797</td>
<td>dyn_no_grant_admin_opt</td>
<td>user @1 has no grant admin option on SQL role @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068798</td>
<td>dyn_user_not_role_member</td>
<td>user @1 is not a member of SQL role @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068799</td>
<td>dyn_delete_role_failed</td>
<td>@1 is not the owner of SQL role @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068800</td>
<td>dyn_grant_role_to_user</td>
<td>@1 is a SQL role and not a user</td>
</tr>
<tr>
<td>-901</td>
<td>336068801</td>
<td>dyn_inv_sql_role_name</td>
<td>user name @1 could not be used for SQL role</td>
</tr>
<tr>
<td>-901</td>
<td>336068802</td>
<td>dyn_dup_sql_role</td>
<td>SQL role @1 already exists</td>
</tr>
<tr>
<td>-901</td>
<td>336068803</td>
<td>dyn_kywd_spec_for_role</td>
<td>keyword @1 can not be used as a SQL role name</td>
</tr>
<tr>
<td>-901</td>
<td>336068804</td>
<td>dyn_roles_not_supported</td>
<td>SQL roles are not supported in on older versions of the database. A backup and restore of the database is required.</td>
</tr>
<tr>
<td>-901</td>
<td>336068820</td>
<td>dyn_zero_len_id</td>
<td>Zero length identifiers are not allowed</td>
</tr>
<tr>
<td>-901</td>
<td>336068822</td>
<td>dyn_gen_not_found</td>
<td>Sequence @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068840</td>
<td>dyn_wrong_gtt_scope</td>
<td>@1 cannot reference @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068843</td>
<td>dyn_coll_used_table</td>
<td>Collation @1 is used in table @2 (field name @3) and cannot be dropped</td>
</tr>
<tr>
<td>-901</td>
<td>336068844</td>
<td>dyn_coll_used_domain</td>
<td>Collation @1 is used in domain @2 and cannot be dropped</td>
</tr>
<tr>
<td>-901</td>
<td>336068846</td>
<td>dyn_cannot_del_def_coll</td>
<td>Cannot delete default collation of CHARACTER SET @1</td>
</tr>
<tr>
<td>-901</td>
<td>336068849</td>
<td>dyn_table_not_found</td>
<td>Table @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068851</td>
<td>dyn_coll_used_procedure</td>
<td>Collation @1 is used in procedure @2 (parameter name @3) and cannot be dropped</td>
</tr>
<tr>
<td>-901</td>
<td>336068856</td>
<td>dyn_ods_not_supp_feature</td>
<td>Feature '@1' is not supported in ODS @2.@3</td>
</tr>
<tr>
<td>-901</td>
<td>336068858</td>
<td>dyn_no_empty_pw</td>
<td>Password should not be empty string</td>
</tr>
<tr>
<td>-901</td>
<td>336068859</td>
<td>dyn_dup_index</td>
<td>Index @1 already exists</td>
</tr>
<tr>
<td>-901</td>
<td>336068864</td>
<td>dyn_package_not_found</td>
<td>Package @1 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068865</td>
<td>dyn_schema_not_found</td>
<td>Schema @1 not found</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>336068871</td>
<td>dyn_funcnotdef_package</td>
<td>Function @1 has not been defined on the package body @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068872</td>
<td>dyn_procnotdef_package</td>
<td>Procedure @1 has not been defined on the package body @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068873</td>
<td>dyn_funcsignat_package</td>
<td>Function @1 has a signature mismatch on package body @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068874</td>
<td>dyn_procsignat_package</td>
<td>Procedure @1 has a signature mismatch on package body @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068875</td>
<td>dyn_defvaldecl_package_proc</td>
<td>Default values for parameters are not allowed in the definition of a previously declared packaged procedure @1.@2</td>
</tr>
<tr>
<td>-901</td>
<td>336068877</td>
<td>dyn_package_body_exists</td>
<td>Package body @1 already exists</td>
</tr>
<tr>
<td>-901</td>
<td>336068879</td>
<td>dyn_newfc_oldsyntax</td>
<td>Cannot alter new style function @1 with ALTER EXTERNAL FUNCTION. Use ALTER FUNCTION instead.</td>
</tr>
<tr>
<td>-901</td>
<td>336068886</td>
<td>dyn_func_param_not_found</td>
<td>Parameter @1 in function @2 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068887</td>
<td>dyn_routine_param_not_found</td>
<td>Parameter @1 of routine @2 not found</td>
</tr>
<tr>
<td>-901</td>
<td>336068888</td>
<td>dyn_routine_param_ambiguous</td>
<td>Parameter @1 of routine @2 is ambiguous (found in both procedures and functions). Use a specifier keyword.</td>
</tr>
<tr>
<td>-901</td>
<td>336068889</td>
<td>dyn_coll_used_function</td>
<td>Collation @1 is used in function @2 (parameter name @3) and cannot be dropped</td>
</tr>
<tr>
<td>-901</td>
<td>336068890</td>
<td>dyn_domain_used_function</td>
<td>Domain @1 is used in function @2 (parameter name @3) and cannot be dropped</td>
</tr>
<tr>
<td>-901</td>
<td>336068891</td>
<td>dyn_alter_user_no_clause</td>
<td>ALTER USER requires at least one clause to be specified</td>
</tr>
<tr>
<td>-901</td>
<td>336068894</td>
<td>dyn_duplicate_package_item</td>
<td>Duplicate @1 @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068895</td>
<td>dyn_cant_modify_sysobj</td>
<td>System @1 @2 cannot be modified</td>
</tr>
<tr>
<td>-901</td>
<td>336068896</td>
<td>dyn_cant_use_zero_increment</td>
<td>INCREMENT BY 0 is an illegal option for sequence @1</td>
</tr>
<tr>
<td>-901</td>
<td>336068897</td>
<td>dyn_cant_use_in_foreignkey</td>
<td>Can’t use @1 in FOREIGN KEY constraint</td>
</tr>
<tr>
<td>-901</td>
<td>336068898</td>
<td>dyn_defvaldecl_package_func</td>
<td>Default values for parameters are not allowed in the definition of a previously declared packaged function @1.@2</td>
</tr>
<tr>
<td>-901</td>
<td>336068900</td>
<td>dyn_cyclic_role</td>
<td>role @1 can not be granted to role @2</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>336068904</td>
<td>dyn_cant_use_zero_inc_ident</td>
<td>INCREMENT BY 0 is an illegal option for identity column @1 of table @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068907</td>
<td>dyn_no_ddl_grant_opt_priv</td>
<td>no @1 privilege with grant option on DDL @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068908</td>
<td>dyn_no_grant_opt_priv</td>
<td>no @1 privilege with grant option on object @2</td>
</tr>
<tr>
<td>-901</td>
<td>336068909</td>
<td>dyn_func_not_existent</td>
<td>Function @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068910</td>
<td>dyn_proc_not_existent</td>
<td>Procedure @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068911</td>
<td>dyn_pack_not_existent</td>
<td>Package @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068912</td>
<td>dyn_trig_not_existent</td>
<td>Trigger @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068913</td>
<td>dyn_view_not_existent</td>
<td>View @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068914</td>
<td>dyn_rel_not_existent</td>
<td>Table @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068915</td>
<td>dyn_exc_not_existent</td>
<td>Exception @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068916</td>
<td>dyn_gen_not_existent</td>
<td>Generator/Sequence @1 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336068917</td>
<td>dyn_fld_not_existent</td>
<td>Field @1 of table @2 does not exist</td>
</tr>
<tr>
<td>-901</td>
<td>336397211</td>
<td>dsq1_too_many_values</td>
<td>Too many values (more than @1) in member list to match against</td>
</tr>
<tr>
<td>-901</td>
<td>336397236</td>
<td>dsq1_unsupp_feature_dialect</td>
<td>feature is not supported in dialect @1</td>
</tr>
<tr>
<td>-901</td>
<td>336397239</td>
<td>dsq1_unsupported_in_auto_trans</td>
<td>@1 is not supported inside IN AUTONOMOUS TRANSACTION block</td>
</tr>
<tr>
<td>-901</td>
<td>336397258</td>
<td>dsq1_alter_charset_failed</td>
<td>ALTER CHARACTER SET @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397259</td>
<td>dsq1_comment_on_failed</td>
<td>COMMENT ON @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397260</td>
<td>dsq1_create_func_failed</td>
<td>CREATE FUNCTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397261</td>
<td>dsq1_alter_func_failed</td>
<td>ALTER FUNCTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397262</td>
<td>dsq1_create_alter_func_failed</td>
<td>CREATE OR ALTER FUNCTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397263</td>
<td>dsq1_drop_func_failed</td>
<td>DROP FUNCTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397264</td>
<td>dsq1_recreate_func_failed</td>
<td>RECREATE FUNCTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397265</td>
<td>dsq1_create_proc_failed</td>
<td>CREATE PROCEDURE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397266</td>
<td>dsq1_alter_proc_failed</td>
<td>ALTER PROCEDURE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397267</td>
<td>dsq1_create_alter_proc_failed</td>
<td>CREATE OR ALTER PROCEDURE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397268</td>
<td>dsq1_drop_proc_failed</td>
<td>DROP PROCEDURE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397269</td>
<td>dsq1_recreate_proc_failed</td>
<td>RECREATE PROCEDURE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397270</td>
<td>dsq1_create_trigger_failed</td>
<td>CREATE TRIGGER @1 failed</td>
</tr>
</tbody>
</table>

Appendix B: Обработка ошибок, коды и сообщения
<table>
<thead>
<tr>
<th>SQL-CODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>-901</td>
<td>336397271</td>
<td>dsq1_alter_trigger_failed</td>
<td>ALTER TRIGGER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397272</td>
<td>dsq1_create_alter_trigger_failed</td>
<td>CREATE OR ALTER TRIGGER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397273</td>
<td>dsq1_drop_trigger_failed</td>
<td>DROP TRIGGER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397274</td>
<td>dsq1_recreate_trigger_failed</td>
<td>RECREATE TRIGGER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397275</td>
<td>dsq1_create_collation_failed</td>
<td>CREATE COLLATION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397276</td>
<td>dsq1_drop_collation_failed</td>
<td>DROP COLLATION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397277</td>
<td>dsq1_create_domain_failed</td>
<td>CREATE DOMAIN @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397278</td>
<td>dsq1_alter_domain_failed</td>
<td>ALTER DOMAIN @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397279</td>
<td>dsq1_drop_domain_failed</td>
<td>DROP DOMAIN @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397280</td>
<td>dsq1_create_except_failed</td>
<td>CREATE EXCEPTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397281</td>
<td>dsq1_alter_except_failed</td>
<td>ALTER EXCEPTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397282</td>
<td>dsq1_create_alter_except_failed</td>
<td>CREATE OR ALTER EXCEPTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397283</td>
<td>dsq1_recreate_except_failed</td>
<td>RECREATE EXCEPTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397284</td>
<td>dsq1_drop_except_failed</td>
<td>DROP EXCEPTION @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397285</td>
<td>dsq1_create_sequence_failed</td>
<td>CREATE SEQUENCE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397286</td>
<td>dsq1_create_table_failed</td>
<td>CREATE TABLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397287</td>
<td>dsq1_alter_table_failed</td>
<td>ALTER TABLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397288</td>
<td>dsq1_drop_table_failed</td>
<td>DROP TABLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397289</td>
<td>dsq1_recreate_table_failed</td>
<td>RECREATE TABLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397290</td>
<td>dsq1_create_pack_failed</td>
<td>CREATE PACKAGE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397291</td>
<td>dsq1_alter_pack_failed</td>
<td>ALTER PACKAGE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397292</td>
<td>dsq1_create_alter_pack_failed</td>
<td>CREATE OR ALTER PACKAGE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397293</td>
<td>dsq1_drop_pack_failed</td>
<td>DROP PACKAGE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397294</td>
<td>dsq1_recreate_pack_failed</td>
<td>RECREATE PACKAGE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397295</td>
<td>dsq1_create_pack_body_failed</td>
<td>CREATE PACKAGE BODY @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397296</td>
<td>dsq1_drop_pack_body_failed</td>
<td>DROP PACKAGE BODY @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397297</td>
<td>dsq1_recreate_pack_body_failed</td>
<td>RECREATE PACKAGE BODY @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397298</td>
<td>dsq1_create_view_failed</td>
<td>CREATE VIEW @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397299</td>
<td>dsq1_alter_view_failed</td>
<td>ALTER VIEW @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397300</td>
<td>dsq1_create_alter_view_failed</td>
<td>CREATE OR ALTER VIEW @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397301</td>
<td>dsq1_recreate_view_failed</td>
<td>RECREATE VIEW @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397302</td>
<td>dsq1_drop_view_failed</td>
<td>DROP VIEW @1 failed</td>
</tr>
<tr>
<td>SQL- CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>-901</td>
<td>336397303</td>
<td>dsql_drop_sequence_failed</td>
<td>DROP SEQUENCE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397304</td>
<td>dsql_recreate_sequence_failed</td>
<td>RECREATE SEQUENCE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397305</td>
<td>dsql_drop_index_failed</td>
<td>DROP INDEX @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397306</td>
<td>dsql_drop_filter_failed</td>
<td>DROP FILTER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397307</td>
<td>dsql_drop_shadow_failed</td>
<td>DROP SHADOW @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397308</td>
<td>dsql_drop_role_failed</td>
<td>DROP ROLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397309</td>
<td>dsql_drop_user_failed</td>
<td>DROP USER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397310</td>
<td>dsql_create_role_failed</td>
<td>CREATE ROLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397311</td>
<td>dsql_alter_role_failed</td>
<td>ALTER ROLE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397312</td>
<td>dsql_alter_index_failed</td>
<td>ALTER INDEX @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397313</td>
<td>dsql_alter_database_failed</td>
<td>ALTER DATABASE failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397314</td>
<td>dsql_create_shadow_failed</td>
<td>CREATE SHADOW @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397315</td>
<td>dsql_create_filter_failed</td>
<td>DECLARE FILTER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397316</td>
<td>dsql_create_index_failed</td>
<td>CREATE INDEX @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397317</td>
<td>dsql_create_user_failed</td>
<td>CREATE USER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397318</td>
<td>dsql_alter_user_failed</td>
<td>ALTER USER @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397319</td>
<td>dsql_grant_failed</td>
<td>GRANT failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397320</td>
<td>dsql_revoke_failed</td>
<td>REVOKE failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397322</td>
<td>dsql_mapping_failed</td>
<td>@2 MAPPING @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397323</td>
<td>dsql_alter_sequence_failed</td>
<td>ALTER SEQUENCE @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397324</td>
<td>dsql_create_generator_failed</td>
<td>CREATE GENERATOR @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397325</td>
<td>dsql_set_generator_failed</td>
<td>SET GENERATOR @1 failed</td>
</tr>
<tr>
<td>-901</td>
<td>336397330</td>
<td>dsql_max_exception_arguments</td>
<td>Number of arguments (@1) exceeds the maximum (@2) number of EXCEPTION USING arguments</td>
</tr>
<tr>
<td>-901</td>
<td>336397331</td>
<td>dsql_string_byte_length</td>
<td>String literal with @1 bytes exceeds the maximum length of @2 bytes</td>
</tr>
<tr>
<td>-901</td>
<td>336397332</td>
<td>dsql_string_char_length</td>
<td>String literal with @1 characters exceeds the maximum length of @2 characters for the @3 character set</td>
</tr>
<tr>
<td>-901</td>
<td>336397333</td>
<td>dsql_max_nesting</td>
<td>Too many BEGIN...END nesting. Maximum level is @1</td>
</tr>
<tr>
<td>-901</td>
<td>336397334</td>
<td>dsql_recreate_user_failed</td>
<td>RECREATE USER @1 failed</td>
</tr>
<tr>
<td>-902</td>
<td>335544333</td>
<td>bug_check</td>
<td>internal Firebird consistency check (@1)</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>--------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>-902</td>
<td>335544335</td>
<td>db_corrupt</td>
<td>database file appears corrupt (@1)</td>
</tr>
<tr>
<td>-902</td>
<td>335544344</td>
<td>io_error</td>
<td>I/O error during &quot;@1&quot; operation for file &quot;@2&quot;</td>
</tr>
<tr>
<td>-902</td>
<td>335544346</td>
<td>metadata_corrupt</td>
<td>corrupt system table</td>
</tr>
<tr>
<td>-902</td>
<td>335544373</td>
<td>sys_request</td>
<td>operating system directive @1 failed</td>
</tr>
<tr>
<td>-902</td>
<td>335544384</td>
<td>badblk</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544385</td>
<td>invpoolcl</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544387</td>
<td>relbadblk</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544388</td>
<td>blktoobig</td>
<td>block size exceeds implementation restriction</td>
</tr>
<tr>
<td>-902</td>
<td>335544394</td>
<td>badodsver</td>
<td>incompatible version of on-disk structure</td>
</tr>
<tr>
<td>-902</td>
<td>335544397</td>
<td>dirtypage</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544398</td>
<td>waifortra</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544399</td>
<td>doubleloc</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544400</td>
<td>nodnotfnd</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544401</td>
<td>dupnodfnd</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544402</td>
<td>locnotmar</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544404</td>
<td>corrupt</td>
<td>database corrupted</td>
</tr>
<tr>
<td>-902</td>
<td>335544405</td>
<td>badpage</td>
<td>checksum error on database page @1</td>
</tr>
<tr>
<td>-902</td>
<td>335544406</td>
<td>badindex</td>
<td>index is broken</td>
</tr>
<tr>
<td>-902</td>
<td>335544409</td>
<td>trareqmis</td>
<td>transaction—request mismatch (synchronization error)</td>
</tr>
<tr>
<td>-902</td>
<td>335544410</td>
<td>badhndcnt</td>
<td>bad handle count</td>
</tr>
<tr>
<td>-902</td>
<td>335544411</td>
<td>wrotpbver</td>
<td>wrong version of transaction parameter block</td>
</tr>
<tr>
<td>-902</td>
<td>335544412</td>
<td>wroblrver</td>
<td>unsupported BLR version (expected @1, encountered @2)</td>
</tr>
<tr>
<td>-902</td>
<td>335544413</td>
<td>wrodpbver</td>
<td>wrong version of database parameter block</td>
</tr>
<tr>
<td>-902</td>
<td>335544415</td>
<td>badrelation</td>
<td>database corrupted</td>
</tr>
<tr>
<td>-902</td>
<td>335544416</td>
<td>nodetach</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544417</td>
<td>notremote</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544422</td>
<td>dbfile</td>
<td>internal error</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>-902</td>
<td>335544423</td>
<td>orphan</td>
<td>internal error</td>
</tr>
<tr>
<td>-902</td>
<td>335544432</td>
<td>lockmanerr</td>
<td>lock manager error</td>
</tr>
<tr>
<td>-902</td>
<td>335544436</td>
<td>sqlerr</td>
<td>SQL error code = @1</td>
</tr>
<tr>
<td>-902</td>
<td>335544448</td>
<td>bad_sec_info</td>
<td></td>
</tr>
<tr>
<td>-902</td>
<td>335544449</td>
<td>invalid_sec_info</td>
<td></td>
</tr>
<tr>
<td>-902</td>
<td>335544470</td>
<td>buf_invalid</td>
<td>cache buffer for page @1 invalid</td>
</tr>
<tr>
<td>-902</td>
<td>335544471</td>
<td>indexnotdefined</td>
<td>there is no index in table @1 with id @2</td>
</tr>
<tr>
<td>-902</td>
<td>335544472</td>
<td>login</td>
<td>Your user name and password are not defined. Ask your database administrator to set up a Firebird login.</td>
</tr>
<tr>
<td>-902</td>
<td>335544478</td>
<td>jrn_enable</td>
<td>enable journal for database before starting online dump</td>
</tr>
<tr>
<td>-902</td>
<td>335544479</td>
<td>old_failure</td>
<td>online dump failure. Retry dump</td>
</tr>
<tr>
<td>-902</td>
<td>335544480</td>
<td>old_in_progress</td>
<td>an online dump is already in progress</td>
</tr>
<tr>
<td>-902</td>
<td>335544481</td>
<td>old_no_space</td>
<td>no more disk/tape space. Cannot continue online dump</td>
</tr>
<tr>
<td>-902</td>
<td>335544482</td>
<td>no_wal_no_jrn</td>
<td>journaling allowed only if database has Write-ahead Log</td>
</tr>
<tr>
<td>-902</td>
<td>335544483</td>
<td>num_old_files</td>
<td>maximum number of online dump files that can be specified is 16</td>
</tr>
<tr>
<td>-902</td>
<td>335544484</td>
<td>wal_file_open</td>
<td>error in opening Write-ahead Log file during recovery</td>
</tr>
<tr>
<td>-902</td>
<td>335544486</td>
<td>wal_failure</td>
<td>Write-ahead log subsystem failure</td>
</tr>
<tr>
<td>-902</td>
<td>335544505</td>
<td>no_archive</td>
<td>must specify archive file when enabling long term journaling for databases with round-robin log files</td>
</tr>
<tr>
<td>-902</td>
<td>335544506</td>
<td>shutinprog</td>
<td>database @1 shutdown in progress</td>
</tr>
<tr>
<td>-902</td>
<td>335544520</td>
<td>jrn_present</td>
<td>long-term journaling already enabled</td>
</tr>
<tr>
<td>-902</td>
<td>335544528</td>
<td>shutdown</td>
<td>database @1 shutdown</td>
</tr>
<tr>
<td>-902</td>
<td>335544557</td>
<td>shutfail</td>
<td>database shutdown unsuccessful</td>
</tr>
<tr>
<td>-902</td>
<td>335544564</td>
<td>no_jrn</td>
<td>long-term journaling not enabled</td>
</tr>
<tr>
<td>-902</td>
<td>335544569</td>
<td>dsql_error</td>
<td>Dynamic SQL Error</td>
</tr>
<tr>
<td>-902</td>
<td>335544653</td>
<td>psw_attach</td>
<td>cannot attach to password database</td>
</tr>
<tr>
<td>-902</td>
<td>335544654</td>
<td>psw_start_trans</td>
<td>cannot start transaction for password database</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-902</td>
<td>335544717</td>
<td>err_stack_limit</td>
<td>stack size insufficient to execute current request</td>
</tr>
<tr>
<td>-902</td>
<td>335544721</td>
<td>network_error</td>
<td>Unable to complete network request to host &quot;@1&quot;.</td>
</tr>
<tr>
<td>-902</td>
<td>335544722</td>
<td>net_connect_err</td>
<td>Failed to establish a connection.</td>
</tr>
<tr>
<td>-902</td>
<td>335544723</td>
<td>net_connect_listen_err</td>
<td>Error while listening for an incoming connection.</td>
</tr>
<tr>
<td>-902</td>
<td>335544724</td>
<td>net_event_connect_err</td>
<td>Failed to establish a secondary connection for event processing.</td>
</tr>
<tr>
<td>-902</td>
<td>335544725</td>
<td>net_event_listen_err</td>
<td>Error while listening for an incoming event connection request.</td>
</tr>
<tr>
<td>-902</td>
<td>335544726</td>
<td>net_read_err</td>
<td>Error reading data from the connection.</td>
</tr>
<tr>
<td>-902</td>
<td>335544727</td>
<td>net_write_err</td>
<td>Error writing data to the connection.</td>
</tr>
<tr>
<td>-902</td>
<td>335544732</td>
<td>unsupported_network_drive</td>
<td>Access to databases on file servers is not supported.</td>
</tr>
<tr>
<td>-902</td>
<td>335544733</td>
<td>io_create_err</td>
<td>Error while trying to create file</td>
</tr>
<tr>
<td>-902</td>
<td>335544734</td>
<td>io_open_err</td>
<td>Error while trying to open file</td>
</tr>
<tr>
<td>-902</td>
<td>335544735</td>
<td>io_close_err</td>
<td>Error while trying to close file</td>
</tr>
<tr>
<td>-902</td>
<td>335544736</td>
<td>io_read_err</td>
<td>Error while trying to read from file</td>
</tr>
<tr>
<td>-902</td>
<td>335544737</td>
<td>io_write_err</td>
<td>Error while trying to write to file</td>
</tr>
<tr>
<td>-902</td>
<td>335544738</td>
<td>io_delete_err</td>
<td>Error while trying to delete file</td>
</tr>
<tr>
<td>-902</td>
<td>335544739</td>
<td>io_access_err</td>
<td>Error while trying to access file</td>
</tr>
<tr>
<td>-902</td>
<td>335544745</td>
<td>login_same_as_role_name</td>
<td>Your login @1 is same as one of the SQL role name. Ask your database administrator to set up a valid Firebird login.</td>
</tr>
<tr>
<td>-902</td>
<td>335544791</td>
<td>file_in_use</td>
<td>The file @1 is currently in use by another process. Try again later.</td>
</tr>
<tr>
<td>-902</td>
<td>335544795</td>
<td>unexp_spb_form</td>
<td>unexpected item in service parameter block, expected @1</td>
</tr>
<tr>
<td>-902</td>
<td>335544809</td>
<td>extern_func_dir_error</td>
<td>Function @1 is in @2, which is not in a permitted directory for external functions.</td>
</tr>
<tr>
<td>-902</td>
<td>335544819</td>
<td>io_32bit_exceeded_err</td>
<td>File exceeded maximum size of 2GB. Add another database file or use a 64 bit I/O version of Firebird.</td>
</tr>
<tr>
<td>SQL CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-902</td>
<td>335544831</td>
<td>conf_access_denied</td>
<td>Use of @1 at location @2 is not allowed by server configuration</td>
</tr>
<tr>
<td>-902</td>
<td>335544834</td>
<td>cursor_not_open</td>
<td>Cursor is not open</td>
</tr>
<tr>
<td>-902</td>
<td>335544841</td>
<td>cursor_already_open</td>
<td>Cursor is already open</td>
</tr>
<tr>
<td>-902</td>
<td>335544856</td>
<td>att_shutdown</td>
<td>connection shutdown</td>
</tr>
<tr>
<td>-902</td>
<td>335544882</td>
<td>long_login</td>
<td>Login name too long (@1 characters, maximum allowed @2)</td>
</tr>
<tr>
<td>-902</td>
<td>335544936</td>
<td>psw_db_error</td>
<td>Security database error</td>
</tr>
<tr>
<td>-902</td>
<td>335544970</td>
<td>missing_required_spb</td>
<td>Missing required item @1 in service parameter block</td>
</tr>
<tr>
<td>-902</td>
<td>335544971</td>
<td>net_server_shutdown</td>
<td>@1 server is shutdown</td>
</tr>
<tr>
<td>-902</td>
<td>335544974</td>
<td>no_threads</td>
<td>Could not start first worker thread - shutdown server</td>
</tr>
<tr>
<td>-902</td>
<td>335544975</td>
<td>net_event_connect_timeout</td>
<td>Timeout occurred while waiting for a secondary connection for event processing</td>
</tr>
<tr>
<td>-902</td>
<td>335544984</td>
<td>instance_conflict</td>
<td>Database is probably already opened by another engine instance in another Windows session</td>
</tr>
<tr>
<td>-902</td>
<td>335544987</td>
<td>no_trusted_spb</td>
<td>Use of TRUSTED switches in spb_command_line is prohibited</td>
</tr>
<tr>
<td>-902</td>
<td>335545029</td>
<td>missing_data_structures</td>
<td>Install incomplete. To complete security database initialization please CREATE USER. For details read doc/README.security_database.txt.</td>
</tr>
<tr>
<td>-902</td>
<td>335545030</td>
<td>protect_sys_tab</td>
<td>@1 operation is not allowed for system table @2</td>
</tr>
<tr>
<td>-902</td>
<td>335545032</td>
<td>wroblrver2</td>
<td>unsupported BLR version (expected between @1 and @2, encountered @3)</td>
</tr>
<tr>
<td>-902</td>
<td>335545043</td>
<td>decrypt_error</td>
<td>Missing crypt plugin, but page appears encrypted</td>
</tr>
<tr>
<td>-902</td>
<td>335545044</td>
<td>no_providers</td>
<td>No providers loaded</td>
</tr>
<tr>
<td>-902</td>
<td>335545053</td>
<td>miss_config</td>
<td>Missing configuration file: @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545054</td>
<td>conf_line</td>
<td>@1: illegal line &lt;@2&gt;</td>
</tr>
<tr>
<td>-902</td>
<td>335545055</td>
<td>conf_include</td>
<td>Invalid include operator in @1 for &lt;@2&gt;</td>
</tr>
<tr>
<td>-902</td>
<td>335545056</td>
<td>include_depth</td>
<td>Include depth too big</td>
</tr>
<tr>
<td>-902</td>
<td>335545057</td>
<td>include_miss</td>
<td>File to include not found</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>-902</td>
<td>335545060</td>
<td>sec_context</td>
<td>Missing security context for @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545061</td>
<td>multi_segment</td>
<td>Missing segment @1 in multisegment connect block parameter</td>
</tr>
<tr>
<td>-902</td>
<td>335545062</td>
<td>login_changed</td>
<td>Different logins in connect and attach packets - client library error</td>
</tr>
<tr>
<td>-902</td>
<td>335545063</td>
<td>auth_handshake_limit</td>
<td>Exceeded exchange limit during authentication handshake</td>
</tr>
<tr>
<td>-902</td>
<td>335545064</td>
<td>wirecrypt_incompatible</td>
<td>Incompatible wire encryption levels requested on client and server</td>
</tr>
<tr>
<td>-902</td>
<td>335545065</td>
<td>miss_wirecrypt</td>
<td>Client attempted to attach unencrypted but wire encryption is required</td>
</tr>
<tr>
<td>-902</td>
<td>335545066</td>
<td>wirecrypt_key</td>
<td>Client attempted to start wire encryption using unknown key @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545067</td>
<td>wirecrypt_plugin</td>
<td>Client attempted to start wire encryption using unsupported plugin @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545068</td>
<td>secdb_name</td>
<td>Error getting security database name from configuration file</td>
</tr>
<tr>
<td>-902</td>
<td>335545069</td>
<td>auth_data</td>
<td>Client authentication plugin is missing required data from server</td>
</tr>
<tr>
<td>-902</td>
<td>335545070</td>
<td>auth_datalength</td>
<td>Client authentication plugin expected @2 bytes of @3 from server, got @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545106</td>
<td>login_error</td>
<td>Error occurred during login, please check server firebird.log for details</td>
</tr>
<tr>
<td>-902</td>
<td>335545107</td>
<td>already_opened</td>
<td>Database already opened with engine instance, incompatible with current</td>
</tr>
<tr>
<td>-902</td>
<td>335545108</td>
<td>bad_crypt_key</td>
<td>Invalid crypt key @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545112</td>
<td>miss_prvlg</td>
<td>System privilege @1 is missing</td>
</tr>
<tr>
<td>-902</td>
<td>335545113</td>
<td>crypt_checksum</td>
<td>Invalid or missing checksum of encrypted database</td>
</tr>
<tr>
<td>-902</td>
<td>335545114</td>
<td>not_dba</td>
<td>You must have SYSDBA rights at this server</td>
</tr>
<tr>
<td>-902</td>
<td>335545126</td>
<td>sqlTooLong</td>
<td>SQL statement is too long. Maximum size is @1 bytes.</td>
</tr>
<tr>
<td>-902</td>
<td>335545130</td>
<td>att_shut_killed</td>
<td>Killed by database administrator.</td>
</tr>
<tr>
<td>-902</td>
<td>335545131</td>
<td>att_shut_idle</td>
<td>Idle timeout expired.</td>
</tr>
<tr>
<td>-902</td>
<td>335545132</td>
<td>att_shut_db_down</td>
<td>Database is shutdown.</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-902</td>
<td>335545133</td>
<td>att_shut_engine</td>
<td>Engine is shutdown.</td>
</tr>
<tr>
<td>-902</td>
<td>335545134</td>
<td>overriding_without_identity</td>
<td>OVERRIDING clause can be used only when an identity column is present in the INSERT's field list for table/view @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545135</td>
<td>overriding_system_invalid</td>
<td>OVERRIDING SYSTEM VALUE can be used only for identity column defined as 'GENERATED ALWAYS' in INSERT for table/view @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545136</td>
<td>overriding_user_invalid</td>
<td>OVERRIDING USER VALUE can be used only for identity column defined as 'GENERATED BY DEFAULT' in INSERT for table/view @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545137</td>
<td>overriding_system_missing</td>
<td>OVERRIDING SYSTEM VALUE should be used to override the value of an identity column defined as 'GENERATED ALWAYS' in table/view @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545166</td>
<td>icu_entrypoint</td>
<td>Missing entrypoint @1 in ICU library</td>
</tr>
<tr>
<td>-902</td>
<td>335545167</td>
<td>icu_library</td>
<td>Could not find acceptable ICU library</td>
</tr>
<tr>
<td>-902</td>
<td>335545170</td>
<td>icnv_open</td>
<td>Error opening international conversion descriptor from @1 to @2</td>
</tr>
<tr>
<td>-902</td>
<td>335545173</td>
<td>deflate_init</td>
<td>Compression stream init error @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545174</td>
<td>inflate_init</td>
<td>Decompression stream init error @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545180</td>
<td>non_plugin_protocol</td>
<td>Plugin not supported by network protocol</td>
</tr>
<tr>
<td>-902</td>
<td>335545200</td>
<td>map_event</td>
<td>Error using events in mapping shared memory: @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545201</td>
<td>map_overflow</td>
<td>Global mapping memory overflow</td>
</tr>
<tr>
<td>-902</td>
<td>335545204</td>
<td>db_crypt_key</td>
<td>Missing database encryption key for your attachment</td>
</tr>
<tr>
<td>-902</td>
<td>335545259</td>
<td>bind_err</td>
<td>Error processing isc_dpb_set_bind clumplet &quot;@1&quot;</td>
</tr>
<tr>
<td>-902</td>
<td>335545260</td>
<td>bind_statement</td>
<td>The following statement failed: @1</td>
</tr>
<tr>
<td>-902</td>
<td>335545270</td>
<td>wrong_page</td>
<td>RDB$PAGES written by non-system transaction, DB appears to be damaged</td>
</tr>
<tr>
<td>-902</td>
<td>335545271</td>
<td>repl_error</td>
<td>Replication error</td>
</tr>
<tr>
<td>-902</td>
<td>335545272</td>
<td>ses_reset_failed</td>
<td>Reset of user session failed. Connection is shut down.</td>
</tr>
<tr>
<td>-902</td>
<td>335545273</td>
<td>block_size</td>
<td>File size is less than expected</td>
</tr>
</tbody>
</table>

Appendix B: Обработка ошибок, коды и сообщения
<table>
<thead>
<tr>
<th>SQL-CODE</th>
<th>GDSCODE</th>
<th>Symbol</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>-904</td>
<td>335544324</td>
<td>bad_db_handle</td>
<td>invalid database handle (no active connection)</td>
</tr>
<tr>
<td>-904</td>
<td>335544375</td>
<td>unavailable</td>
<td>unavailable database</td>
</tr>
<tr>
<td>-904</td>
<td>335544381</td>
<td>imp_exc</td>
<td>Implementation limit exceeded</td>
</tr>
<tr>
<td>-904</td>
<td>335544386</td>
<td>nopoolids</td>
<td>too many requests</td>
</tr>
<tr>
<td>-904</td>
<td>335544389</td>
<td>bufexh</td>
<td>buffer exhausted</td>
</tr>
<tr>
<td>-904</td>
<td>335544391</td>
<td>bufinuse</td>
<td>buffer in use</td>
</tr>
<tr>
<td>-904</td>
<td>335544393</td>
<td>reqinuse</td>
<td>request in use</td>
</tr>
<tr>
<td>-904</td>
<td>335544424</td>
<td>no_lock_mng</td>
<td>no lock manager available</td>
</tr>
<tr>
<td>-904</td>
<td>335544430</td>
<td>virmemexh</td>
<td>unable to allocate memory from operating system</td>
</tr>
<tr>
<td>-904</td>
<td>335544451</td>
<td>update_conflict</td>
<td>update conflicts with concurrent update</td>
</tr>
<tr>
<td>-904</td>
<td>335544453</td>
<td>obj_in_use</td>
<td>object @1 is in use</td>
</tr>
<tr>
<td>-904</td>
<td>335544455</td>
<td>shadow_accessed</td>
<td>cannot attach active shadow file</td>
</tr>
<tr>
<td>-904</td>
<td>335544460</td>
<td>shadow_missing</td>
<td>a file in manual shadow @1 is unavailable</td>
</tr>
<tr>
<td>-904</td>
<td>335544661</td>
<td>index_root_page_full</td>
<td>cannot add index, index root page is full.</td>
</tr>
<tr>
<td>-904</td>
<td>335544676</td>
<td>sort_mem_err</td>
<td>sort error: not enough memory</td>
</tr>
<tr>
<td>-904</td>
<td>335544683</td>
<td>req_depth_exceeded</td>
<td>request depth exceeded. (Recursive definition?)</td>
</tr>
<tr>
<td>-904</td>
<td>335544758</td>
<td>sort_rec_size_err</td>
<td>sort record size of @1 bytes is too big</td>
</tr>
<tr>
<td>-904</td>
<td>335544761</td>
<td>too_many_handles</td>
<td>too many open handles to database</td>
</tr>
<tr>
<td>-904</td>
<td>335544762</td>
<td>optimizer_blk_exc</td>
<td>size of optimizer block exceeded</td>
</tr>
<tr>
<td>-904</td>
<td>335544792</td>
<td>service_att_err</td>
<td>Cannot attach to services manager</td>
</tr>
<tr>
<td>-904</td>
<td>335544799</td>
<td>svc_name_missing</td>
<td>The service name was not specified.</td>
</tr>
<tr>
<td>-904</td>
<td>335544813</td>
<td>optimizer_between_err</td>
<td>Unsupported field type specified in BETWEEN predicate.</td>
</tr>
<tr>
<td>-904</td>
<td>335544827</td>
<td>exec_sql_invalid_arg</td>
<td>Invalid argument in EXECUTE STATEMENT - cannot convert to string</td>
</tr>
<tr>
<td>-904</td>
<td>335544828</td>
<td>exec_sql_invalid_req</td>
<td>Wrong request type in EXECUTE STATEMENT '@1'</td>
</tr>
<tr>
<td>-904</td>
<td>335544829</td>
<td>exec_sql_invalid_var</td>
<td>Variable type (position @1) in EXECUTE STATEMENT '@2' INTO does not match returned column type</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>--------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>-904</td>
<td>335544830</td>
<td>exec_sql_max_call_exceeded</td>
<td>Too many recursion levels of EXECUTE STATEMENT</td>
</tr>
<tr>
<td>-904</td>
<td>335544832</td>
<td>wrong_backup_state</td>
<td>Cannot change difference file name while database is in backup mode</td>
</tr>
<tr>
<td>-904</td>
<td>335544833</td>
<td>wal_backup_err</td>
<td>Physical backup is not allowed while Write-Ahead Log is in use</td>
</tr>
<tr>
<td>-904</td>
<td>335544852</td>
<td>partner_idx_incompat_type</td>
<td>partner index segment no @1 has incompatible data type</td>
</tr>
<tr>
<td>-904</td>
<td>335544857</td>
<td>blobtooobig</td>
<td>Maximum BLOB size exceeded</td>
</tr>
<tr>
<td>-904</td>
<td>335544862</td>
<td>record_lock_not_supp</td>
<td>Stream does not support record locking</td>
</tr>
<tr>
<td>-904</td>
<td>335544863</td>
<td>partner_idx_not_found</td>
<td>Cannot create foreign key constraint @1. Partner index does not exist or is inactive.</td>
</tr>
<tr>
<td>-904</td>
<td>335544864</td>
<td>tra_num_exc</td>
<td>Transactions count exceeded. Perform backup and restore to make database operable again</td>
</tr>
<tr>
<td>-904</td>
<td>335544865</td>
<td>field_disappeared</td>
<td>Column has been unexpectedly deleted</td>
</tr>
<tr>
<td>-904</td>
<td>335544878</td>
<td>concurrent_transaction</td>
<td>concurrent transaction number is @1</td>
</tr>
<tr>
<td>-904</td>
<td>335544935</td>
<td>circular_computed</td>
<td>Cannot have circular dependencies with computed fields</td>
</tr>
<tr>
<td>-904</td>
<td>335544992</td>
<td>lock_dir_access</td>
<td>Can not access lock files directory @1</td>
</tr>
<tr>
<td>-904</td>
<td>335545020</td>
<td>request_outdated</td>
<td>Request can’t access new records in relation @1 and should be recompiled</td>
</tr>
<tr>
<td>-904</td>
<td>335545096</td>
<td>read_conflict</td>
<td>read conflicts with concurrent update</td>
</tr>
<tr>
<td>-904</td>
<td>335545110</td>
<td>max_idx_depth</td>
<td>Maximum index depth (@1 levels) is reached</td>
</tr>
<tr>
<td>-906</td>
<td>335544452</td>
<td>unlicensed</td>
<td>product @1 is not licensed</td>
</tr>
<tr>
<td>-906</td>
<td>335544744</td>
<td>max_att_exceeded</td>
<td>Maximum user count exceeded. Contact your database administrator.</td>
</tr>
<tr>
<td>-909</td>
<td>335544667</td>
<td>drdb_completed_with_errs</td>
<td>drop database completed with errors</td>
</tr>
<tr>
<td>-911</td>
<td>335544459</td>
<td>rec_in_limbo</td>
<td>record from transaction @1 is stuck in limbo</td>
</tr>
<tr>
<td>-913</td>
<td>335544336</td>
<td>deadlock</td>
<td>deadlock</td>
</tr>
<tr>
<td>-922</td>
<td>335544323</td>
<td>bad_db_format</td>
<td>file @1 is not a valid database</td>
</tr>
<tr>
<td>-923</td>
<td>335544421</td>
<td>connect_reject</td>
<td>connection rejected by remote interface</td>
</tr>
<tr>
<td>SQL-CODE</td>
<td>GDSCODE</td>
<td>Symbol</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>-923</td>
<td>335544461</td>
<td>cant_validate</td>
<td>secondary server attachments cannot validate databases</td>
</tr>
<tr>
<td>-923</td>
<td>335544462</td>
<td>cant_start_journal</td>
<td>secondary server attachments cannot start journaling</td>
</tr>
<tr>
<td>-923</td>
<td>335544464</td>
<td>cant_start_logging</td>
<td>secondary server attachments cannot start logging</td>
</tr>
<tr>
<td>-924</td>
<td>335544325</td>
<td>bad_dpb_content</td>
<td>bad parameters on attach or create database</td>
</tr>
<tr>
<td>-924</td>
<td>335544433</td>
<td>journerrrr</td>
<td>communication error with journal &quot;@1&quot;</td>
</tr>
<tr>
<td>-924</td>
<td>335544441</td>
<td>bad_detach</td>
<td>database detach completed with errors</td>
</tr>
<tr>
<td>-924</td>
<td>335544648</td>
<td>conn_lost</td>
<td>Connection lost to pipe server</td>
</tr>
<tr>
<td>-924</td>
<td>335544972</td>
<td>bad_conn_str</td>
<td>Invalid connection string</td>
</tr>
<tr>
<td>-924</td>
<td>335545085</td>
<td>baddpb_damaged_mode</td>
<td>Incompatible mode of attachment to damaged database</td>
</tr>
<tr>
<td>-924</td>
<td>335545086</td>
<td>baddpb_buffers_range</td>
<td>Attempt to set in database number of buffers which is out of acceptable range [@1:@2]</td>
</tr>
<tr>
<td>-924</td>
<td>335545087</td>
<td>baddpb_temp_buffers</td>
<td>Attempt to temporarily set number of buffers less than @1</td>
</tr>
<tr>
<td>-926</td>
<td>335544447</td>
<td>no_rollback</td>
<td>no rollback performed</td>
</tr>
<tr>
<td>-999</td>
<td>335544689</td>
<td>ib_error</td>
<td>Firebird error</td>
</tr>
</tbody>
</table>
Зарезервированные слова являются частью языка SQL Firebird. Они не могут быть использованы в качестве идентификаторов (например, имён таблиц или процедур), за исключением случаев когда они заключены в двойные кавычки (при использовании 3 диалекта). Вы должны стараться избегать их использования, если на то нет серьёзных причин.

Ключевые слова также являются частью языка. Они имеют особое значение при использовании в определённом контексте, но не являются зарезервированными для монопольного использования самим сервером Firebird. Вы можете использовать их в качестве идентификаторов без заключения в двойные кавычки.

### Зарезервированнные слова

<table>
<thead>
<tr>
<th>word</th>
<th>word</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>ADMIN</td>
<td>ALL</td>
</tr>
<tr>
<td>ALTER</td>
<td>AND</td>
<td>ANY</td>
</tr>
<tr>
<td>AS</td>
<td>AT</td>
<td>AVG</td>
</tr>
<tr>
<td>BEGIN</td>
<td>BETWEEN</td>
<td>BIGINT</td>
</tr>
<tr>
<td>BINARY</td>
<td>BIT_LENGTH</td>
<td>BLOB</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>BOTH</td>
<td>BY</td>
</tr>
<tr>
<td>CASE</td>
<td>CAST</td>
<td>CHAR</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>CHARACTER_LENGTH</td>
<td>CHAR_LENGTH</td>
</tr>
<tr>
<td>CHECK</td>
<td>CLOSE</td>
<td>COLLATE</td>
</tr>
<tr>
<td>COLUMN</td>
<td>COMMENT</td>
<td>COMMIT</td>
</tr>
<tr>
<td>CONNECT</td>
<td>CONSTRAINT</td>
<td>CORR</td>
</tr>
<tr>
<td>COUNT</td>
<td>COVAR_POP</td>
<td>COVAR_SAMP</td>
</tr>
<tr>
<td>CREATE</td>
<td>CROSS</td>
<td>CURRENT</td>
</tr>
<tr>
<td>CURRENT_CONNECTION</td>
<td>CURRENT_DATE</td>
<td>CURRENT_ROLE</td>
</tr>
<tr>
<td>CURRENT_TIME</td>
<td>CURRENT_TIMESTAMP</td>
<td>CURRENT_TRANSACTION</td>
</tr>
<tr>
<td>CURRENT_USER</td>
<td>CURSOR</td>
<td>DATE</td>
</tr>
<tr>
<td>DAY</td>
<td>DEC</td>
<td>DECFLOAT</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECLARE</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETING</td>
<td>DETERMINISTIC</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>DISTINCT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DROP</td>
<td>ELSE</td>
<td>END</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>EXECUTE</td>
<td>EXISTS</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>EXTRACT</td>
<td>FALSE</td>
</tr>
<tr>
<td>FETCH</td>
<td>FILTER</td>
<td>FLOAT</td>
</tr>
<tr>
<td>FOR</td>
<td>FOREIGN</td>
<td>FROM</td>
</tr>
<tr>
<td>FULL</td>
<td>FUNCTION</td>
<td>GDSCODE</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>GRANT</td>
<td>GROUP</td>
</tr>
<tr>
<td>HAVING</td>
<td>HOUR</td>
<td>IN</td>
</tr>
<tr>
<td>INDEX</td>
<td>INNER</td>
<td>INSENSITIVE</td>
</tr>
<tr>
<td>INSERT</td>
<td>INSERTING</td>
<td>INT</td>
</tr>
<tr>
<td>INT128</td>
<td>INTEGER</td>
<td>INTO</td>
</tr>
<tr>
<td>IS</td>
<td>JOIN</td>
<td>LATERAL</td>
</tr>
<tr>
<td>LEADING</td>
<td>LEFT</td>
<td>LIKE</td>
</tr>
<tr>
<td>LOCAL</td>
<td>LOCALTIME</td>
<td>LOCALTIMESTAMP</td>
</tr>
<tr>
<td>LONG</td>
<td>LOWER</td>
<td>MAX</td>
</tr>
<tr>
<td>MERGE</td>
<td>MIN</td>
<td>MINUTE</td>
</tr>
<tr>
<td>MONTH</td>
<td>NATIONAL</td>
<td>NATURAL</td>
</tr>
<tr>
<td>NCHAR</td>
<td>NO</td>
<td>NOT</td>
</tr>
<tr>
<td>NULL</td>
<td>NUMERIC</td>
<td>OCTET_LENGTH</td>
</tr>
<tr>
<td>OF</td>
<td>OFFSET</td>
<td>ON</td>
</tr>
<tr>
<td>ONLY</td>
<td>OPEN</td>
<td>OR</td>
</tr>
<tr>
<td>ORDER</td>
<td>OUTER</td>
<td>OVER</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>PLAN</td>
<td>POSITION</td>
</tr>
<tr>
<td>POST_EVENT</td>
<td>PRECISION</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>PUBLICATION</td>
<td>RDB$DB_KEY</td>
</tr>
<tr>
<td>RDB$ERROR</td>
<td>RDB$GET_CONTEXT</td>
<td>RDB$GET_TRANSACTIONS_CN</td>
</tr>
<tr>
<td>RDB$RECORD_VERSION</td>
<td>RDB$ROLE_IN_USE</td>
<td>RDB$SET_CONTEXT</td>
</tr>
<tr>
<td>RDB$SYSTEM_PRIVILEGE</td>
<td>REAL</td>
<td>RECORD_VERSION</td>
</tr>
<tr>
<td>RECREATE</td>
<td>RECURSIVE</td>
<td>REFERENCES</td>
</tr>
<tr>
<td>REGR_AVGX</td>
<td>REGR_AVGY</td>
<td>REGR_COUNT</td>
</tr>
<tr>
<td>REGR_INTERCEPT</td>
<td>REGR_R2</td>
<td>REGR_SLOPE</td>
</tr>
<tr>
<td>REGR_SXX</td>
<td>REGR_SXY</td>
<td>REGR_SYY</td>
</tr>
<tr>
<td>RELEASE</td>
<td>RESETTING</td>
<td>RETURN</td>
</tr>
<tr>
<td>RETURNING_VALUES</td>
<td>RETURNS</td>
<td>REVOKE</td>
</tr>
<tr>
<td>RIGHT</td>
<td>ROLLBACK</td>
<td>ROW</td>
</tr>
<tr>
<td>ROWS</td>
<td>ROW_COUNT</td>
<td>SAVEPOINT</td>
</tr>
<tr>
<td>SCROLL</td>
<td>SECOND</td>
<td>SELECT</td>
</tr>
<tr>
<td>SENSITIVE</td>
<td>SET</td>
<td>SIMILAR</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SOME</td>
<td>SQLCODE</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>START</td>
<td>STDDEV_POP</td>
</tr>
<tr>
<td>STDDEV_SAMP</td>
<td>SUM</td>
<td>TABLE</td>
</tr>
<tr>
<td>THEN</td>
<td>TIME</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>TIMEZONE_HOUR</td>
<td>TIMEZONE_MINUTE</td>
<td>TO</td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRIGGER</td>
<td>TRIM</td>
</tr>
</tbody>
</table>
Ключевые слова

Следующие термины имеют особое значение в DSQL Firebird. Некоторые из них также являются и зарезервированными словами.

<table>
<thead>
<tr>
<th>Ключевое слово</th>
<th>Значение</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>UNBOUNDED</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UPDATING</td>
<td>UPPER</td>
</tr>
<tr>
<td>USING</td>
<td>VALUE</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARYING</td>
<td>VAR_POP</td>
</tr>
<tr>
<td>VIEW</td>
<td>WHEN</td>
</tr>
<tr>
<td>WHILE</td>
<td>WINDOW</td>
</tr>
<tr>
<td>WITHOUT</td>
<td>YEAR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ключевые символы:

!<   ^<   ^=
^>   ,    :=
!=   !>   (   
)    <    <=
<>   =    >   
>=   ||   <=
~=   =>   ABS

ABSOLUTE ACCENT ACOS
ACOSH  ACTION  ACTIVE
ADD    ADMIN  AFTER
ALL    ALTER  ALWAYS
AND    ANY    AS
ASC    ASCENDING  ASCII_CHAR
ASCII_VAL  ASIN  ASINH
AT     ATAN   ATAN2
ATANH AUTO  AUTONOMOUS
AVG    BACKUP  BASE64_DECODE
BASE64_ENCODE  BEFORE  BEGIN
BETWEEN  BIGINT  BINARY
BIND    BIN_AND  BIN_NOT
BIN_OR  BIN_SHL  BIN_SHR
BIN_XOR BIT_LENGTH  BLOB
BLOCK   BODY   BOOLEAN
BOTH    BREAK  BY
CALLER  CASCADE  CASE
CAST  CEIL  CEILING
CHAR  CHARACTER  CHARACTER_LENGTH
CHAR_LENGTH  CHAR_TO_UUID  CHECK
CLEAR  CLOSE  COALESCE
COLLATE  COLLATION  COLUMN
COMMENT  COMMIT  COMMITTED
COMMON  COMPARE_DECFLOAT  COMPUTED
CONDITIONAL  CONNECT  CONNECTIONS
CONSISTENCY  CONSTRAINT  CONTAINING
CONTINUE  CORR  COS
COSH  COT  COUNT
COUNTER  COVAR_POP  COVAR_SAMP
CREATE  CROSS  CRYPT_HASH
CSTRING  CTR_BIG_ENDIAN  CTR_LENGTH
CTR_LITTLE_ENDIAN  CUME_DIST  CURRENT
CURRENT_CONNECTION  CURRENT_DATE  CURRENT_ROLE
CURRENT_TIME  CURRENT_TIMESTAMP  CURRENT_TRANSACTION
CURRENT_USER  CURSOR  DATA
DATABASE  DATE  DATEADD
DATEDIFF  DAY  DDL
DEC  DECIMAL  DECIMAL
DECLARE  DECODE  DECRYPT
DEFAULT  DEFINER  DELETE
DELETING  DENSE_RANK  DESC
DESCENDING  DESCRIPTOR  DETERMINISTIC
DIFFERENCE  DISABLE  DISCONNECT
DISTINCT  DO  DOMAIN
DOUBLE  DROP  ELSE
ENABLE  ENCRYPT  END
ENGINE  ENTRY_POINT  ESCAPE
EXCEPTION  EXCESS  EXCLUDE
EXECUTE  EXISTS  EXIT
EXP  EXTENDED  EXTERNAL
EXTRACT  FALSE  FETCH
FILE  FILTER  FIRST
FIRSTNAME  FIRST_DAY  FIRST_VALUE
FLOAT  FLOOR  FOLLOWING
FOR  FOREIGN  FREE_IT
FROM  FULL  FUNCTION
Appendix C: Зарезервированные и ключевые слова

<table>
<thead>
<tr>
<th>GDSCODE</th>
<th>GENERATED</th>
<th>GENERATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEN_ID</td>
<td>GEN_UUID</td>
<td>GLOBAL</td>
</tr>
<tr>
<td>GRANT</td>
<td>GRANTED</td>
<td>GROUP</td>
</tr>
<tr>
<td>HASH</td>
<td>HAVING</td>
<td>HEX DECODE</td>
</tr>
<tr>
<td>HEX_ENCODE</td>
<td>HOUR</td>
<td>IDENTITY</td>
</tr>
<tr>
<td>IDLE</td>
<td>IF</td>
<td>IGNORE</td>
</tr>
<tr>
<td>IIF</td>
<td>IN</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>INCREMENT</td>
<td>INDEX</td>
</tr>
<tr>
<td>INNER</td>
<td>INPUT_TYPE</td>
<td>INSENSITIVE</td>
</tr>
<tr>
<td>INSERT</td>
<td>INSERTING</td>
<td>INT</td>
</tr>
<tr>
<td>INT128</td>
<td>INTEGER</td>
<td>INTO</td>
</tr>
<tr>
<td>INVOKER</td>
<td>IS</td>
<td>ISOLATION</td>
</tr>
<tr>
<td>IV</td>
<td>JOIN</td>
<td>KEY</td>
</tr>
<tr>
<td>LAG</td>
<td>LAST</td>
<td>LASTNAME</td>
</tr>
<tr>
<td>LAST_DAY</td>
<td>LAST_VALUE</td>
<td>LATERAL</td>
</tr>
<tr>
<td>LEAD</td>
<td>LEADING</td>
<td>LEAVE</td>
</tr>
<tr>
<td>LEFT</td>
<td>LEGACY</td>
<td>LENGTH</td>
</tr>
<tr>
<td>LEVEL</td>
<td>LIFETIME</td>
<td>LIKE</td>
</tr>
<tr>
<td>LIMBO</td>
<td>LINGER</td>
<td>LIST</td>
</tr>
<tr>
<td>LN</td>
<td>LOCAL</td>
<td>LOCALTIME</td>
</tr>
<tr>
<td>LOCALTIMESTAMP</td>
<td>LOCK</td>
<td>LOG</td>
</tr>
<tr>
<td>LOG10</td>
<td>LONG</td>
<td>LOWER</td>
</tr>
<tr>
<td>LPAD</td>
<td>LPARAM</td>
<td>MAKE_DBKEY</td>
</tr>
<tr>
<td>MANUAL</td>
<td>MAPPING</td>
<td>MATCHED</td>
</tr>
<tr>
<td>MATCHING</td>
<td>MAX</td>
<td>MAXVALUE</td>
</tr>
<tr>
<td>MERGE</td>
<td>MESSAGE</td>
<td>MIDDLENAME</td>
</tr>
<tr>
<td>MILLISECOND</td>
<td>MIN</td>
<td>MINUTE</td>
</tr>
<tr>
<td>MINVALUE</td>
<td>MOD</td>
<td>MODE</td>
</tr>
<tr>
<td>MODULE_NAME</td>
<td>MONTH</td>
<td>NAME</td>
</tr>
<tr>
<td>NAMES</td>
<td>NATIONAL</td>
<td>NATIVE</td>
</tr>
<tr>
<td>NATURAL</td>
<td>NCHAR</td>
<td>NEXT</td>
</tr>
<tr>
<td>NO</td>
<td>NORMALIZE_DECIMAL</td>
<td>NOT</td>
</tr>
<tr>
<td>NTH_VALUE</td>
<td>NTILE</td>
<td>NULL</td>
</tr>
<tr>
<td>NULLIF</td>
<td>NULLS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>OCTET_LENGTH</td>
<td>OF</td>
</tr>
<tr>
<td>OFFSET</td>
<td>OLDEST</td>
<td>ON</td>
</tr>
<tr>
<td>ONLY</td>
<td>OPEN</td>
<td>OPTION</td>
</tr>
<tr>
<td>OR</td>
<td>ORDER</td>
<td>OS_NAME</td>
</tr>
<tr>
<td>OTHERS</td>
<td>OUTER</td>
<td>OUTPUT_TYPE</td>
</tr>
<tr>
<td>OVER</td>
<td>OVERFLOW</td>
<td>OVERLAY</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>OVERRIDING</td>
<td>PACKAGE</td>
<td>PAD</td>
</tr>
<tr>
<td>PAGE</td>
<td>PAGES</td>
<td>PAGE_SIZE</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>PARTITION</td>
<td>PASSWORD</td>
</tr>
<tr>
<td>PERCENT_RANK</td>
<td>PI</td>
<td>PLACING</td>
</tr>
<tr>
<td>PLAN</td>
<td>PLUGIN</td>
<td>POOL</td>
</tr>
<tr>
<td>POSITION</td>
<td>POST_EVENT</td>
<td>POWER</td>
</tr>
<tr>
<td>PRECEDING</td>
<td>PRECISION</td>
<td>PRESERVE</td>
</tr>
<tr>
<td>PRIMARY</td>
<td>PRIOR</td>
<td>PRIVILEGE</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>PROCEDURE</td>
<td>PROTECTED</td>
</tr>
<tr>
<td>PUBLICATION</td>
<td>QUANTIZE</td>
<td>RAND</td>
</tr>
<tr>
<td>RANGE</td>
<td>RANK</td>
<td>RDB$DB_KEY</td>
</tr>
<tr>
<td>RDB$ERROR</td>
<td>RDB$GET_CONTEXT</td>
<td>RDB$GET_TRANSACTION_CN</td>
</tr>
<tr>
<td>RDB$RECORD_VERSION</td>
<td>RDB$ROLE_IN_USE</td>
<td>RDB$SET_CONTEXT</td>
</tr>
<tr>
<td>RDB$SYSTEM_PRIVILEGE</td>
<td>READING</td>
<td>REAL</td>
</tr>
<tr>
<td>RECORD_VERSION</td>
<td>RECREATE</td>
<td>RECURSIVE</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>REGR_AVGX</td>
<td>REGR_AVGY</td>
</tr>
<tr>
<td>REGR_COUNT</td>
<td>REGR_INTERCEPT</td>
<td>REGR_R2</td>
</tr>
<tr>
<td>REGR_SLOPE</td>
<td>REGR_SXX</td>
<td>REGR_SXY</td>
</tr>
<tr>
<td>REGR_SYY</td>
<td>RELATIVE</td>
<td>RELEASE</td>
</tr>
<tr>
<td>REPLACE</td>
<td>REQUESTS</td>
<td>RESERV</td>
</tr>
<tr>
<td>RESERVING</td>
<td>RESET</td>
<td>RESETTING</td>
</tr>
<tr>
<td>RESTART</td>
<td>RESTRICT</td>
<td>RETAIN</td>
</tr>
<tr>
<td>RETURN</td>
<td>RETURNING</td>
<td>RETURNING_VALUES</td>
</tr>
<tr>
<td>RETURNS</td>
<td>REVERSE</td>
<td>REVOKE</td>
</tr>
<tr>
<td>RIGHT</td>
<td>ROLE</td>
<td>ROLLBACK</td>
</tr>
<tr>
<td>ROUND</td>
<td>ROW</td>
<td>ROWS</td>
</tr>
<tr>
<td>ROW_COUNT</td>
<td>ROW_NUMBER</td>
<td>RPAD</td>
</tr>
<tr>
<td>RSA_DECRYPT</td>
<td>RSA_ENCRYPT</td>
<td>RSA_PRIVATE</td>
</tr>
<tr>
<td>RSA_PUBLIC</td>
<td>RSA_SIGN_HASH</td>
<td>RSA_VERIFY_HASH</td>
</tr>
<tr>
<td>SALT_LENGTH</td>
<td>SAVEPOINT</td>
<td>SCALAR_ARRAY</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>SCROLL</td>
<td>SECOND</td>
</tr>
<tr>
<td>SECURITY</td>
<td>SEGMENT</td>
<td>SELECT</td>
</tr>
<tr>
<td>SENSITIVE</td>
<td>SEQUENCE</td>
<td>SERVERWIDE</td>
</tr>
<tr>
<td>SESSION</td>
<td>SET</td>
<td>SHADOW</td>
</tr>
<tr>
<td>SHARED</td>
<td>SIGN</td>
<td>SIGNATURE</td>
</tr>
<tr>
<td>SIMILAR</td>
<td>SIN</td>
<td>SINGULAR</td>
</tr>
<tr>
<td>SINH</td>
<td>SIZE</td>
<td>SKIP</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SNAPSHOT</td>
<td>SOME</td>
</tr>
<tr>
<td>Reserved and Keywords</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQRT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATISTICS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBSTRING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUSPEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMPORARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMEZONE_HOUR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTALORDER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TWO_PHASE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNCOMMITTED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIQUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARCHAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR_POP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WINDOW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YEARDAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLCODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STABILITY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STDDEV_POP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB_TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMEOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMEZONE_MINUTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAILING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIGGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNDO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNKNOWN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPPER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR_SAMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WEEK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHERE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WITH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPACE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLSTATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATEMENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STDDEV_SAMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TANH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSACTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUSTED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNBOUNDED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UUID_TO_CHAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARBINARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARYING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIEW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WEEKDAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WITHOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YEAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D: Системные таблицы

При первоначальном создании базы данных система управления базами данных создаёт множество системных таблиц. В системных таблицах хранятся метаданные — описания всех объектов базы данных. Системные таблицы содержат префикс RDB$ в имени.

Системные таблицы

**RDB$AUTH_MAPPING**
Сведения об отображении объектов безопасности.

**RDB$BACKUP_HISTORY**
Хранит историю копирования базы данных с помощью nbackup.

**RDB$CHARACTER_SETS**
Описание доступных наборов символов в базе данных.

**RDB$CHECK_CONSTRAINTS**
Соответствие имён триггеров именам ограничений, связанных с характеристиками NOT NULL, ограничениями CHECK и предложениями ON UPDATE и ON DELETE в ограничениях внешнего ключа.

**RDB$COLLATIONS**
Порядки сортировки для всех наборов символов.

**RDB$CONFIG**
Виртуальная таблица, отображающая актуальные параметры конфигурации, заданные в firebird.conf, databases.conf или через передаваемые через DPB.

**RDB$DATABASE**
Основные данные о базе данных.

**RDB$DB_CREATORS**
Содержит сведения о пользователях имеющих права на создание базы данных. Используется только в том случае, если текущая база данных назначена как база данных безопасности.

**RDB$DEPENDENCIES**
Сведения о зависимостях между объектами базы данных.

**RDB$EXCEPTIONS**
Пользовательские исключения базы данных.

**RDB$FIELD_DIMENSIONS**
Размерности столбцов, являющихся массивами.

**RDB$FIELDS**
Характеристики столбцов и доменов, как системных, так и созданных пользователем.
 Appendix D: Системные таблицы

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FILES</td>
<td>Сведения о вторичных файлах и файлах теневых копий.</td>
</tr>
<tr>
<td>RDB$FILTERS</td>
<td>Данные о BLOB-фильтрах.</td>
</tr>
<tr>
<td>RDB$FORMATS</td>
<td>Данные об изменениях таблиц.</td>
</tr>
<tr>
<td>RDB$FUNCTION_ARGUMENTS</td>
<td>Параметры хранимых или внешних функций.</td>
</tr>
<tr>
<td>RDB$FUNCTIONS</td>
<td>Описание хранимых или внешних функций.</td>
</tr>
<tr>
<td>RDB$GENERATORS</td>
<td>Сведения о генераторах (последовательностях).</td>
</tr>
<tr>
<td>RDB$INDEX_SEGMENTS</td>
<td>Сегменты индексов.</td>
</tr>
<tr>
<td>RDB$INDICES</td>
<td>Определение индексов базы данных (созданных пользователем или системой).</td>
</tr>
<tr>
<td>RDB$LOG_FILES</td>
<td>В настоящей версии не используется.</td>
</tr>
<tr>
<td>RDB$PACKAGES</td>
<td>Сведения о PSQL пакетах.</td>
</tr>
<tr>
<td>RDB$PAGES</td>
<td>Сведения о страницах базы данных.</td>
</tr>
<tr>
<td>RDB$PROCEDURE_PARAMETERS</td>
<td>Параметры хранимых процедур.</td>
</tr>
<tr>
<td>RDB$PROCEDURES</td>
<td>Описания хранимых процедур.</td>
</tr>
<tr>
<td>RDB$PUBLICATION_TABLES</td>
<td>Таблицы включенные в публикацию.</td>
</tr>
<tr>
<td>RDB$PUBLICATIONS</td>
<td>Публикации. Публикация — набор таблиц для репликации.</td>
</tr>
<tr>
<td>RDB$REF_CONSTRAINTS</td>
<td>Описания именованных ограничений базы данных (внешних ключей).</td>
</tr>
</tbody>
</table>
RDB$RELATION_CONSTRAINTS
Описание всех ограничений на уровне таблиц.

RDB$RELATION_FIELDS
Характеристики столбцов таблиц.

RDB$RELATIONS
Заголовки таблиц и представлений.

RDB$ROLES
Определение ролей.

RDB$SECURITY_CLASSES
Списки управления доступом.

RDB$TIME_ZONES
Список часовых поясов поддерживаемых сервером.

RDB$TRANSACTIONS
Состояние транзакций при обращении к нескольким базам данных.

RDB$TRIGGER_MESSAGES
Сообщения триггеров.

RDB$TRIGGERS
Описания триггеров.

RDB$TYPES
Описание перечислимых типов данных.

RDB$USER_PRIVILEGES
Полномочия пользователей системы.

RDB$VIEW_RELATIONS
Описывает представления. Содержит имена таблиц используемые при определении представления.

RDB$AUTH_MAPPING
Сведения о локальных отображениях объектов безопасности.

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$MAP_NAME</td>
<td>CHAR(63)</td>
<td>Имя отображения.</td>
</tr>
<tr>
<td>RDB$MAP_USING</td>
<td>CHAR(1)</td>
<td>Является ли аутентификация общесерверной (S) или обычной (P).</td>
</tr>
</tbody>
</table>
### Название столбца | Тип данных | Описание
--- | --- | ---
RDB$MAP_PLUGIN | CHAR(63) | Имя плагина аутентификации, из которого происходит отображение.
RDB$MAP_DB | CHAR(63) | Имя базы данных, в которой прошла аутентификация. Из неё происходит отображение.
RDB$MAP_FROM_TYPE | CHAR(63) | Тип объекта, который будет отображён.
RDB$MAP_FROM | CHAR(255) | Имя объекта, из которого будет выполнено отображение.
RDB$MAP_TO_TYPE | SMALLINT | Тип объекта, в который будет выполнено отображение:
| | | 0 — USER;
| | | 1 — ROLE.
RDB$MAP_TO | CHAR(63) | Наименование объекта, в который будет выполнено отображение (имя пользователя или роли).
RDB$SYSTEM_FLAG | SMALLINT | Признак: определён пользователем — значение 0; определён в системе — значение 1.
RDB$DESCRIPTION | BLOB TEXT | Произвольное текстовое описание порядка сортировки.

### RDB$BACKUP_HISTORY
Таблица хранит историю копирования базы данных при помощи утилиты nbackup.

*Table 276. Описание столбцов таблицы RDB$BACKUP_HISTORY*

| Наименование столбца | Тип данных | Описание |
--- | --- | ---
RDB$BACKUP_ID | INTEGER | Присваиваемый ядром идентификатор.
RDB$TIMESTAMP | DATE | Дата и время выполнения копирования.
RDB$BACKUP_LEVEL | INTEGER | Уровень копирования.
RDB$GUID | CHAR(38) | Уникальный идентификатор.
RDB$SCN | INTEGER | Системный номер.
RDB$FILE_NAME | VARCHAR(255) | Полный путь и имя файла копии.

### RDB$CHARACTER_SETS
Содержит наборы символов, доступные в базе данных.
<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$CHARACTER_SET_NAME</td>
<td>CHAR(63)</td>
<td>Имя набора символов.</td>
</tr>
<tr>
<td>RDB$FORM_OF_USE</td>
<td>CHAR(63)</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$NUMBER_OF_CHARACTERS</td>
<td>INTEGER</td>
<td>Количество символов в наборе. Для существующих наборов символов не используется.</td>
</tr>
<tr>
<td>RDB$DEFAULT_COLLATE_NAME</td>
<td>CHAR(63)</td>
<td>Имя порядка сортировки по умолчанию для набора символов.</td>
</tr>
<tr>
<td>RDB$CHARACTER_SET_ID</td>
<td>SMALLINT</td>
<td>Уникальный идентификатор набора символов.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Системный флаг: имеет значение 1, если набор символов был определён в системе при создании базы данных; значение 0 для набора символов, определённого пользователем.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольное текстовое описание набора символов.</td>
</tr>
<tr>
<td>RDB$FUNCTION_NAME</td>
<td>CHAR(63)</td>
<td>Имя внешней функции для наборов символов, определённых пользователем, доступ к которым осуществляется через внешнюю функцию.</td>
</tr>
<tr>
<td>RDB$BYTES_PER_CHARACTER</td>
<td>SMALLINT</td>
<td>Количество байтов для представления одного символа.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого набора символов.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя — владельца (создателя) набора символов.</td>
</tr>
</tbody>
</table>

**RDB$CHECK_CONSTRAINTS**

Описывает соответствие имен тритгеров именам ограничений, связанных с характеристиками NOT NULL, ограничениями CHECK и предложениями ON UPDATE, ON DELETE в ограничениях внешнего ключа.

*Table 278. Описание столбцов таблицы RDB$CHECK_CONSTRAINTS*
### RDB$COLLATIONS

Порядки сортировки для наборов символов.

*Table 279. Описание столбцов таблицы RDB$COLLATIONS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$COLLATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя порядка сортировки.</td>
</tr>
<tr>
<td>RDB$COLLATION_ID</td>
<td>SMALLINT</td>
<td>Идентификатор порядка сортировки. Вместе с идентификатором набора символов является уникальным идентификатором порядка сортировки.</td>
</tr>
<tr>
<td>RDB$CHARACTER_SET_ID</td>
<td>SMALLINT</td>
<td>Идентификатор набора символов. Вместе с идентификатором порядка сортировки является уникальным идентификатором.</td>
</tr>
</tbody>
</table>
## Наименование столбца | Тип данных | Описание
--- | --- | ---
RDB$COLLATION_ATTRIBUTES | SMALLINT | Атрибуты сортировки. Представляет собой битовую маску, где 1-й бит показывает учитывать ли конечные пробелы при сравнении (0 — NO PAD; 1 — PAD SPACE); 2-й бит показывает является ли сравнение чувствительным к регистру символов (0 — CASE SENSITIVE, 1 — CASE CASE SENSITIVE); 3-й бит показывает будет ли сравнение чувствительным к акцентам (0 — ACCENT SENSITIVE, 1 — ACCENT SENSITIVE). Таким образом, значение 5 означает, что сравнение не является чувствительным к конечным пробелам и к акцентированным буквам.
RDB$SYSTEM_FLAG | SMALLINT | Признак: определён пользователем — значение 0; определён в системе — значение 1.
RDB$DESCRIPTION | BLOB TEXT | Произвольное текстовое описание порядка сортировки.
RDB$FUNCTION_NAME | CHAR(63) | В настоящий момент не используется.
RDB$BASE_COLLATION_NAME | CHAR(63) | Имя базового порядка сортировки для данного порядка сортировки.
RDB$SPECIFIC_ATTRIBUTES | BLOB TEXT | Описание особых атрибутов.
RDB$SECURITY_CLASS | CHAR(63) | Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой сортировки.
RDB$OWNER_NAME | CHAR(63) | Имя пользователя — владельца (создателя) сортировки.

### RDB$CONFIG

Виртуальная таблица, отображающая актуальные параметры конфигурации, заданные в `firebird.conf`, `databases.conf` или через передаваемые через DPB.
Таблица RDB$CONFIG при необходимости заполняется из структур в памяти, а экземпляр хранится на уровне запроса SQL. По соображениям безопасности доступ разрешен только SYSDBA и владельцу базы данных. Непривилегированный пользователь видит пустое содержимое, ошибка не возникает.

*Table 280. Описание столбцов таблицы RDB$CONFIG*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$CONFIG_ID</td>
<td>INTEGER</td>
<td>Уникальный идентификатор записи. Не имеет значения.</td>
</tr>
<tr>
<td>RDB$CONFIG_NAME</td>
<td>VARCHAR(63)</td>
<td>Наименование параметра, например “DefaultDbCachePages”, “TempCacheLimit” и т. д.</td>
</tr>
<tr>
<td>RDB$CONFIG_VALUE</td>
<td>VARCHAR(255)</td>
<td>Фактическое значение настройки, может задаваться в конфигурации и при необходимости браться из ядра Firebird (в случае неверного значения).</td>
</tr>
<tr>
<td>RDB$CONFIG_DEFAULT</td>
<td>VARCHAR(255)</td>
<td>Значение настройки по умолчанию, фиксированное в коде Firebird.</td>
</tr>
<tr>
<td>RDB$CONFIG_IS_SET</td>
<td>BOOLEAN</td>
<td>TRUE, если значение установлено пользователем, FALSE в противном случае.</td>
</tr>
<tr>
<td>RDB$CONFIG_SOURCE</td>
<td>VARCHAR(255)</td>
<td>Имя конфигурационного файла, в котором был задан параметр, относительно корневой папки firebird, например: “firebird.conf”, “databases.conf” или специальное значение “DPB”, если параметр был установлен в DPB, если значение параметра не было задано, то это поле содержит NULL.</td>
</tr>
</tbody>
</table>

**RDB$DATABASE**

Основные данные о базе данных. Содержит только одну запись.

*Table 281. Описание столбцов таблицы RDB$DATABASE*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Текст примечания для базы данных.</td>
</tr>
<tr>
<td>RDB$RELATION_ID</td>
<td>SMALLINT</td>
<td>Количество таблиц и представлений в базе данных.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Класс безопасности, определённый в RDB$SECURITY_CLASSES, для обращения к общим для базы данных ограничениям доступа.</td>
</tr>
<tr>
<td>RDB$CHARACTER_SET_NAME</td>
<td>CHAR(63)</td>
<td>Имя набора символов по умолчанию для базы данных, установленного в предложении DEFAULT CHARACTER SET при создании базы данных. NULL — набор символов NONE.</td>
</tr>
<tr>
<td>RDB$LINGER</td>
<td>INTEGER</td>
<td>Количество секунд &quot;задержки&quot; (установленной оператором alter database set linger) до закрытия последнего соединения базы данных (в SuperServer). Если задержка не установлена, то содержит NULL.</td>
</tr>
<tr>
<td>RDB$SQL_SECURITY</td>
<td>BOOLEAN</td>
<td>Режим SQL SECURITY по умолчанию (DEFINER или INVOKER) для вновь создаваемым объектам: NULL — режим по умолчанию (INVOKER); FALSE — INVOKER. Вновь создаваемые объекты выполняются с правами вызывающего пользователя; TRUE — DEFINER. Вновь создаваемые объекты выполняются с правами определяющего пользователя.</td>
</tr>
</tbody>
</table>

**RDB$DB_CREATORS**

Содержит сведения о пользователях имеющих права на создание базы данных. Используется только в том случае, если текущая база данных назначена как база данных безопасности.

**Table 282. Описание столбцов таблицы RDB$DB_CREATORS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$USER</td>
<td>CHAR(63)</td>
<td>Имя пользователя или роли, которому даны полномочия на создание базы данных.</td>
</tr>
<tr>
<td>RDB$USER_TYPE</td>
<td>SMALLINT</td>
<td>Тип пользователя: 8 — пользователь; 13 — роль.</td>
</tr>
</tbody>
</table>
### RDB$DEPENDENCIES

Сведения о зависимостях между объектами базы данных.

*Table 283. Описание столбцов таблицы RDB$DEPENDENCIES*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$DEPENDENT_NAME</td>
<td>CHAR(63)</td>
<td>Имя представления, процедуры, триггера, ограничения CHECK или вычисляемого столбца, для которого описывается зависимость.</td>
</tr>
<tr>
<td>RDB$DEPENDED_ON_NAME</td>
<td>CHAR(63)</td>
<td>Объект, зависящий от описываемого объекта — таблица, на которую ссылается представление, процедура, триггер, ограничение CHECK или вычисляемый столбец.</td>
</tr>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя столбца в зависимой таблице, на который ссылается представление, процедура, триггер, ограничение CHECK или вычисляемый столбец.</td>
</tr>
<tr>
<td>RDB$DEPENDENT_TYPE</td>
<td>SMALLINT</td>
<td>Идентифицирует тип объекта, для которого описывается зависимость:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — таблица;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — представление;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — триггер;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — вычисляемый столбец;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 — ограничение CHECK;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 — процедура;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 — выражение для индекса;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 — столбец;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 — хранимая функция;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 — заголовок пакета;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 — тело пакета.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>RDB$DEPENDED_ON_TYPE</td>
<td>SMALLINT</td>
<td>Идентифицирует тип зависимого объекта: 0 — таблица (или её столбец); 1 — представление; 2 — триггер; 3 — вычисляемый столбец; 4 — ограничение CHECK; 5 — процедура; 6 — выражение для индекса; 7 — исключение; 8 — пользователь; 9 — столбец; 10 — индекс; 14 — генератор (последовательность); 15 — UDF или хранимая функция; 17 — сортировка; 18 — заголовок пакета; 19 — тело пакета.</td>
</tr>
</tbody>
</table>

| RDB$PACKAGE_NAME | CHAR(63) | Пакет процедуры или функции, для которой описывается зависимость. |

**RDB$Exceptions**

Пользовательские исключения базы данных.

*Table 284. Описание столбцов таблицы RDB$Exceptions*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$EXCEPTION_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользовательского исключения.</td>
</tr>
<tr>
<td>RDB$EXCEPTION_NUMBER</td>
<td>INTEGER</td>
<td>Назначенный системой уникальный номер исключения.</td>
</tr>
<tr>
<td>RDB$MESSAGE</td>
<td>CHAR(1023)</td>
<td>Текст сообщения в исключении.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольное текстовое описание исключения.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Признак: определено пользователем = 0; определено системой = 1 или выше.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого исключения.</td>
</tr>
</tbody>
</table>
### RDB$FIELD_DIMENSIONS

Размерности столбцов, являющихся массивами.

**Table 285. Описание столбцов таблицы RDB$FIELD_DIMENSIONS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя столбца, являющегося массивом. Должно содержаться в поле RDB$FIELD_NAME таблицы RDB$FIELDS.</td>
</tr>
<tr>
<td>RDB$DIMENSION</td>
<td>SMALLINT</td>
<td>Определяет одну размерность столбца массива. Нумерация размерностей начинается с 0.</td>
</tr>
<tr>
<td>RDB$LOWER_BOUND</td>
<td>INTEGER</td>
<td>Нижняя граница этой размерности.</td>
</tr>
<tr>
<td>RDB$UPPER_BOUND</td>
<td>INTEGER</td>
<td>Верхняя граница описываемой размерности.</td>
</tr>
</tbody>
</table>

### RDB$FIELDS

Характеристики столбцов и доменов, как системных, так и созданных пользователем. В этой таблице хранятся подробности атрибутов всех столбцов.

Столбец RDB$FIELDS.RDB$FIELD_NAME ссылается на RDB$RELATION_FIELDS.RDB$FIELD_SOURCE, RDB$RELATION_FIELDS.RDB$FIELD_NAME, но не на RDB$QUERY_NAME.

**Table 286. Описание столбцов таблицы RDB$FIELDS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Уникальное имя домена, созданного пользователем, или домена, автоматически построенного ядром Firebird для столбца таблицы. Во втором случае имя будет начинаться с символов RDB$.</td>
</tr>
<tr>
<td>RDB$QUERY_NAME</td>
<td>CHAR(63)</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$VALIDATION_BLR</td>
<td>BLOB BLR</td>
<td>Двоичное представление (BLR) выражения SQL, задающее проверку значения CHECK у домена.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>RDB$VALIDATION_SOURCE</td>
<td>BLOB TEXT</td>
<td>Оригинальный исходный текст на языке SQL, задающий проверку значения CHECK.</td>
</tr>
<tr>
<td>RDB$COMPUTED_BLR</td>
<td>BLOB BLR</td>
<td>Двоичное представление (BLR) выражения SQL, которое используется сервером базы данных для вычисления при обращении к столбцу COMPUTED BY.</td>
</tr>
<tr>
<td>RDB$COMPUTED_SOURCE</td>
<td>BLOB TEXT</td>
<td>Оригинальный исходный текст выражения, которое определяет столбец COMPUTED BY.</td>
</tr>
<tr>
<td>RDB$DEFAULT_VALUE</td>
<td>BLOB BLR</td>
<td>Значение по умолчанию в двоичном виде BLR.</td>
</tr>
<tr>
<td>RDB$DEFAULT_SOURCE</td>
<td>BLOB TEXT</td>
<td>Значение по умолчанию в исходном виде на языке SQL.</td>
</tr>
<tr>
<td>RDB$FIELD_LENGTH</td>
<td>SMALLINT</td>
<td>Размер столбца в байтах. FLOAT, DATE, TIME, INTEGER занимают 4 байта. DOUBLE PRECISION, BIGINT, TIMESTAMP и идентификатор BLOB — 8 байтов. Для типов данных CHAR и VARCHAR столбец задаёт максимальное количество байтов, указанное при объявлении строкового домена (столбца).</td>
</tr>
<tr>
<td>RDB$FIELD_SCALE</td>
<td>SMALLINT</td>
<td>Отрицательное число задаёт масштаб для столбцов DECIMAL и NUMERIC — количество дробных знаков после десятичной точки.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>RDB$FIELD_TYPE</td>
<td>SMALLINT</td>
<td>Код типа данных для столбца:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 – SMALLINT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 – INTEGER;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 – FLOAT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 – DATE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 – TIME WITHOUT TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 – CHAR или BINARY;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 – BIGINT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23 – BOOLEAN;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 – DECFLOAT(16);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 – DECFLOAT(34);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26 – INT128;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27 – DOUBLE PRECISION;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28 – TIME WITH TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29 – TIMESTAMP WITH TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35 – TIMESTAMP WITHOUT TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37 – VARCHAR или VARBINARY;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>261 – BLOB.</td>
</tr>
</tbody>
</table>

Коды для DECIMAL и NUMERIC имеют тот же размер, что и целые типы, используемые для их хранения.

Для типов BINARY, VARBINARY поле RDB$FIELD_SUB_TYPE = 0, для CHAR и VARCHAR поле RDB$FIELD_SUB_TYPE = 1.
<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FIELD_SUB_TYPE</td>
<td>SMALLINT</td>
<td>Для типа данных BLOB задаёт подтип:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 – не определён;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 – текст;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 – BLR;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 – список управления доступом;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 4 – резервируется для дальнейшего использования;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 5 – кодированное описание метаданных таблицы;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 6 – описание транзакции к нескольким базам данных, которая не завершилась нормально.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Для типа данных CHAR задаёт:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 – неопределённые данные;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 – фиксированные двоичные данные.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Для целочисленных типов данных (SMALLINT, INTEGER, BIGINT, INT128) и чисел с фиксированной точкой (NUMERIC, DECIMAL) задаёт конкретный тип данных:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 или NULL – тип данных соответствует значению в поле RDB$FIELD_TYPE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 – NUMERIC;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 – DECIMAL.</td>
</tr>
<tr>
<td>RDB$MISSING_VALUE</td>
<td>BLOB</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$MISSING_SOURCE</td>
<td>BLOB TEXT</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст комментария для домена (столбца таблицы).</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Признак: значение 1 – домен, автоматически созданный системой, значение 0 – домен определён пользователем.</td>
</tr>
<tr>
<td>RDB$QUERY_HEADER</td>
<td>BLOB TEXT</td>
<td>Не используется.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$SEGMENT_LENGTH</td>
<td>SMALLINT</td>
<td>Для столбцов BLOB задаёт длину буфера BLOB в байтах. Для остальных типов данных содержит NULL.</td>
</tr>
<tr>
<td>RDB$EDIT_STRING</td>
<td>VARCHAR(127)</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$EXTERNAL_LENGTH</td>
<td>SMALLINT</td>
<td>Длина столбца в байтах, если он входит в состав внешней таблицы. Всегда NULL для обычных таблиц.</td>
</tr>
<tr>
<td>RDB$EXTERNAL_SCALE</td>
<td>SMALLINT</td>
<td>Показатель степени для столбца целого типа данных во внешней таблице; задаётся степенью 10, на которую умножается целое.</td>
</tr>
<tr>
<td>RDB$EXTERNAL_TYPE</td>
<td>SMALLINT</td>
<td>Тип данных поля, как он представляется во внешней таблице.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 – SMALLINT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 – INTEGER;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 – FLOAT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 – DATE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 – TIME WITHOUT TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 – CHAR;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 – BIGINT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23 – BOOLEAN;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 – DECFLOAT(16);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 – DECFLOAT(34);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26 – INT128;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27 – DOUBLE PRECISION;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28 – TIME WITH TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29 – TIMESTAMP WITH TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35 – TIMESTAMP WITHOUT TIME ZONE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37 – VARCHAR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Коды для DECIMAL и NUMERIC имеют тот же размер, что и целые типы, используемые для их хранения.</td>
</tr>
<tr>
<td>RDB$DIMENSIONS</td>
<td>SMALLINT</td>
<td>Задаёт количество размерностей массива, если столбец был определён как массив. Для столбцов, не являющихся массивами, всегда NULL.</td>
</tr>
<tr>
<td>RDB$NULL_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, может ли столбец принимать пустое значение (в поле будет значение NULL) или не может (в поле будет содержаться значение 1).</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$CHARACTER_LENGTH</td>
<td>SMALLINT</td>
<td>Длина столбцов CHAR или VARCHAR в символах (не в байтах).</td>
</tr>
<tr>
<td>RDB$COLLATION_ID</td>
<td>SMALLINT</td>
<td>Идентификатор порядка сортировки для символьного столбца или домена. Если не задан, значением поля будет 0.</td>
</tr>
<tr>
<td>RDB$CHARACTER_SET_ID</td>
<td>SMALLINT</td>
<td>Идентификатора набора символов для символьного столбца, столбца BLOB или домена.</td>
</tr>
<tr>
<td>RDB$FIELD_PRECISION</td>
<td>SMALLINT</td>
<td>Указывает общее количество цифр для числового типа данных с фиксированной точкой (DECIMAL и NUMERIC). Для целочисленных типов данных значением является 0, для всех остальных типов данных – NULL.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого домена.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя – владельца (создателя) домена.</td>
</tr>
</tbody>
</table>

**RDB$FILES**

Сведения о вторичных файлах и файлах оперативных копий.

*Table 287. Описание столбцов таблицы RDB$FILES*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FILE_NAME</td>
<td>VARCHAR(255)</td>
<td>Полный путь к файлу и имя вторичного файла базы данных в многофайловой базе данных или файла оперативной копии.</td>
</tr>
<tr>
<td>RDB$FILE_SEQUENCE</td>
<td>SMALLINT</td>
<td>Порядковый номер вторичного файла в последовательности или номер файла копии в наборе оперативных копий.</td>
</tr>
<tr>
<td>RDB$FILE_START</td>
<td>INTEGER</td>
<td>Начальный номер страницы вторичного файла или файла оперативной копии.</td>
</tr>
</tbody>
</table>
### RDB$FILTERS

Содержит данные о BLOB-фильтрах.

**Table 288. Описание столбцов таблицы RDB$FILTERS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FUNCTION_NAME</td>
<td>CHAR(63)</td>
<td>Уникальное имя фильтра BLOB.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Написанная пользователем документация о фильтре BLOB и используемых двух подтипах.</td>
</tr>
<tr>
<td>RDB$MODULE_NAME</td>
<td>VARCHAR(255)</td>
<td>Имя динамической библиотеки / совместно используемого объекта, где расположен код фильтра BLOB.</td>
</tr>
<tr>
<td>RDB$ENTRYPOINT</td>
<td>CHAR(255)</td>
<td>Точка входа в библиотеке фильтров для этого фильтра BLOB.</td>
</tr>
<tr>
<td>RDB$INPUT_SUB_TYPE</td>
<td>SMALLINT</td>
<td>Подтип BLOB для преобразуемых данных.</td>
</tr>
<tr>
<td>RDB$OUTPUT_SUB_TYPE</td>
<td>SMALLINT</td>
<td>Подтип BLOB, в который преобразуются входные данные.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Признак: внешнее определённый фильтр (т.е. определённый пользователем = значение 0, внутренне определённый = значение 1 или более)</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого BLOB фильтра.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя — владелец (создателя) BLOB фильтра.</td>
</tr>
</tbody>
</table>
**RDB$FORMATS**

Таблица RDB$FORMATS хранит данные об изменениях метаданных таблиц. Каждый раз, когда метаданные таблицы изменяются, таблица получает новый номер формата. Когда номер формата любой таблицы достигает 255 (или 32000 для представлений), вся база данных становится недоступной для работы с ней. В этом случае необходимо выполнить резервное копирование с помощью утилиты gbak, после чего восстановить эту копию и продолжить работу с заново созданной базой данных.

*Table 289. Описание столбцов таблицы RDB$FORMATS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$RELATION_ID</td>
<td>SMALLINT</td>
<td>Идентификатор таблицы или представления.</td>
</tr>
<tr>
<td>RDB$FORMAT</td>
<td>SMALLINT</td>
<td>Идентификатор формата таблицы. Форматов может быть 255 для таблиц и 32000 для представлений.</td>
</tr>
<tr>
<td>RDB$DESCRIPTOR</td>
<td>BLOB FORMAT</td>
<td>Отображение в виде BLOB столбцов и характеристик данных на момент, когда была создана запись формата.</td>
</tr>
</tbody>
</table>

**RDB$FUNCTION_ARGUMENTS**

Параметры хранимых или внешних функций.

*Table 290. Описание столбцов таблицы RDB$FUNCTION_ARGUMENTS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FUNCTION_NAME</td>
<td>CHAR(63)</td>
<td>Имя функции.</td>
</tr>
<tr>
<td>RDB$ARGUMENT_POSITION</td>
<td>SMALLINT</td>
<td>Позиция аргумента в списке аргументов.</td>
</tr>
<tr>
<td>RDB$MECHANISM</td>
<td>SMALLINT</td>
<td>Механизм передачи параметра для Legacy функций:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — по значению;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — по ссылке;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — через дескриптор;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — через дескриптор BLOB.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| RDB$FIELD_TYPE       | SMALLINT   | Код типа данных аргумента:  
|                      |            | 7 — SMALLINT;  
|                      |            | 8 — INTEGER;  
|                      |            | 12 — DATE;  
|                      |            | 13 — TIME WITHOUT TIME ZONE;  
|                      |            | 14 — CHAR;  
|                      |            | 16 — BIGINT;  
|                      |            | 23 — BOOLEAN;  
|                      |            | 24 — DECFLOAT(16);  
|                      |            | 25 — DECFLOAT(34);  
|                      |            | 26 — INT128;  
|                      |            | 27 — DOUBLE PRECISION;  
|                      |            | 28 — TIME WITH TIME ZONE;  
|                      |            | 29 — TIMESTAMP WITH TIME ZONE;  
|                      |            | 35 — TIMESTAMP WITHOUT TIME ZONE;  
|                      |            | 37 — VARCHAR;  
|                      |            | 40 — CSTRING;  
|                      |            | 45 — blob id;  
|                      |            | 261 — BLOB. |
| RDB$FIELD_SCALE      | SMALLINT   | Масштаб для целого числа или аргумента с фиксированной точкой. Это показатель числа 10. |
| RDB$FIELD_LENGTH     | SMALLINT   | Длина аргумента в байтах:  
|                      |            | 1 — для BOOLEAN;  
|                      |            | 2 — для SMALLINT;  
|                      |            | 4 — для INTEGER;  
|                      |            | 4 — для DATE;  
|                      |            | 4 — для TIME WITHOUT TIME ZONE;  
|                      |            | 8 — для TIME WITH TIME ZONE;  
|                      |            | 8 — для BIGINT;  
|                      |            | 8 — для DOUBLE PRECISION;  
|                      |            | 8 — для TIMESTAMP WITHOUT TIME ZONE;  
|                      |            | 8 — для DECFLOAT(16);  
|                      |            | 8 — для blob id;  
|                      |            | 12 — для TIMESTAMP WITH TIME ZONE;  
|                      |            | 16 — для DECFLOAT(34);  
<p>|                      |            | 16 — для INT128. |
| RDB$FIELD_SUB_TYPE   | SMALLINT   | Для аргумента типа данных BLOB задаёт подтип BLOB. |
| RDB$CHARACTER_SET_ID | SMALLINT   | Идентификатор набора символов для символьного аргумента. |</p>
<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FIELD_PRECISION</td>
<td>SMALLINT</td>
<td>Количество цифр точности, допустимой для типа данных аргумента.</td>
</tr>
<tr>
<td>RDB$CHARACTER_LENGTH</td>
<td>SMALLINT</td>
<td>Длина аргумента CHAR или VARCHAR в символах (не в байтах).</td>
</tr>
<tr>
<td>RDB$PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Имя пакета функции (если функция упакованная), в которой используется параметр.</td>
</tr>
<tr>
<td>RDB$ARGUMENT_NAME</td>
<td>CHAR(63)</td>
<td>Имя параметра.</td>
</tr>
<tr>
<td>RDB$FIELD_SOURCE</td>
<td>CHAR(63)</td>
<td>Имя домена, созданного пользователем (при использовании ссылки на домен вместо типа), или домена, автоматически построенного системой для параметра функции. Во втором случае имя будет начинаться с символов RDB$.</td>
</tr>
<tr>
<td>RDB$DEFAULT_VALUE</td>
<td>BLOB BLR</td>
<td>Значение по умолчанию на языке BLR.</td>
</tr>
<tr>
<td>RDB$DEFAULT_SOURCE</td>
<td>BLOB TEXT</td>
<td>Значение по умолчанию в исходном виде на языке SQL.</td>
</tr>
<tr>
<td>RDB$COLLATION_ID</td>
<td>SMALLINT</td>
<td>Идентификатор используемого порядка сортировки для символьного параметра.</td>
</tr>
<tr>
<td>RDB$NULL_FLAG</td>
<td>SMALLINT</td>
<td>Признак допустимости пустого значения NULL.</td>
</tr>
<tr>
<td>RDB$ARGUMENT_MECHANISM</td>
<td>SMALLINT</td>
<td>Механизм передачи параметра для не Legacy функций:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — по значению;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — по ссылке;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — через дескриптор;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — через дескриптор BLOB.</td>
</tr>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя столбца, на которое ссылается параметр с помощью предложения TYPE OF COLUMN.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы, на которую ссылается параметр с помощью предложения TYPE OF COLUMN.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, является ли параметр определённым системой (значение 1 и выше) или пользователем (значение 0).</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Текст произвольного примечания к параметру.</td>
</tr>
</tbody>
</table>

**RDB$FUNCTIONS**

Описание хранимых или внешних функций.

*Table 291. Описание столбцов таблицы RDB$FUNCTIONS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FUNCTION_NAME</td>
<td>CHAR(63)</td>
<td>Имя функции.</td>
</tr>
<tr>
<td>RDB$FUNCTION_TYPE</td>
<td>SMALLINT</td>
<td>В настоящий момент не используется.</td>
</tr>
<tr>
<td>RDB$QUERY_NAME</td>
<td>CHAR(63)</td>
<td>В настоящий момент не используется.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст комментария к функции.</td>
</tr>
<tr>
<td>RDB$MODULE_NAME</td>
<td>VARCHAR(255)</td>
<td>Имя внешнего модуля (динамической библиотеки), где расположен код функции.</td>
</tr>
<tr>
<td>RDB$ENTRYPOINT</td>
<td>CHAR(255)</td>
<td>Имя точки входа в библиотеке, где находится эта функция.</td>
</tr>
<tr>
<td>RDB$RETURN_ARGUMENT</td>
<td>SMALLINT</td>
<td>Номер позиции возвращаемого аргумента в списке параметров, соответствующем входным аргументам.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Признак определения функции: 0 — определённая системой, 1 — определённая пользователем.</td>
</tr>
<tr>
<td>RDB$ENGINE_NAME</td>
<td>CHAR(63)</td>
<td>Имя движка для использования внешних функций. Обычно UDR.</td>
</tr>
<tr>
<td>RDB$PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Имя пакета, если функция является упакованной.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$PRIVATE_FLAG</td>
<td>SMALLINT</td>
<td>Для неупакованных хранимых функций всегда NULL, для упакованных 0 — если функция описана в заголовке пакета и 1 — если функция описана или реализована только в теле пакета (не описана в заголовке).</td>
</tr>
<tr>
<td>RDB$FUNCTION_SOURCE</td>
<td>BLOB TEXT</td>
<td>Исходный код функции на языке SQL.</td>
</tr>
<tr>
<td>RDB$FUNCTION_ID</td>
<td>SMALLINT</td>
<td>Уникальный идентификатор функции.</td>
</tr>
<tr>
<td>RDB$FUNCTION_BLR</td>
<td>BLOB BLR</td>
<td>Двоичное представление (BLR) кода функции.</td>
</tr>
<tr>
<td>RDB$VALID_BLR</td>
<td>SMALLINT</td>
<td>Указывает, остаётся ли текст хранимой функции корректным после последнего изменения функции при помощи оператора ALTER FUNCTION.</td>
</tr>
<tr>
<td>RDB$DEBUG_INFO</td>
<td>BLOB</td>
<td>Содержит отладочную информацию о переменных, используемых в хранимой функции.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может указывать на класс безопасности, определённый в системной таблице RDB$SECURITY_CLASSES, для применения ограничений управления доступом.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя — владелец (создатель) функции.</td>
</tr>
<tr>
<td>RDB$LEGACY_FLAG</td>
<td>SMALLINT</td>
<td>Признак legacy стиля функции.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — если функция описана в legacy стиле (DECLARE EXTERNAL FUNCTION), в противном случае 0 (CREATE FUNCTION).</td>
</tr>
<tr>
<td>RDB$DETERMINISTIC_FLAG</td>
<td>SMALLINT</td>
<td>Флаг детерминистической функции.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — если функция детерминистическая (DETERMINISTIC), в противном случае — 0.</td>
</tr>
</tbody>
</table>
**RDB$SQL_SECURITY**

С какими правами выполняется функция:

TRUE — с правами определяющего пользователя (SQL SECURITY DEFINER);
FALSE — с правами вызывающего пользователя (SQL SECURITY INVOKER);
NULL — привилегии выполнения наследуются от пакета.

**RDB$GENERATORS**

Сведения о генераторах (последовательностях).

*Table 292. Описание столбцов таблицы RDB$GENERATORS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$GENERATOR_NAME</td>
<td>CHAR(63)</td>
<td>Уникальное имя генератора.</td>
</tr>
<tr>
<td>RDB$GENERATOR_ID</td>
<td>SMALLINT</td>
<td>Назначаемый системой уникальный идентификатор для генератора.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Признак:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — генератор определён пользователем,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 или выше — определён системой.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 — внутренний генератор для identity столбца.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст примечания к генератору.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может указывать на класс безопасности, определённый в системной таблице RDB$SECURITY_CLASSES, для применения ограничений управления доступом.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя — владельца (создателя) генератора.</td>
</tr>
<tr>
<td>RDB$INITIAL_VALUE</td>
<td>BIGINT</td>
<td>Хранит начальное значение генератора или значение генератора, установленное при предыдущем рестарте (WITH RESTART).</td>
</tr>
<tr>
<td>RDB$GENERATOR_INCREMENT</td>
<td>INTEGER</td>
<td>Шаг приращения генератора при использовании оператора NEXT VALUE FOR.</td>
</tr>
</tbody>
</table>
**RDB$INDEX_SEGMENTS**

Сегменты и позиции индексов. Таблица описывает все столбцы таблицы, входящие в состав конкретного индекса. Для каждого столбца индекса создаётся отдельная строка в данной таблице.

*Table 293. Описание столбцов таблицы RDB$INDEX_SEGMENTS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$INDEX_NAME</td>
<td>CHAR(63)</td>
<td>Имя индекса, к которому относится данный сегмент. Должно соответствовать главной записи в системной таблице RDB$INDICES.</td>
</tr>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя одного из столбцов, входящего в состав индекса. Должно соответствовать значению в столбце RDB$FIELD_NAME в таблице RDB$RELATION_FIELDS.</td>
</tr>
<tr>
<td>RDB$FIELD_POSITION</td>
<td>SMALLINT</td>
<td>Позиция столбца в индексе. Нумерация начинается с нуля.</td>
</tr>
<tr>
<td>RDB$STATISTICS</td>
<td>DOUBLE PRECISION</td>
<td>Последнее известное (рассчитанное) значение селективности индекса по данному столбцу.</td>
</tr>
</tbody>
</table>

**RDB$INDICES**

Определение индексов базы данных (созданных пользователем или системой). Описывает каждый индекс, созданный пользователем или системой. Для каждого столбца таблицы, входящего в состав индекса, присутствует строка системной таблицы RDB$INDEX_SEGMENTS, где описываются характеристики столбца индекса.

*Table 294. Описание столбцов таблицы RDB$INDICES*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$INDEX_NAME</td>
<td>CHAR(63)</td>
<td>Уникальное имя индекса, заданное пользователем или автоматически сгенерированное системой.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы, к которой применяется индекс. Соответствует RDB$RELATION_NAME в строке таблицы RDB$RELATIONS.</td>
</tr>
<tr>
<td>RDB$INDEX_ID</td>
<td>SMALLINT</td>
<td>Внутренний (системный) идентификатор индекса.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$UNIQUE_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, является ли индекс уникальным: 0 — не уникальный; 1 — уникальный.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст комментария к индексу.</td>
</tr>
<tr>
<td>RDB$SEGMENT_COUNT</td>
<td>SMALLINT</td>
<td>Количество сегментов (столбцов) в индексе.</td>
</tr>
<tr>
<td>RDB$INDEX_INACTIVE</td>
<td>SMALLINT</td>
<td>Указывает, является ли в настоящий момент индекс активным: 0 — активный; 1 — неактивный; 3 — состояние только для Foreign Key. Это состояние существует только во время восстановления данных. Оно необходимо чтобы различать частично “неактивное” состояние некоторых индексов от неактивного состояния всех индексов (gbak -i).</td>
</tr>
<tr>
<td>RDB$INDEX_TYPE</td>
<td>SMALLINT</td>
<td>Направление индекса: 0 — ascending; 1 — descending.</td>
</tr>
<tr>
<td>RDB$FOREIGN_KEY</td>
<td>CHAR(63)</td>
<td>Имя ассоциированного ограничения внешнего ключа, если существует.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, является ли индекс определённым системой (значение 1 или выше) или пользователем (значение 0).</td>
</tr>
<tr>
<td>RDB$EXPRESSION_BLR</td>
<td>BLOB BLR</td>
<td>Выражение, записанное на языке двоичного представления (BLR). Будет использовано для вычисления во время выполнения, когда будут реализованы индексы выражений.</td>
</tr>
<tr>
<td>RDB$EXPRESSION_SOURCE</td>
<td>BLOB TEXT</td>
<td>Исходный текст выражения. Будет использовано, когда будут реализованы индексы выражений.</td>
</tr>
</tbody>
</table>
### RDB$LOG_FILES

В настоящей версии не используется.

**Table 295. Описание столбцов таблицы RDB$LOG_FILES**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FILE_NAME</td>
<td>VARCHAR(255)</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$FILE_SEQUENCE</td>
<td>SMALLINT</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$FILE_LENGTH</td>
<td>INTEGER</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$FILE_PARTITIONS</td>
<td>SMALLINT</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$FILE_P_OFFSET</td>
<td>INTEGER</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$FILE_FLAGS</td>
<td>SMALLINT</td>
<td>Не используется.</td>
</tr>
</tbody>
</table>

### RDB$PACKAGES

Сведения о PSQL пакетах.

**Table 296. Описание столбцов таблицы RDB$PACKAGES**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Уникальное имя пакета.</td>
</tr>
<tr>
<td>RDB$PACKAGE_HEADER_SOURCE</td>
<td>BLOB TEXT</td>
<td>Исходный код заголовка пакета на языке SQL.</td>
</tr>
<tr>
<td>RDB$PACKAGE_BODY_SOURCE</td>
<td>BLOB TEXT</td>
<td>Исходный код тела пакета на языке SQL.</td>
</tr>
<tr>
<td>RDB$VALID_BODY_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, остаётся ли текст тела пакета корректным после последнего изменения заголовка пакета или его пересоздания.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может указывать на класс безопасности, определённый в системной таблице RDB$SECURITY_CLASSES, для применения ограничений управления доступом.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя – владельца (создателя) пакета.</td>
</tr>
</tbody>
</table>
### RDB$SYSTEM_FLAG

Указывает, что пакет определён пользователем (значение 0) или системой (значение 1 или выше).

### RDB$DESCRIPTION

Произвольный текст примечания к пакету.

### RDB$SQL_SECURITY

С какими правами выполняются процедуры и функции пакета:

- **TRUE** — с правами определяющего пользователя (SQL SECURITY DEFINER);
- **FALSE** — с правами вызывающего пользователя (SQL SECURITY INVOKER).

### RDB$PAGES

Сведения о страницах базы данных.

**Table 297. Описание столбцов таблицы RDB$PAGES**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$PAGE_NUMBER</td>
<td>INTEGER</td>
<td>Уникальный номер физически созданной страницы базы данных.</td>
</tr>
<tr>
<td>RDB$RELATION_ID</td>
<td>SMALLINT</td>
<td>Идентификатор таблицы, для которой выделена эта страница.</td>
</tr>
<tr>
<td>RDB$PAGE_SEQUENCE</td>
<td>INTEGER</td>
<td>Последовательный номер страницы по отношению к другим страницам, выделенным для данной таблицы.</td>
</tr>
<tr>
<td>RDB$PAGE_TYPE</td>
<td>SMALLINT</td>
<td>Описывает тип страницы. Для системного использования.</td>
</tr>
</tbody>
</table>

### RDB$PROCEDURE_PARAMETERS

Описывает параметры хранимых процедур.

**Table 298. Описание столбцов таблицы RDB$PROCEDURE_PARAMETERS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$PARAMETER_NAME</td>
<td>CHAR(63)</td>
<td>Имя параметра.</td>
</tr>
<tr>
<td>RDB$PROCEDURE_NAME</td>
<td>CHAR(63)</td>
<td>Имя процедуры, в которой используется параметр.</td>
</tr>
<tr>
<td>RDB$PARAMETER_NUMBER</td>
<td>SMALLINT</td>
<td>Последовательный номер параметра.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$PARAMETER_TYPE</td>
<td>SMALLINT</td>
<td>Указывает, является ли параметр входным (значение 0) или выходным (значение 1).</td>
</tr>
<tr>
<td>RDB$FIELD_SOURCE</td>
<td>CHAR(63)</td>
<td>Имя домена, созданного пользователем (при использовании ссылки на домен вместо типа), или домена, автоматически построенного системой для параметра процедуры. Во втором случае имя будет начинаться с символов RDB$.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Текст произвольного примечания к параметру.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, является ли параметр определённым системой (значение 1 и выше) или пользователем (значение 0).</td>
</tr>
<tr>
<td>RDB$DEFAULT_VALUE</td>
<td>BLOB BLR</td>
<td>Значение по умолчанию на языке BLR.</td>
</tr>
<tr>
<td>RDB$DEFAULT_SOURCE</td>
<td>BLOB TEXT</td>
<td>Значение по умолчанию в исходном виде на языке SQL.</td>
</tr>
<tr>
<td>RDB$COLLATION_ID</td>
<td>SMALLINT</td>
<td>Идентификатор используемого порядка сортировки для символьного параметра.</td>
</tr>
<tr>
<td>RDB$NULL_FLAG</td>
<td>SMALLINT</td>
<td>Признак допустимости пустого значения NULL.</td>
</tr>
<tr>
<td>RDB$PARAMETER_MECHANISM</td>
<td>SMALLINT</td>
<td>Механизм передачи параметра: 0 — по значению; 1 — по ссылке; 2 — через дескриптор; 3 — через дескриптор BLOB.</td>
</tr>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя столбца, на которое ссылается параметр с помощью предложения TYPE OF COLUMN.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы, на которую ссылается параметр с помощью предложения TYPE OF COLUMN.</td>
</tr>
<tr>
<td>RDB$PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Имя пакета процедуры (если процедура упакованная), в которой используется параметр.</td>
</tr>
</tbody>
</table>
### RDB$PROCEDURES

Описывает хранимые процедуры.

#### Table 299. Описание столбцов таблицы RDB$PROCEDURES

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$PROCEDURE_NAME</td>
<td>CHAR(63)</td>
<td>Имя хранимой процедуры.</td>
</tr>
<tr>
<td>RDB$PROCEDURE_ID</td>
<td>SMALLINT</td>
<td>Уникальный идентификатор процедуры.</td>
</tr>
<tr>
<td>RDB$PROCEDURE_INPUTS</td>
<td>SMALLINT</td>
<td>Указывает количество входных параметров или их отсутствие (значение NULL).</td>
</tr>
<tr>
<td>RDB$PROCEDURE_OUTPUTS</td>
<td>SMALLINT</td>
<td>Указывает количество выходных параметров или их отсутствие (значение NULL).</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст примечания к процедуре.</td>
</tr>
<tr>
<td>RDB$PROCEDURE_SOURCE</td>
<td>BLOB TEXT</td>
<td>Исходный код процедуры на языке SQL.</td>
</tr>
<tr>
<td>RDB$PROCEDURE_BLR</td>
<td>BLOB BLR</td>
<td>Двоичное представление (BLR) кода процедуры.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может указывать на класс безопасности, определённый в системной таблице RDB$SECURITY_CLASSES, для применения ограничений управления доступом.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя — владелец (создатель) процедуры.</td>
</tr>
<tr>
<td>RDB$RUNTIME</td>
<td>BLOB</td>
<td>Описание метаданных процедуры. Внутреннее использование для оптимизации.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, что процедура определена пользователем (значение 0) или системой (значение 1 или выше).</td>
</tr>
<tr>
<td>RDB$PROCEDURE&gt;Type</td>
<td>SMALLINT</td>
<td>Тип процедуры:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — селективная хранимая процедура (содержит в своём составе оператор SUSPEND);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — выполняемая хранимая процедура.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$VALID_BLR</td>
<td>SMALLINT</td>
<td>Указывает, остается ли текст хранимой процедуры корректным после последнего изменения процедуры при помощи оператора ALTER PROCEDURE.</td>
</tr>
<tr>
<td>RDB$DEBUG_INFO</td>
<td>BLOB</td>
<td>Содержит отладочную информацию о переменных, используемых в хранимой процедуре.</td>
</tr>
<tr>
<td>RDB$ENGINE_NAME</td>
<td>CHAR(63)</td>
<td>Имя двигателя для использования внешних процедур. Обычно UDR.</td>
</tr>
<tr>
<td>RDB$ENTRYPOINT</td>
<td>CHAR(255)</td>
<td>Имя точки входа в библиотеке, где находится эта процедура.</td>
</tr>
<tr>
<td>RDB$PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Имя пакета, если процедура является упакованной.</td>
</tr>
<tr>
<td>RDB$PRIVATE_FLAG</td>
<td>SMALLINT</td>
<td>Для неупакованных хранимых процедур всегда NULL, для упакованных 0 — если процедура описана в заголовке пакета и 1 — если процедура описана или реализована только в теле пакета (не описана в заголовке).</td>
</tr>
<tr>
<td>RDB$SQL_SECURITY</td>
<td>BOOLEAN</td>
<td>С какими правами выполняется процедура:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE — с правами определяющего пользователя (SQL SECURITY DEFINER); FALSE — с правами вызывающего пользователя (SQL SECURITY INVOKER); NULL — привилегии выполнения наследуются от пакета.</td>
</tr>
</tbody>
</table>

**RDB$PUBLICATION_TABLES**  
Таблицы включенные в набор репликации (публикацию).

*Table 300. Описание столбцов таблицы RDB$PUBLICATION_TABLES*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$PUBLICATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя публикации.</td>
</tr>
<tr>
<td>RDB$TABLE_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы.</td>
</tr>
</tbody>
</table>
RDB$PUBLICATIONS

Публикации. Публикация — набор таблиц для репликации.

Table 301. Описание столбцов таблицы RDB$PUBLICATIONS

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$PUBLICATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя публикации.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Владелец. Имя пользователя, создавшего публикацию.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, что публикация определена пользователем (значение 0) или системой (значение 1 или выше).</td>
</tr>
<tr>
<td>RDB$ACTIVE_FLAG</td>
<td>SMALLINT</td>
<td>Активная ли публикация. 1 — публикация активна, 0 — публикация отключена.</td>
</tr>
<tr>
<td>RDB$AUTO_ENABLE</td>
<td>SMALLINT</td>
<td>Признак автоматического добавления новых таблиц в публикацию. 1 — новые таблицы автоматически добавляются в публикацию, 0 — не добавляются (требуется ручное добавление).</td>
</tr>
</tbody>
</table>

В Firebird 4.0 может быть только одна системная публикация — публикация по умолчанию с именем RDB$DEFAULT. В следующий версиях Firebird будет возможность создавать несколько пользовательских публикаций.

RDB$REF_CONSTRAINTS

Описания именованных ограничений базы данных (внешних ключей).

Table 302. Описание столбцов таблицы RDB$REF_CONSTRAINTS

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$CONSTRAINT_NAME</td>
<td>CHAR(63)</td>
<td>Имя ограничения внешнего ключа. Задаётся пользователем или автоматически генерируется системой.</td>
</tr>
<tr>
<td>RDB$CONST_NAME_UQ</td>
<td>CHAR(63)</td>
<td>Имя ограничения первичного или уникального ключа, на которое ссылается предложение REFERENCES в данном ограничении.</td>
</tr>
</tbody>
</table>
### RDB$RELATION_CONSTRAINTS

Описание всех ограничений на уровне таблиц: первичного, уникального, внешнего ключей, ограничений CHECK, NOT NULL.

**Table 303. Описание столбцов таблицы RDB$RELATION_CONSTRAINTS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$CONSTRAINT_NAME</td>
<td>CHAR(63)</td>
<td>Имя ограничения на уровне таблицы, заданное пользователем или автоматически присвоенное системой.</td>
</tr>
<tr>
<td>RDB$CONSTRAINT_TYPE</td>
<td>CHAR(11)</td>
<td>Содержит название типа ограничения: PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK, NOT NULL.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы, к которой применяется это ограничение.</td>
</tr>
<tr>
<td>RDB$DEFERRABLE</td>
<td>CHAR(3)</td>
<td>В настоящий момент во всех случаях NO.</td>
</tr>
<tr>
<td>RDB$INITIALLY_DEFERRED</td>
<td>CHAR(3)</td>
<td>В настоящий момент во всех случаях NO.</td>
</tr>
<tr>
<td>RDB$INDEX_NAME</td>
<td>CHAR(63)</td>
<td>Имя индекса, который поддерживает это ограничение (содержит NULL, если ограничением является CHECK или NOT NULL).</td>
</tr>
</tbody>
</table>
## RDB$RELATION_FIELDS

Характеристики столбцов таблиц и представлений.

**Table 304. Описание столбцов таблицы RDB$RELATION_FIELDS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя столбца.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы (представления), где присутствует описываеый столбец.</td>
</tr>
<tr>
<td>RDB$FIELD_SOURCE</td>
<td>CHAR(63)</td>
<td>Содержит имя домена (определяённого пользователем или созданного автоматически системой), на котором основывается данный столбец.</td>
</tr>
<tr>
<td>RDB$QUERY_NAME</td>
<td>CHAR(63)</td>
<td>В настоящей версии системы не используется.</td>
</tr>
<tr>
<td>RDB$BASE_FIELD</td>
<td>CHAR(63)</td>
<td>Только для представления. Имя столбца из базовой таблицы</td>
</tr>
<tr>
<td>RDB$EDIT_STRING</td>
<td>VARCHAR(127)</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$FIELD_POSITION</td>
<td>SMALLINT</td>
<td>Позиция столбца в таблице или представлении. Нумерация начинается с 0.</td>
</tr>
<tr>
<td>RDB$QUERY_HEADER</td>
<td>BLOB TEXT</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$UPDATE_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, является ли столбец обычным столбцом (значение 1) или вычисляемым (значение 0).</td>
</tr>
<tr>
<td>RDB$FIELD_ID</td>
<td>SMALLINT</td>
<td>В настоящей версии системы в точности соответствует значению в столбце RDB$FIELD_POSITION.</td>
</tr>
<tr>
<td>RDB$VIEW_CONTEXT</td>
<td>SMALLINT</td>
<td>Для столбца представления это внутренний идентификатор базовой таблицы, откуда приходит это поле.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Примечание к столбцу таблицы или представления.</td>
</tr>
<tr>
<td>RDB$DEFAULT_VALUE</td>
<td>BLOB BLR</td>
<td>Записанное в двоичном виде (BLR) значение по умолчанию — предложение DEFAULT, если оно присутствует при описании столбца таблицы (представления).</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, определено пользователем (значение 0) или системой (значение 1 или выше).</td>
</tr>
</tbody>
</table>
### Наименование столбца | Тип данных | Описание
--- | --- | ---
RDB$SECURITY_CLASS | CHAR(63) | Может ссылаться на класс безопасности, определённый в RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого столбца.
RDB$COMPLEX_NAME | CHAR(63) | Не используется.
RDB$NULL_FLAG | SMALLINT | Указывает, допускает ли столбец значения NULL (значение NULL) или не допускает (значение 1).
RDB$DEFAULT_SOURCE | BLOB TEXT | Исходный текст предложения DEFAULT, если присутствует.
RDB$COLLATION_ID | SMALLINT | Идентификатор последовательности сортировки в составе набора символов для столбца не по умолчанию.
RDB$GENERATOR_NAME | CHAR(63) | Имя внутреннего генератора для реализации identity столбца.
RDB$IDENTITY_TYPE | SMALLINT | Для Identity столбцов определённых GENERATED BY DEFAULT хранит значение 0, GENERATED ALWAYS хранит значение 1. Для не Identity столбцов хранит NULL.

### RDB$RELATIONS

Хранит некоторые характеристики таблиц и представлений.

*Table 305. Описание столбцов таблицы RDB$RELATIONS*

| Наименование столбца | Тип данных | Описание |
--- | --- | ---
RDB$VIEW_BLR | BLOB BLR | Для представления содержит на языке BLR спецификации запроса. Для таблицы в поле содержится NULL. |
RDB$VIEW_SOURCE | BLOB TEXT | Для представления содержит оригинальный исходный текст запроса на языке SQL (включая пользовательские комментарии). Для таблицы в поле содержится NULL. |
RDB$DESCRIPTION | BLOB TEXT | Произвольный текст примечания к таблице (представлению). |
<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$RELATION_ID</td>
<td>SMALLINT</td>
<td>Внутренний идентификатор таблицы (представления).</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Указывает, создана ли таблица (представление) пользователем (значение 0) или системой (значение 1 или выше).</td>
</tr>
<tr>
<td>RDB$DBKEY_LENGTH</td>
<td>SMALLINT</td>
<td>Общая длина ключа. Для таблицы это 8 байтов. Для представления это 8, умноженное на количество таблиц, на которые ссылается представление.</td>
</tr>
<tr>
<td>RDB$FORMAT</td>
<td>SMALLINT</td>
<td>Внутреннее использование.</td>
</tr>
<tr>
<td>RDB$FIELD_ID</td>
<td>SMALLINT</td>
<td>Количество столбцов в таблице (представлении).</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы или представления.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой таблицы (представления).</td>
</tr>
<tr>
<td>RDB$EXTERNAL_FILE</td>
<td>VARCHAR(255)</td>
<td>Полный путь к внешнему файлу данных, если таблица описана с предложением EXTERNAL_FILE.</td>
</tr>
<tr>
<td>RDB$RUNTIME</td>
<td>BLOB</td>
<td>Описание метаданных таблицы. Внутреннее использование для оптимизации.</td>
</tr>
<tr>
<td>RDB$EXTERNAL_DESCRIPTION</td>
<td>BLOB</td>
<td>Произвольное примечание к внешнему файлу таблицы.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(31)</td>
<td>Имя пользователя — владелец (создателя) таблицы или представления.</td>
</tr>
<tr>
<td>RDB$DEFAULT_CLASS</td>
<td>CHAR(31)</td>
<td>Класс безопасности по умолчанию. Применяется, когда новый столбец добавляется в таблицу.</td>
</tr>
<tr>
<td>RDB$FLAGS</td>
<td>SMALLINT</td>
<td>Внутренние флаги.</td>
</tr>
</tbody>
</table>
### Наименование столбца | Тип данных | Описание
--- | --- | ---
RDB$RELATION_TYPE | SMALLINT | Тип описываемого объекта:
• 0 – постоянная таблица созданная пользователем или системная таблица;
• 1 – представление;
• 2 – внешняя таблица;
• 3 – виртуальная таблица (таблицы мониторинга MON$, псевдотаблицы безопасности SEC$);
• 4 – GTT уровня соединения (PRESERVE ROWS);
• 5 – GTT уровня транзакции (DELETE ROWS).

RDB$SQL_SECURITY | BOOLEAN | С какими правами вычисляются вычисляемые столбцы:
TRUE — с правами определяющего пользователя (SQL SECURITY DEFINER);
FALSE или NULL — с правами вызывающего пользователя (SQL SECURITY INVOKER).

### RDB$ROLES
Определение ролей.

**Table 306. Описание столбцов таблицы RDB$ROLES**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$ROLE_NAME</td>
<td>CHAR(63)</td>
<td>Имя роли.</td>
</tr>
<tr>
<td>RDB$OWNER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя-владельца роли.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст примечания к роли.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Системный флаг.</td>
</tr>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой роли.</td>
</tr>
</tbody>
</table>
### RDB$SYSTEM_PRIVILEGES

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$SYSTEM_PRIVILEGES</td>
<td>BINARY(8)</td>
<td>Битовый набор с системными привилегиями, предоставленными роли, со следующими битами:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - не используется</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - USER_MANAGEMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - READ_RAW_PAGES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - CREATE_USER_TYPES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 - USE_NBACKUP UTILITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 - CHANGE_SHUTDOWN_MODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 - TRACE_ANY_ATTACHMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 - MONITOR_ANY_ATTACHMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 - ACCESS_SHUTDOWN_DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 - CREATE_DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 - DROP_DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 - USE_GBAKUTILITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 - USE_GSTATUTILITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 - USE_GFIXUTILITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 - IGNORE_DB_TRIGGERS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 - CHANGE_HEADER_SETTINGS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 - SELECT_ANY_OBJECT_IN_DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 - ACCESS_ANY_OBJECT_IN_DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 - MODIFY_ANY_OBJECT_IN_DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 - CHANGE_MAPPING_RULES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 - USE_GRANTED_BY_CLAUSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21 - GRANT_REVOKE_ON_ANY_OBJECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22 - GRANT_REVOKE_ANY_DDL_RIGHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23 - CREATE_PRIVILEGED_ROLES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 - GET_DBCRYPT_INFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 - MODIFY_EXT_CONN_POOL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26 - REPLICATE INTO DATABASE</td>
</tr>
</tbody>
</table>

### RDB$SECURITY_CLASSES

Списки управления доступом.

*Table 307. Описание столбцов таблицы RDB$SECURITY_CLASSES*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$SECURITY_CLASS</td>
<td>CHAR(63)</td>
<td>Имя класса безопасности.</td>
</tr>
<tr>
<td>RDB$ACL</td>
<td>BLOB ACL</td>
<td>Список управления доступом, связанный с классом безопасности. Перечисляет пользователей и их полномочия.</td>
</tr>
</tbody>
</table>
**RDB$DESCRIPTION**
Произвольный текст примечания к классу безопасности.

**RDB$TIME_ZONES**
Виртуальная таблица со списком часовых поясов поддерживаемых сервером.

*Table 308. Описание столбцов таблицы RDB$TIME_ZONES*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$TIME_ZONE_ID</td>
<td>INTEGER</td>
<td>Идентификатор часового пояса.</td>
</tr>
<tr>
<td>RDB$TIME_ZONE_NAME</td>
<td>CHAR(63)</td>
<td>Наименование часового пояса</td>
</tr>
</tbody>
</table>

**RDB$TRANSACTIONS**
RDB$TRANSACTIONS хранит состояние распределённых и других транзакций, которые подготовлены для двухфазного подтверждения с явно подготовленным сообщением.

*Table 309. Описание столбцов таблицы RDB$TRANSACTIONS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Уникальный идентификатор отслеживаемой транзакции.</td>
</tr>
<tr>
<td>RDB$TRANSACTION_STATE</td>
<td>SMALLINT</td>
<td>Состояние транзакции: 0 — зависшая; 1 — подтверждённая; 2 — отменённая.</td>
</tr>
<tr>
<td>RDB$TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Не используется.</td>
</tr>
<tr>
<td>RDB$TRANSACTION_DESCRIPTION</td>
<td>BLOB</td>
<td>Описывает подготовленную транзакцию и может быть поступающее пользовательское сообщение isc_prepare_transaction2 даже если это не распределённая транзакция. Может быть использовано в случае потери соединения, которое не может быть восстановлено.</td>
</tr>
</tbody>
</table>

**RDB$TRIGGER_MESSAGES**
Сообщения тритеров.

*Table 310. Описание столбцов таблицы RDB$TRIGGER_MESSAGES*
### RDB$TRIGGERS

Описания триггеров.

**Table 311. Описание столбцов таблицы RDB$TRIGGERS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$TRIGGER_NAME</td>
<td>CHAR(63)</td>
<td>Имя триггера.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы или представления, для которого используется триггер. Если триггер применяется не к событию таблицы, а к событию базы данных, то в этом поле находится <code>NULL</code>.</td>
</tr>
<tr>
<td>RDB$TRIGGER_SEQUENCE</td>
<td>SMALLINT</td>
<td>Последовательность (позиция) триггера. Ноль обычно означает, что последовательность не задана.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$TRIGGER_TYPE</td>
<td>BIGINT</td>
<td>Событие, на которое вызывается триgger:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — BEFORE INSERT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — AFTER INSERT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — BEFORE UPDATE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 — AFTER UPDATE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 — BEFORE DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 — AFTER DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 — BEFORE INSERT OR UPDATE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 — AFTER INSERT OR UPDATE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 — BEFORE INSERT OR DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26 — AFTER INSERT OR DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27 — BEFORE UPDATE OR DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28 — AFTER UPDATE OR DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>113 — BEFORE INSERT OR UPDATE OR DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>114 — AFTER INSERT OR UPDATE OR DELETE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8192 — ON CONNECT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8193 — ON DISCONNECT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8194 — ON TRANSACTION START;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8195 — ON TRANSACTION COMMIT;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8196 — ON TRANSACTION ROLLBACK.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Описание событий DDL триггеров смотри ниже.</td>
</tr>
<tr>
<td>RDB$TRIGGER_SOURCE</td>
<td>BLOB TEXT</td>
<td>Хранит исходный код триггера в PSQL.</td>
</tr>
<tr>
<td>RDB$TRIGGER_BLR</td>
<td>BLOB BLR</td>
<td>Хранит триггер в двоичном коде BLR.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Текст примечания триггера.</td>
</tr>
<tr>
<td>RDB$TRIGGER_INACTIVE</td>
<td>SMALLINT</td>
<td>Указывает, является ли триггер в настоящее время неактивным (1) или активным (0).</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Признак — триггер определён пользователем (0) или системой (1 или выше).</td>
</tr>
<tr>
<td>RDB$FLAGS</td>
<td>SMALLINT</td>
<td>Внутреннее использование.</td>
</tr>
<tr>
<td>RDB$VALID_BLR</td>
<td>SMALLINT</td>
<td>Указывает, остаётся ли текст триггера корректным после последнего изменения триггера при помощи оператора ALTER TRIGGER.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>RDB$DEBUG_INFO</td>
<td>BLOB</td>
<td>Содержит отладочную информацию о переменных, используемых в триггере.</td>
</tr>
<tr>
<td>RDB$ENGINE_NAME</td>
<td>CHAR(63)</td>
<td>Имя движка для использования внешних триггеров. Обычно UDR.</td>
</tr>
<tr>
<td>RDB$ENTRYPOINT</td>
<td>CHAR(255)</td>
<td>Имя точки входа в библиотеке, где находится этот триггер.</td>
</tr>
<tr>
<td>RDB$SQL_SECURITY</td>
<td>BOOLEAN</td>
<td>С какими правами выполняется триггер:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE — с правами определяющего пользователя (SQL SECURITY DEFINER);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE — с правами вызывающего пользователя (SQL SECURITY INVOKER);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL — привилегии выполнения наследуются от таблицы.</td>
</tr>
</tbody>
</table>

Для DDL триггеров тип триггера (RDB$TRIGGER_TYPE) получается путём побитового ИЛИ над фазой события (0 - BEFORE, 1 - AFTER) и всех перечисленных типов событий:

- CREATE TABLE — 0x0000000000004002;
- ALTER TABLE — 0x0000000000004004;
- DROP TABLE — 0x0000000000004008;
- CREATE PROCEDURE — 0x0000000000004010;
- ALTER PROCEDURE — 0x0000000000004020;
- DROP PROCEDURE — 0x0000000000004040;
- CREATE FUNCTION — 0x0000000000004080;
- ALTER FUNCTION — 0x0000000000004100;
- DROP FUNCTION — 0x0000000000004200;
- CREATE TRIGGER — 0x0000000000004400;
- ALTER TRIGGER — 0x0000000000004800;
- DROP TRIGGER — 0x0000000000005000;
- CREATE EXCEPTION — 0x00000000000014000;
- ALTER EXCEPTION — 0x00000000000024000;
- DROP EXCEPTION — 0x00000000000044000;
- CREATE VIEW — 0x00000000000084000;
- ALTER VIEW — 0x0000000000104000;
- DROP VIEW — 0x000000000204000;
CREATE DOMAIN — 0x0000000000404000;
ALTER DOMAIN — 0x0000000000804000;
DROP DOMAIN — 0x0000000001004000;
CREATE ROLE — 0x0000000002004000;
ALTER ROLE — 0x0000000004004000;
DROP ROLE — 0x0000000008004000;
CREATE INDEX — 0x0000000010004000;
ALTER INDEX — 0x0000000020004000;
DROP INDEX — 0x0000000040004000;
CREATE SEQUENCE — 0x0000000080004000;
ALTER SEQUENCE — 0x0000000100004000;
DROP SEQUENCE — 0x0000000200004000;
CREATE USER — 0x0000000400004000;
ALTER USER — 0x0000000800004000;
DROP USER — 0x0000001000004000;
CREATE COLLATION — 0x0000002000004000;
DROP COLLATION — 0x0000004000004000;
ALTER CHARACTER SET — 0x0000008000004000;
CREATE PACKAGE — 0x0000010000004000;
ALTER PACKAGE — 0x0000020000004000;
DROP PACKAGE — 0x0000040000004000;
CREATE PACKAGE BODY — 0x0000080000004000;
DROP PACKAGE BODY — 0x0000100000004000;
CREATE MAPPING — 0x0000200000004000;
ALTER MAPPING — 0x0000400000004000;
DROP MAPPING — 0x0000800000004000;
ANY DDL STATEMENT — 0x7FFFFFFFFFFFDFFE.

Например, триггер

BEFORE CREATE PROCEDURE OR CREATE FUNCTION будет иметь тип 0x0000000000004090,
AFTER CREATE PROCEDURE OR CREATE FUNCTION — 0x0000000000004091,
BEFORE DROP FUNCTION OR DROP EXCEPTION — 0x00000000000044200,
AFTER DROP FUNCTION OR DROP EXCEPTION — 0x00000000000044201,
BEFORE DROP TRIGGER OR DROP DOMAIN — 0x00000000000100500,
AFTER DROP TRIGGER OR DROP DOMAIN — 0x00000000000100501.
Описание перечислимых типов данных.

Table 312. Описание столбцов таблицы RDB$TYPES

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя перечисляемого типа. Совпадает с именем столбца, для которого определён данный перечислимый тип.</td>
</tr>
<tr>
<td>RDB$TYPE</td>
<td>SMALLINT</td>
<td>Задаёт идентификатор для типа. Последовательность чисел является уникальной для каждого отдельного перечислимого типа:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — таблица; 1 — представление; 2 — триггер; 3 — вычисляемый столбец; 4 — проверка; 5 — процедура.</td>
</tr>
<tr>
<td>RDB$TYPE_NAME</td>
<td>CHAR(63)</td>
<td>Текстовое представление для перечислимого типа.</td>
</tr>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольный текст примечания к перечислимому типу.</td>
</tr>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>0 — определён пользователем 1 и выше — системой.</td>
</tr>
</tbody>
</table>

Полномочия пользователей системы.

Table 313. Описание столбцов таблицы RDB$USER_PRIVILEGES
<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$GRANTOR</td>
<td>CHAR(63)</td>
<td>Имя пользователя, предоставляющего полномочие.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$PRIVILEGE</td>
<td>CHAR(6)</td>
<td>Привилегия, предоставляемая в полномочии:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A – all (все привилегии);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S – select (выборка данных);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I – insert (добавление данных);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U – update (изменение данных);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D – delete (удаление строк);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R – reference (внешний ключ);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X – execute (выполнение);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G – usage (использование);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M – membership (членство).</td>
</tr>
<tr>
<td>RDB$GRANT_OPTION</td>
<td>SMALLINT</td>
<td>Содержит ли полномочие авторизацию WITH GRANT OPTION:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – не содержит;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – содержит.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя объекта (таблица, роль, процедура) на который предоставляется полномочие.</td>
</tr>
<tr>
<td>RDB$FIELD_NAME</td>
<td>CHAR(63)</td>
<td>Имя столбца, к которому применяется привилегия на уровне столбца (только привилегии UPDATE и REFERENCES).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Если предоставляется членство в роли, то в данном столбце содержится NULL если роль предоставляется обычным образом, и D если роль предоставляется с использованием ключевого слова DEFAULT.</td>
</tr>
<tr>
<td>RDB$USER_TYPE</td>
<td>SMALLINT</td>
<td>Идентифицирует тип пользователя (или объекта), которому предоставляется привилегия:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – представление;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – триггер;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 – процедура;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 – пользователь;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 – роль;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 – функция;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 – пакет;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 – системная привилегия.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$OBJECT_TYPE</td>
<td>SMALLINT</td>
<td>Идентифицирует тип объекта, на который предоставляется привилегия:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – таблица;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – представление;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – триггер;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 – процедура;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 – исключение;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 – пользователь;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 – домен;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 – набор символов;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 – роль;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 – генератор (последовательность);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 – функция;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 – BLOB фильтр;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 – сортировка;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 – пакет.</td>
</tr>
</tbody>
</table>

**RDB$VIEW_RELATIONS**

Описывает представления.

*Table 314. Описание столбцов таблицы RDB$VIEW_RELATIONS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$VIEW_NAME</td>
<td>CHAR(63)</td>
<td>Имя представления.</td>
</tr>
<tr>
<td>RDB$RELATION_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы, представления или хранимой процедуры на которое ссылается данное представление.</td>
</tr>
<tr>
<td>RDB$VIEW_CONTEXT</td>
<td>SMALLINT</td>
<td>Псевдоним (контекст), используемый для ссылки на столбец представления. Имеет то же значение, что и псевдоним, используемый в самом тексте представления на языке BLR в операторе запроса этого представления.</td>
</tr>
<tr>
<td>RDB$CONTEXT_NAME</td>
<td>CHAR(255)</td>
<td>Текстовый вариант псевдонима, указанного в столбце RDB$VIEW_CONTEXT.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RDB$CONTEXT_TYPE</td>
<td>SMALLINT</td>
<td>Тип контекста:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – таблица;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – представление;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – хранимая процедура.</td>
</tr>
<tr>
<td>RDB$PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Имя пакета для упакованной хранимой процедуры.</td>
</tr>
</tbody>
</table>
Appendix E: Таблицы мониторинга

СУБД Firebird предоставляет возможность отслеживать работу с конкретной базой данных, выполняемую на стороне сервера. Для этих целей используются таблицы мониторинга. Таблицы мониторинга имеют префикс имени MON$. Эти таблицы являются виртуальными в том смысле, что до обращения к ним со стороны пользователя, никаких данных в них не записано. Они заполняются данными только в момент запроса пользователя (в том числе, поэтому на такие таблицы бесполезно пытаться создавать триггеры). При этом описания этих таблиц в базе данных присутствуют постоянно.

Ключевым понятием функции мониторинга является снимок активности. Снимок представляет собой текущее состояние базы данных, содержащее множество информации о самой базе данных, активных соединениях, пользователях, транзакциях, подготовленных и выполняемых запросах и т.д.

Снимок создаётся при первой выборке из любой таблицы мониторинга и сохраняется до конца текущей транзакции, чтобы запросы к множеству таблиц (например, главная-подчинённая) всегда возвращал непротиворечивые данные.

Другими словами таблицы мониторинга ведут себя подобно snapshot table stability (isc_tpb_consistency) транзакции, даже если запросы к ним выполняются в транзакции с меньшим уровнем изолированности.

Для обновления снимка, текущая транзакция должна быть завершена и таблицы мониторинга должны быть запрошены в новом контексте транзакции.

Безопасность: * Полный доступ ко всей информации, предоставляемой таблицами мониторинга, имеют SYSDBA и владелец базы данных;
* Обычные пользователи ограничены информацией о собственных соединениях, другие соединения невидимы для них.

Частый сбор информации с помощью таблиц мониторинга в сильно нагруженной среде может негативно отразиться на производительности системы.

Таблицы мониторинга

MON$ATTACHMENTS
Сведения о текущих соединениях с базой данных.

MON$CALL_STACK
Обращения к стеку активными запросами хранимых процедур и триггеров.

MON$CONTEXT_VARIABLES
Сведения о пользовательских контекстных переменных.

MON$DATABASE
Сведения о базе данных, с которой выполнено соединение.
**MON$IO_STATS**
Статистика по вводу-выводу.

**MON$MEMORY_USAGE**
Статистика использования памяти.

**MON$RECORD_STATS**
Статистика на уровне записей.

**MON$STATEMENTS**
Подготовленные к выполнению операторы.

**MON$TABLE_STATS**
Статистика на уровне таблиц.

**MON$TRANSACTIONS**
Запущенные транзакции.

**MON$ATTACHMENTS**
Сведения о текущих соединениях с базой данных.

*Table 315. Описание столбцов таблицы MON$ATTACHMENTS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$ATTACHMENT_ID</td>
<td>BIGINT</td>
<td>Идентификатор соединения.</td>
</tr>
<tr>
<td>MON$SERVER_PID</td>
<td>INTEGER</td>
<td>Идентификатор серверного процесса.</td>
</tr>
<tr>
<td>MON$STATE</td>
<td>SMALLINT</td>
<td>Состояние соединения:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — бездействующее;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — активное.</td>
</tr>
<tr>
<td>MON$ATTACHMENT_NAME</td>
<td>VARCHAR(255)</td>
<td>Строка соединения — полный путь к файлу и имя первичного файла базы данных.</td>
</tr>
<tr>
<td>MON$USER</td>
<td>CHAR(63)</td>
<td>Имя пользователя, соединённого с базой данных.</td>
</tr>
<tr>
<td>MON$ROLE</td>
<td>CHAR(63)</td>
<td>Имя роли, указанное при соединении. Если роль во время соединения не была задана, поле содержит текст NONE.</td>
</tr>
<tr>
<td>MON$REMOTE_PROTOCOL</td>
<td>VARCHAR(10)</td>
<td>Используемый сетевой протокол.</td>
</tr>
<tr>
<td>MON$REMOTE_ADDRESS</td>
<td>VARCHAR(255)</td>
<td>Адрес удалённого клиента.</td>
</tr>
<tr>
<td>MON$REMOTE_PID</td>
<td>INTEGER</td>
<td>Идентификатор клиентского процесса.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>MON$CHARACTER_SET_ID</td>
<td>SMALLINT</td>
<td>Идентификатор набора символов в соединении.</td>
</tr>
<tr>
<td>MON$TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Дата и время начала соединения.</td>
</tr>
<tr>
<td>MON$GARBAGE_COLLECTION</td>
<td>SMALLINT</td>
<td>Флаг сборки мусора (указывается в DPB при подключении): 1 - позволяет, 0 - не позволяет.</td>
</tr>
<tr>
<td>MON$REMOTE_PROCESS</td>
<td>VARCHAR(255)</td>
<td>Полный путь к файлу и имя программного файла, выполнившего данное соединение.</td>
</tr>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
<tr>
<td>MON$CLIENT_VERSION</td>
<td>VARCHAR(255)</td>
<td>Версия клиентской библиотеки.</td>
</tr>
<tr>
<td>MON$REMOTE_VERSION</td>
<td>VARCHAR(255)</td>
<td>Версия сетевого протокола.</td>
</tr>
<tr>
<td>MON$REMOTE_HOST</td>
<td>VARCHAR(255)</td>
<td>Имя удалённого клиентского хоста.</td>
</tr>
<tr>
<td>MON$REMOTE_OS_USER</td>
<td>VARCHAR(255)</td>
<td>Имя пользователя в операционной системе клиента.</td>
</tr>
<tr>
<td>MON$AUTH_METHOD</td>
<td>VARCHAR(255)</td>
<td>Метод проверки подлинности, используемый при подключении.</td>
</tr>
<tr>
<td>MON$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Флаг того, что подключение системное: 0 — пользовательское подключение; 1 — системное подключение.</td>
</tr>
<tr>
<td>MON$IDLE_TIMEOUT</td>
<td>INTEGER</td>
<td>Тайм-аут простого соединения уровня соединения. Содержит значение тайм-аута простого уровня соединения, в секундах. Если тайм-аут не установлен — 0.</td>
</tr>
<tr>
<td>MON$IDLE_TIMER</td>
<td>TIMESTAMP</td>
<td>Время истечения таймера ожидания. Содержит NULL, если тайм-аут простого соединения не установлен, или если таймер не запущен.</td>
</tr>
<tr>
<td>MON$STATEMENT_TIMEOUT</td>
<td>INTEGER</td>
<td>Тайм-аут SQL оператора уровня соединения. Содержит значение тайм-аута, установленное на уровне соединения, в миллисекундах. Если тайм-аут не установлен — 0.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$WIRE_COMPRESSED</td>
<td>BOOLEAN</td>
<td>Используется ли сжатие сетевого трафика. Если используется сжатие сетевого трафика значение равно TRUE, если не используется — FALSE. Для встроенных соединений — возвращает NULL.</td>
</tr>
<tr>
<td>MON$WIRE_ENCRYPTED</td>
<td>BOOLEAN</td>
<td>Используется ли шифрование сетевого трафика. Если используется шифрование сетевого трафика значение равно TRUE, если не используется — FALSE. Для встроенных соединений — возвращает NULL.</td>
</tr>
<tr>
<td>MON$WIRE_CRYPT_PLUGIN</td>
<td>CHAR(63)</td>
<td>Имя текущего плагина для шифрования сетевого трафика, если оно используется, в противном случае NULL.</td>
</tr>
</tbody>
</table>

**Example 473. Получение сведений о клиентских приложениях**

```sql
SELECT MON$USER, MON$REMOTE_ADDRESS, MON$REMOTE_PID, MON$TIMESTAMP
FROM MON$ATTACHMENTS
WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

**Использование MON$ATTACHMENTS для закрытия подключений**

Таблицы мониторинга доступны только для чтения. Однако в сервер встроен механизм для удаления (и только удаления) записей в таблице MON$ATTACHMENTS, что позволяет, закрыть соединение с базой данных.

- Вся текущая активность в удаляемом соединении немедленно прекращается, и все активные транзакции откатываются (тритеры на события ON DISCONNECT и ON TRANSACTION ROLLBACK не вызываются);
- Закрытое соединение вернёт приложению ошибку с кодом isc_att_shutdown;
- Последующие попытки использовать это соединение (т.е. использовать его handle в API-вызовах) вернут ошибки;
- Завершение системных соединений (MON$SYSTEM_FLAG = 1) невозможно. Сервер пропустит системные подключения затронутые оператором DELETE FROM MON$ATTACHMENTS.
Example 474. Отключение всех соединений, за исключением своего

```sql
DELETE FROM MON$ATTACHMENTS
WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

### MON$CALL_STACK

Обращения к стеку запросами хранимых процедур, хранимых функций и триггеров.

**Table 316. Описание столбцов таблицы MON$CALL_STACK**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$CALL_ID</td>
<td>BIGINT</td>
<td>Идентификатор обращения.</td>
</tr>
<tr>
<td>MON$STATEMENT_ID</td>
<td>BIGINT</td>
<td>Идентификатор верхнего уровня оператора SQL — оператора, инициировавшего цепочку обращений. По этому идентификатору можно найти запись об активном операторе в таблице MON$STATEMENTS.</td>
</tr>
<tr>
<td>MON$CALLER_ID</td>
<td>BIGINT</td>
<td>Идентификатор обращающегося триггера, хранимой функции или хранимой процедуры.</td>
</tr>
<tr>
<td>MON$OBJECT_NAME</td>
<td>CHAR(63)</td>
<td>Имя объекта PSQL.</td>
</tr>
</tbody>
</table>
| MON$OBJECT_TYPE      | SMALLINT   | Тип объекта PSQL:  
                        | 2 — триггер;  
                        | 5 — хранимая процедура;  
                        | 15 — хранимая функция. |
| MON$TIMESTAMP        | TIMESTAMP  | Дата и время старта обращения. |
| MON$SOURCE_LINE      | INTEGER    | Номер исходной строки оператора SQL, выполняющегося в настоящий момент. |
| MON$SOURCE_COLUMN    | INTEGER    | Номер исходного столбца оператора SQL, выполняющегося в настоящий момент. |
| MON$STAT_ID          | INTEGER    | Идентификатор статистики. |
| MON$PACKAGE_NAME     | CHAR(63)   | Имя пакета для упакованных процедур/функций. |

В стек вызовов не попадёт информация о вызовах при выполнении оператора EXECUTE STATEMENT.
Example 475. Получение стека вызовов для всех подключений кроме своего

WITH RECURSIVE HEAD AS (  
    SELECT CALL.MON$STATEMENT_ID, CALL.MON$CALL_ID,  
           CALL.MON$OBJECT_NAME, CALL.MON$OBJECT_TYPE  
    FROM MON$CALL_STACK CALL  
    WHERE CALL.MON$CALLER_ID IS NULL  
    UNION ALL  
    SELECT CALL.MON$STATEMENT_ID, CALL.MON$CALL_ID,  
           CALL.MON$OBJECT_NAME, CALL.MON$OBJECT_TYPE  
    FROM MON$CALL_STACK CALL  
    JOIN HEAD ON CALL.MON$CALLER_ID = HEAD.MON$CALL_ID  
)
SELECT MON$ATTACHMENT_ID, MON$OBJECT_NAME, MON$OBJECT_TYPE  
FROM HEAD  
JOIN MON$STATEMENTS STMT ON STMT.MON$STATEMENT_ID = HEAD.MON$STATEMENT_ID  
WHERE STMT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION

MON$CONTEXT_VARIABLES
Сведения о пользовательских контекстных переменных.

Table 317. Описание столбцов таблицы MON$CONTEXT_VARIABLES

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$ATTACHMENT_ID</td>
<td>BIGINT</td>
<td>Идентификатор соединения. Содержит корректное значение только для контекстных переменных уровня соединения, для переменных уровня транзакции устанавливается в NULL.</td>
</tr>
<tr>
<td>MON$TRANSACTION_ID</td>
<td>BIGINT</td>
<td>Идентификатор транзакции. Содержит корректное значение только для контекстных переменных уровня транзакции, для переменных уровня соединения устанавливается в NULL.</td>
</tr>
<tr>
<td>MON$VARIABLE_NAME</td>
<td>VARCHAR(80)</td>
<td>Имя контекстной переменной.</td>
</tr>
<tr>
<td>MON$VARIABLE_VALUE</td>
<td>VARCHAR(32765)</td>
<td>Значение контекстной переменной.</td>
</tr>
</tbody>
</table>

Example 476. Получение всех сессионных контекстных переменных для текущего подключения

SELECT VAR.MON$VARIABLE_NAME, VAR.MON$VARIABLE_VALUE
**MON$DATABASE**

Сведения о базе данных, с которой выполнено соединение.

*Table 318. Описание столбцов таблицы MON$DATABASE*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$DATABASE_NAME</td>
<td>VARCHAR(255)</td>
<td>Полный путь и имя первичного файла базы данных или псевдоним базы данных.</td>
</tr>
<tr>
<td>MON$PAGE_SIZE</td>
<td>SMALLINT</td>
<td>Размер страницы файлов базы данных в байтах.</td>
</tr>
<tr>
<td>MON$ODS_MAJOR</td>
<td>SMALLINT</td>
<td>Старшая версия ODS.</td>
</tr>
<tr>
<td>MON$ODS_MINOR</td>
<td>SMALLINT</td>
<td>Младшая версия ODS.</td>
</tr>
<tr>
<td>MON$OLDEST_TRANSACTION</td>
<td>BIGINT</td>
<td>Номер старейшей заинтересованной транзакции — OIT, Oldest Interesting Transaction.</td>
</tr>
<tr>
<td>MON$OLDEST_ACTIVE</td>
<td>BIGINT</td>
<td>Номер старейшей активной транзакции — OAT, Oldest Active Transaction.</td>
</tr>
<tr>
<td>MON$OLDEST_SNAPSHOT</td>
<td>BIGINT</td>
<td>Номер транзакции, которая была активной на момент старта транзакции OAT, транзакция OST — Oldest Snapshot Transaction.</td>
</tr>
<tr>
<td>MON$NEXT_TRANSACTION</td>
<td>BIGINT</td>
<td>Номер следующей транзакции.</td>
</tr>
<tr>
<td>MON$PAGE_BUFFERS</td>
<td>INTEGER</td>
<td>Количество страниц, выделенных в оперативной памяти для кэша.</td>
</tr>
<tr>
<td>MON$SQL_DIALECT</td>
<td>SMALLINT</td>
<td>SQL диалект базы данных: 1 или 3.</td>
</tr>
<tr>
<td>MON$SHUTDOWN_MODE</td>
<td>SMALLINT</td>
<td>Текущее состояние останова (shutdown) базы данных:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — база данных активна (online);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — останов для нескольких пользователей (multi-user shutdown);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — останов для одного пользователя (single-user shutdown);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — полный останов (full shutdown).</td>
</tr>
<tr>
<td>MON$SWEEP_INTERVAL</td>
<td>INTEGER</td>
<td>Интервал чистки (sweep interval).</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$READ_ONLY</td>
<td>SMALLINT</td>
<td>Признак, является база данных только для чтения, read only, (значение 1) или для чтения и записи, read-write (0).</td>
</tr>
<tr>
<td>MON$FORCED_WRITES</td>
<td>SMALLINT</td>
<td>Указывает, установлен ли для базы режим синхронного вывода (forced writes, значение 1) или режим асинхронного вывода (значение 0).</td>
</tr>
<tr>
<td>MON$RESERVE_SPACE</td>
<td>SMALLINT</td>
<td>Флаг, указывающий на резервирование пространства.</td>
</tr>
<tr>
<td>MON$CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>Дата и время создания базы данных.</td>
</tr>
<tr>
<td>MON$PAGES</td>
<td>BIGINT</td>
<td>Количество страниц, выделенных для базы данных на внешнем устройстве.</td>
</tr>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
<tr>
<td>MON$BACKUP_STATE</td>
<td>SMALLINT</td>
<td>Текущее физическое состояние backup:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — нормальное; 1 — заблокированное; 2 — слияние (объединение).</td>
</tr>
<tr>
<td>MON$CRYPT_STATE</td>
<td>SMALLINT</td>
<td>Текущее состояние шифрования:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — не зашифрована; 1 — зашифрована; 2 — в процессе дешифрования; 3 — в процессе шифрования.</td>
</tr>
<tr>
<td>MON$CRYPT_PAGE</td>
<td>BIGINT</td>
<td>Количество зашифрованных/декшифрованных страниц в процессе шифрования/дешифрования; ноль если этот процесс закончился или не начался.</td>
</tr>
<tr>
<td>MON$OWNER</td>
<td>CHAR(63)</td>
<td>Владелец базы данных.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$SEC_DATABASE</td>
<td>CHAR(7)</td>
<td>Отображает, какой тип базы данных безопасности используется: Default — база данных безопасности по умолчанию, т.е. security4.fdb; Self — в качестве базы данных безопасности используется текущая база данных; Other — в качестве базы данных безопасности используется другая база данных (не сама и не security4.fdb).</td>
</tr>
<tr>
<td>MON$GUID</td>
<td>CHAR(38)</td>
<td>GUID базы данных.</td>
</tr>
<tr>
<td>MON$FILE_ID</td>
<td>VARCHAR(255)</td>
<td>Уникальный идентификатор базы данных на уровне файловой системы.</td>
</tr>
<tr>
<td>MON$NEXT_ATTACHMENT</td>
<td>BIGINT</td>
<td>Номер (идентификатор) следующего соединения.</td>
</tr>
<tr>
<td>MON$NEXT_STATEMENT</td>
<td>BIGINT</td>
<td>Номер (идентификатор) следующего SQL запроса.</td>
</tr>
<tr>
<td>MON$REPLICA_MODE</td>
<td>SMALLINT</td>
<td>Режим репликации: 0 - NONE — база данных является первичной; 1 - READ-ONLY — реплика в режиме только чтение; 2 - READ-WRITE — реплика в режиме чтение и запись.</td>
</tr>
</tbody>
</table>

**MON$IO_STATS**

Статистика по вводу-выводу.

*Table 319. Описание столбцов таблицы MON$IO_STATS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
</tbody>
</table>
### MON$MEMORY_USAGE

Статистика использования памяти.

#### Table 320. Описание столбцов таблицы MON$MEMORY_USAGE

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
<tr>
<td>MON$STAT_GROUP</td>
<td>SMALLINT</td>
<td>Группа статистики:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — база данных (database);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — соединение с базой данных (connection);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — транзакция (transaction);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — оператор (statement);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 — вызов (call).</td>
</tr>
<tr>
<td>MON$MEMORY_USED</td>
<td>BIGINT</td>
<td>Количество используемой памяти, байт. Информация о высокоуровневом распределении памяти, выполненной сервером из пулов. Может быть полезна для отслеживания утечек памяти и чрезмерного потребления памяти в соединениях, процедурах и тд.</td>
</tr>
</tbody>
</table>
Наименование столбца | Тип данных | Описание
--- | --- | ---
MON$MEMORY_ALLOCATED | BIGINT | Количество памяти, выделенной ОС, байт. Информация о низкоуровневом распределении памяти, выполненному менеджером памяти Firebird — объем памяти, выделенный операционной системой, что позволяет контролировать физическое потребление памяти. Обратите внимание, не все записи этого столбца имеют ненулевые значения. Малые выделения памяти здесь не фиксируются, а вместо этого добавляются к пулу памяти базы данных. Только MON$DATABASE (MON$STAT_GROUP = 0) и связанные с выделением памяти объекты имеют ненулевое значение.

MON$MAX_MEMORY_USED | BIGINT | Максимальное количество байт, используемое данным объектом.

MON$MAX_MEMORY_ALLOCATED | BIGINT | Максимальное количество байт, выделенное ОС данному объекту.

Счётчики, связанные с записями уровня базы данных MON$DATABASE (MON$STAT_GROUP = 0), отображают выделение памяти для всех соединений. В архитектурах Classic и SuperClassic нулевые значения счётчиков обозначают, что в этих архитектурах нет общего кэша.

Example 477. Получение 10 запросов потребляющих наибольшее количество памяти

```
SELECT STMT.MON$ATTACHMENT_ID, STMT.MON$SQL_TEXT, MEM.MON$MEMORY_USED
FROM MON$MEMORY_USAGE MEM
  NATURAL JOIN MON$STATEMENTS STMT
ORDER BY MEM.MON$MEMORY_USED DESC
FETCH FIRST 10 ROWS ONLY
```

**MON$RECORD_STATS**

Статистика на уровне записей.

Table 321. Описание столбцов таблицы MON$RECORD_STATS

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$STAT_GROUP</td>
<td>SMALLINT</td>
<td>Группа статистики:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — база данных (database); 1 — соединение с базой данных (connection); 2 — транзакция (transaction); 3 — оператор (statement); 4 — вызов (call).</td>
</tr>
<tr>
<td>MON$RECORD_SEQ_READS</td>
<td>BIGINT</td>
<td>Количество последовательно считанных записей (read sequentially).</td>
</tr>
<tr>
<td>MON$RECORD_IDX_READS</td>
<td>BIGINT</td>
<td>Количество записей, прочитанных при помощи индекса (read via an index).</td>
</tr>
<tr>
<td>MON$RECORD_INSERTS</td>
<td>BIGINT</td>
<td>Количество добавленных записей (inserted records).</td>
</tr>
<tr>
<td>MON$RECORD_UPDATES</td>
<td>BIGINT</td>
<td>Количество изменённых записей (updated records).</td>
</tr>
<tr>
<td>MON$RECORD_DELETES</td>
<td>BIGINT</td>
<td>Количество удалённых записей (deleted records).</td>
</tr>
<tr>
<td>MON$RECORD_BACKOUTS</td>
<td>BIGINT</td>
<td>Количество удалений версий записей созданных при rollback (backed out records).</td>
</tr>
<tr>
<td>MON$RECORD_PURGES</td>
<td>BIGINT</td>
<td>Количество удалений старых версий записей (purged records).</td>
</tr>
<tr>
<td>MON$RECORD_EXPUNGES</td>
<td>BIGINT</td>
<td>Количество удалений всей цепочки версий записи, если самая последняя версия удалена, и не нужна другим транзакциям (expunged records).</td>
</tr>
<tr>
<td>MON$RECORD_LOCKS</td>
<td>BIGINT</td>
<td>Количество записей прочитанных с использованием предложения WITH LOCK.</td>
</tr>
<tr>
<td>MON$RECORD_WAITS</td>
<td>BIGINT</td>
<td>Количество попыток обновления/модификации/блокировки записей принадлежащих нескольким активным транзакциям. Транзакция находится в режиме WAIT.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$RECORD_CONFLICTS</td>
<td>BIGINT</td>
<td>Количество неудачных попыток обновления/модификации/блокировки записей принадлежащих нескольким активным транзакциям. В таких ситуациях сообщается о конфликте обновления (UPDATE CONFLICT).</td>
</tr>
<tr>
<td>MON$BACKVERSION_READS</td>
<td>BIGINT</td>
<td>Количество прочитанных версий при поиске видимых версий записей.</td>
</tr>
<tr>
<td>MON$FRAGMENT_READS</td>
<td>BIGINT</td>
<td>Количество прочитанных фрагментов записей.</td>
</tr>
<tr>
<td>MON$RECORD_RPT_READS</td>
<td>BIGINT</td>
<td>Количество повторно прочитанных записей.</td>
</tr>
<tr>
<td>MON$RECORD_IMGC</td>
<td>BIGINT</td>
<td>Количество записей вычищенных промежуточной сборкой мусора.</td>
</tr>
</tbody>
</table>

Счётчики этой таблицы являются накопительными и накапливают информацию по каждой из групп статистики.

**MON$STATEMENTS**

Подготовленные к выполнению операторы.

*Table 322. Описание столбцов таблицы MON$STATEMENTS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$STATEMENT_ID</td>
<td>BIGINT</td>
<td>Идентификатор оператора.</td>
</tr>
<tr>
<td>MON$ATTACHMENT_ID</td>
<td>BIGINT</td>
<td>Идентификатор соединения.</td>
</tr>
<tr>
<td>MON$TRANSACTION_ID</td>
<td>BIGINT</td>
<td>Идентификатор транзакции.</td>
</tr>
<tr>
<td>MON$STATE</td>
<td>SMALLINT</td>
<td>Состояние оператора: 0 — бездействующий (idle); 1 — выполняемый (active); 2 — приостановленный (stalled).</td>
</tr>
<tr>
<td>MON$TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Дата и время старта оператора.</td>
</tr>
<tr>
<td>MON$SQL_TEXT</td>
<td>BLOB TEXT</td>
<td>Текст оператора на языке SQL.</td>
</tr>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
<tr>
<td>MON$EXPLAINED_PLAN</td>
<td>BLOB TEXT</td>
<td>План оператора в explain форме.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$STATEMENT_TIMEOUT</td>
<td>INTEGER</td>
<td>Тайм-аут SQL оператора уровня SQL оператора. Содержит значение таймаута, установленного на уровне соединения/оператора, в миллисекундах. Если тайм-аут не установлен — 0.</td>
</tr>
<tr>
<td>MON$STATEMENT_TIMER</td>
<td>TIMESTAMP</td>
<td>Время истечения таймера SQL оператора. Содержит NULL, если тайм-аут SQL оператора не установлен, или если таймер не запущен.</td>
</tr>
</tbody>
</table>

Состояние оператора STALLED — это состояние “приостановлено”. Возможно для запроса, который начал своё выполнение, ещё не завершил его, но в данный момент не выполняется. Например, ждёт входных параметров или очередного фетча (fetch) от клиента.

**Example 478. Отображение активных запросов за исключением тех, что выполняются в своём соединении**

```
SELECT ATT.MON$USER, ATT.MON$REMOTE_ADDRESS, STMT.MON$SQL_TEXT, STMT.MON$TIMESTAMP
FROM MON$ATTACHMENTS ATT
JOIN MON$STATEMENTS STMT ON ATT.MON$ATTACHMENT_ID = STMT.MON$ATTACHMENT_ID
WHERE ATT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
AND STMT.MON$STATE = 1
```

**Использование MON$STATEMENTS для отмены запросов**

Таблицы мониторинга доступны только для чтения. Однако в сервер встроен механизм для удаления (и только удаления) записей в таблице MON$STATEMENTS, что позволяет завершить активный запрос.

- Попытка отмены запросов не выполняется, если в соединении в настоящее время нет никаких выполняющихся операторов.
- После отмены запроса вызов API-функций execute/fetch вернёт ошибку с кодом isc_cancelled.
- Последующие запросы в данном соединении не запрещены.
- Отмена запроса не происходит синхронно, оператор лишь помечает запрос на отмену, а сама отмена производится ядром асинхронно.

**Example 479. Отмена всех активных запросов для заданного соединения**

```
DELETE FROM MON$STATEMENTS
WHERE MON$ATTACHMENT_ID = 32
```
**MON$TABLE_STATS**

Статистика на уровне таблицы.

*Table 323. Описание столбцов таблицы MON$TABLE_STATS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
<tr>
<td>MON$STAT_GROUP</td>
<td>SMALLINT</td>
<td>Группа статистики:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — база данных (database);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — соединение с базой данных (connection);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — транзакция (transaction);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — оператор (statement);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 — вызов (call).</td>
</tr>
<tr>
<td>MON$TABLE_NAME</td>
<td>CHAR(63)</td>
<td>Имя таблицы.</td>
</tr>
<tr>
<td>MON$RECORD_STAT_ID</td>
<td>INTEGER</td>
<td>Ссылка на MON$RECORD_STATS.</td>
</tr>
</tbody>
</table>

*Example 480. Получение статистики на уровне записей по каждой таблице для своего соединения*

```sql
SELECT t.mon$table_name,
       r.mon$record_inserts,
       r.mon$record_updates,
       r.mon$record_deletes,
       r.mon$record_backouts,
       r.mon$record_purges,
       r.mon$record_expunges,
       r.mon$record_seq_reads,
       r.mon$record_idx_reads,
       r.mon$record_rpt_reads,
       r.mon$backversion_reads,
       r.mon$fragment_reads,
       r.mon$record_locks,
       r.mon$record_waits,
       r.mon$record_conflicts,
       a.mon$stat_id
FROM mon$record_stats r
JOIN mon$table_stats t ON r.mon$stat_id = t.mon$record_stat_id
JOIN mon$attachments a ON t.mon$stat_id = a.mon$stat_id
WHERE a.mon$attachment_id = CURRENT_CONNECTION
```
**MON$TRANSACTIONS**

Описывает начатые транзакции

*Table 324. Описание столбцов таблицы MON$TRANSACTIONS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$TRANSACTION_ID</td>
<td>BIGINT</td>
<td>Идентификатор (номер) транзакции.</td>
</tr>
<tr>
<td>MON$ATTACHMENT_ID</td>
<td>BIGINT</td>
<td>Идентификатор соединения.</td>
</tr>
<tr>
<td>MON$STATE</td>
<td>SMALLINT</td>
<td>Состояние транзакции:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — бездействующая (транзакция не имеет связанных с ней запросов);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — активная (есть хотя бы один запрос связанный с транзакцией).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Запрос связывается с транзакцией, когда начинает его выполнение. Эта связь разрывается, когда запрос начинает новое выполнение в другой транзакции, или, когда транзакция или запрос удаляется, но не тогда, когда запрос выполнен или из курсора выбраны все записи.</td>
</tr>
<tr>
<td>MON$TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Дата и время старта транзакции.</td>
</tr>
<tr>
<td>MON$TOP_TRANSACTION</td>
<td>INTEGER</td>
<td>Верхний предел используемый транзакцией чистильщика (sweeper) при продвижении глобального OIT. Все транзакции выше этого порога считаются активными. Обычно он эквивалентен MON$TRANSACTION_ID, но использование COMMIT RETAINING или ROLLBACK RETAINING приводит к тому, что MON$TOP_TRANSACTION останется неизменным (“зависшим”) при увеличении идентификатора транзакции.</td>
</tr>
<tr>
<td>MON$OLDEST_TRANSACTION</td>
<td>INTEGER</td>
<td>Номер старейшей заинтересованной транзакции — OIT, Oldest Interesting Transaction.</td>
</tr>
<tr>
<td>MON$OLDEST_ACTIVE</td>
<td>INTEGER</td>
<td>Номер старейшей активной транзакции — OAT, Oldest Active Transaction.</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MON$ISOLATION_MODE</td>
<td>SMALLINT</td>
<td>Режим (уровень) изоляции:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — consistency (snapshot table stability);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — concurrency (snapshot);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 — read committed record version;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 — read committed no record version;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 — read committed read consistency.</td>
</tr>
<tr>
<td>MON$LOCK_TIMEOUT</td>
<td>SMALLINT</td>
<td>Время ожидания:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 — бесконечное ожидание (wait);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — транзакция no wait;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>другое число — время ожидания в секундах (lock timeout).</td>
</tr>
<tr>
<td>MON$READ_ONLY</td>
<td>SMALLINT</td>
<td>Признак, является ли транзакцией только для чтения, read only (значение 1) или для чтения и записи, read-write (0).</td>
</tr>
<tr>
<td>MON$AUTO_COMMIT</td>
<td>SMALLINT</td>
<td>Признак, используется ли автоматическое подтверждение транзакции auto-commit (значение 1) или нет (0).</td>
</tr>
<tr>
<td>MON$AUTO_UNDO</td>
<td>SMALLINT</td>
<td>Признак, используется ли автоматическая отмена транзакции auto-undo (значение 1) или нет (0). Если используется автоматическая отмена транзакции, создаётся точка сохранения уровня транзакции. Существование точки сохранения позволяет отменять изменения, если вызывается ROLLBACK, после чего транзакция просто фиксируется. Если этой точки сохранения не существует или она существует, но количество изменений очень велико, выполняется фактический ROLLBACK, и транзакция помечается в ТИР как «мертвая».</td>
</tr>
<tr>
<td>MON$STAT_ID</td>
<td>INTEGER</td>
<td>Идентификатор статистики.</td>
</tr>
</tbody>
</table>

*Example 481. Получение всех подключений, которые стартовали Read Write транзакции с уровнем изоляции выше Read Committed.*

```
SELECT
```
```
DISTINCT a.*
FROM
  mon$attachments a
JOIN mon$transactions t ON a.mon$attachment_id = t.mon$attachment_id
WHERE
  NOT(t.mon$read_only = 1 AND t.mon$isolation_mode >= 2);
```
Appendix F: Таблицы безопасности

Виртуальные таблицы безопасности имеют префикс имени SEC$. Они отображают данные из текущей базы данных безопасности. Эти таблицы являются виртуальными в том смысле, что до обращения к ним со стороны пользователя, никаких данных в них не записано. Они заполняются данными только в момент запроса пользователя. При этом описание этих таблиц в базе данных присутствуют постоянно. В этом смысле эти виртуальные таблицы подобны таблицам семейства MON$, используемых для мониторинга сервера.

Безопасность:* Полный доступ ко всей информации, предоставляемой таблицами безопасности, имеют SYSDBA и владелец базы данных;
• Обычные пользователи ограничены информацией о самих себе, другие пользователи невидимы для них.

Эти функции во многом зависят от плагина управления пользователями. Имейте в виду, что некоторые опции игнорируются при использовании устаревшего плагина управления пользователями.

Виртуальные таблицы безопасности

**SEC$GLOBAL_AUTH_MAPPING**

Сведения о глобальных отображениях.

**SEC$USERS**

Список пользователей в текущей базе данных безопасности.

**SEC$USER_ATTRIBUTES**

Сведения о дополнительных атрибутах пользователей.

**SEC$GLOBAL_AUTH_MAPPING**

Сведения о глобальных отображениях.

*Table 325. Описание столбцов таблицы SEC$GLOBAL_AUTH_MAPPING*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$MAP_NAME</td>
<td>CHAR(63)</td>
<td>Имя глобального отображения.</td>
</tr>
<tr>
<td>SEC$MAP_USING</td>
<td>CHAR(1)</td>
<td>Является ли аутентификация общесерверной (S) или обычной (P).</td>
</tr>
<tr>
<td>SEC$MAP_PLUGIN</td>
<td>CHAR(63)</td>
<td>Имя плагина аутентификации, из которого происходит отображение.</td>
</tr>
<tr>
<td>SEC$MAP_DB</td>
<td>CHAR(63)</td>
<td>Имя базы данных, в которой прошла аутентификация. Из неё происходит отображение.</td>
</tr>
<tr>
<td>SEC$MAP_FROM_TYPE</td>
<td>CHAR(63)</td>
<td>Тип объекта, который будет отображён.</td>
</tr>
</tbody>
</table>
### SEC$MAP_FROM

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$MAP_FROM</td>
<td>CHAR(255)</td>
<td>Имя объекта, из которого будет произведено отображение.</td>
</tr>
</tbody>
</table>

### SEC$MAP_TO_TYPE

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$MAP_TO_TYPE</td>
<td>SMALLINT</td>
<td>Тип объекта, в который будет произведено отображение:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — USER;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — ROLE.</td>
</tr>
</tbody>
</table>

### SEC$MAP_TO

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$MAP_TO</td>
<td>CHAR(63)</td>
<td>Наименование объекта, в который будет произведено отображение (имя пользователя или роли).</td>
</tr>
</tbody>
</table>

### RDB$SYSTEM_FLAG

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$SYSTEM_FLAG</td>
<td>SMALLINT</td>
<td>Является ли отображение системным:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 — определено пользователем;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 — определено системой.</td>
</tr>
</tbody>
</table>

### RDB$DESCRIPTION

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDB$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Произвольное текстовое описание порядка сортировки.</td>
</tr>
</tbody>
</table>

### SEC$USERS

Список пользователей в текущей базе данных безопасности.

*Table 326. Описание столбцов таблицы SEC$USERS*

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$USER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя.</td>
</tr>
<tr>
<td>SEC$FIRST_NAME</td>
<td>VARCHAR(32)</td>
<td>Первое имя (имя).</td>
</tr>
<tr>
<td>SEC$MIDDLE_NAME</td>
<td>VARCHAR(32)</td>
<td>Среднее имя (отчество).</td>
</tr>
<tr>
<td>SEC$LAST_NAME</td>
<td>VARCHAR(32)</td>
<td>Последнее имя (фамилия).</td>
</tr>
<tr>
<td>SEC$ACTIVE</td>
<td>BOOLEAN</td>
<td>Флаг активности пользователя.</td>
</tr>
<tr>
<td>SEC$ADMIN</td>
<td>BOOLEAN</td>
<td>Отражает, имеет ли пользователь права RDB$ADMIN в базе данных безопасности.</td>
</tr>
<tr>
<td>SEC$DESCRIPTION</td>
<td>BLOB TEXT</td>
<td>Комментарий к пользователю.</td>
</tr>
<tr>
<td>SEC$PLUGIN</td>
<td>CHAR(63)</td>
<td>Имя плагина управления пользователями, с помощью которого был создан данный пользователь.</td>
</tr>
</tbody>
</table>

### SEC$USER_ATTRIBUTES

Сведения о дополнительных атрибутах пользователей.
Table 327. Описание столбцов таблицы SEC$USER_ATTRIBUTES

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$USER_NAME</td>
<td>CHAR(63)</td>
<td>Имя пользователя.</td>
</tr>
<tr>
<td>SEC$KEY</td>
<td>VARCHAR(63)</td>
<td>Имя атрибута.</td>
</tr>
<tr>
<td>SEC$VALUE</td>
<td>VARCHAR(255)</td>
<td>Значение атрибута.</td>
</tr>
<tr>
<td>SEC$PLUGIN</td>
<td>CHAR(63)</td>
<td>Имя плагина управления пользователями, с помощью которого был создан данный пользователь.</td>
</tr>
</tbody>
</table>

Example 482. Отображение списка пользователей и их атрибутов

```sql
SELECT CAST(U.SEC$USER_NAME AS CHAR(20)) AS LOGIN,
       CAST(A.SEC$KEY AS CHAR(10)) AS TAG,
       CAST(A.SEC$VALUE AS CHAR(20)) AS "VALUE",
       U.SEC$PLUGIN AS "PLUGIN"
FROM SEC$USERS U LEFT JOIN SEC$USER_ATTRIBUTES A
ON U.SEC$USER_NAME = A.SEC$USER_NAME
AND U.SEC$PLUGIN = A.SEC$PLUGIN;
```

<table>
<thead>
<tr>
<th>LOGIN</th>
<th>TAG</th>
<th>VALUE</th>
<th>PLUGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDBA</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>Srp</td>
</tr>
<tr>
<td>ALEX</td>
<td>B</td>
<td>x</td>
<td>Srp</td>
</tr>
<tr>
<td>ALEX</td>
<td>C</td>
<td>sample</td>
<td>Srp</td>
</tr>
<tr>
<td>SYSDBA</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>Legacy_UserManager</td>
</tr>
</tbody>
</table>
Appendix G: Таблицы плагинов

Таблицы плагинов — это таблицы или представления, созданные для или с помощью различных плагинов для движка Firebird. Стандартные таблицы плагинов имеют префикс PLG$ (но могут иметь и другие).

Таблицы плагинов не всегда существуют. Например, некоторые таблицы существуют только в базе данных безопасности, а другие таблицы будут созданы только при первом использовании плагина.

В этом приложении описаны только таблицы плагинов, созданные плагинами, включенными в стандартную поставку Firebird 4.0.

Таблицы плагинов не считаются системными таблицами.

Плагин управления пользователями Srp

Таблица PLG$SRP

Таблица PLG$SRP хранит список пользователей и информацию для их аутентификации плагинами аутентификации семейства SRP.

Table 328. Описание столбцов таблицы PLG$SRP

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLG$USER_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя пользователя</td>
</tr>
<tr>
<td>PLG$VERIFIER</td>
<td>VARBINARY(128)</td>
<td>Srp verifier</td>
</tr>
<tr>
<td>PLG$SALT</td>
<td>VARBINARY(32)</td>
<td>Соль</td>
</tr>
<tr>
<td>PLG$COMMENT</td>
<td>BLOB SUB_TYPE TEXT</td>
<td>Текстовый комментарий</td>
</tr>
<tr>
<td>PLG$FIRST</td>
<td>VARCHAR(32)</td>
<td>Первое имя (имя)</td>
</tr>
<tr>
<td>PLG$MIDDLE</td>
<td>VARCHAR(32)</td>
<td>Среднее имя (отчество)</td>
</tr>
<tr>
<td>PLG$LAST</td>
<td>VARCHAR(32)</td>
<td>Последнее имя (фамилия)</td>
</tr>
<tr>
<td>PLG$ATTRIBUTES</td>
<td>BLOB SUB_TYPE TEXT</td>
<td>Пользовательские атрибуты (теги)</td>
</tr>
<tr>
<td>PLG$ACTIVE</td>
<td>BOOLEAN</td>
<td>Флаг - активен ли пользователь</td>
</tr>
</tbody>
</table>

Представление PLG$SRP_VIEW

Представление PLG$SRP_VIEW определяет какие пользователи доступны для просмотра через виртуальную таблицу SEC$USERS и изменения с помощью оператор ALTER USER ….

Table 329. Описание столбцов представления PLG$SRP_VIEW

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLG$USER_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя пользователя</td>
</tr>
</tbody>
</table>
Плагин управления пользователями
Legacy_UserManager

Таблица PLG$USERS

Таблица PLG$USERS хранит список пользователей и информацию для их аутентификации плагином аутентификации Legacy_Auth.

Table 330. Описание столбцов таблицы PLG$USERS

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLG$USER_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя пользователя</td>
</tr>
<tr>
<td>PLG$GROUP_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя группы</td>
</tr>
<tr>
<td>PLG$GROUP_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя группы</td>
</tr>
<tr>
<td>PLG$UID</td>
<td>INTEGER</td>
<td>Идентификатор пользователя в POSIX</td>
</tr>
<tr>
<td>PLG$GID</td>
<td>INTEGER</td>
<td>Идентификатор группы в POSIX</td>
</tr>
<tr>
<td>Наименование столбца</td>
<td>Тип данных</td>
<td>Описание</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>PLG$PASSWD</td>
<td>VARBINARY(64)</td>
<td>Хеш пароля</td>
</tr>
<tr>
<td>PLG$COMMENT</td>
<td>BLOB SUB_TYPE TEXT</td>
<td>Текстовый комментарий</td>
</tr>
<tr>
<td>PLG$FIRST_NAME</td>
<td>VARCHAR(32)</td>
<td>Первое имя (имя)</td>
</tr>
<tr>
<td>PLG$MIDDLE_NAME</td>
<td>VARCHAR(32)</td>
<td>Среднее имя (отчество)</td>
</tr>
<tr>
<td>PLG$LAST_NAME</td>
<td>VARCHAR(32)</td>
<td>Последнее имя (фамилия)</td>
</tr>
</tbody>
</table>

**Представление PLG$VIEW_USERS**

Представление PLG$VIEW_USERS определяет какие пользователи доступны для просмотра через виртуальную таблицу SEC$USERS и изменения с помощью оператор ALTER USER ...

**Table 331. Описание столбцов представления PLG$VIEW_USERS**

<table>
<thead>
<tr>
<th>Наименование столбца</th>
<th>Тип данных</th>
<th>Описание</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLG$USER_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя пользователя</td>
</tr>
<tr>
<td>PLG$GROUP_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя группы</td>
</tr>
<tr>
<td>PLG$GROUP_NAME</td>
<td>VARCHAR(63)</td>
<td>Имя группы</td>
</tr>
<tr>
<td>PLG$UID</td>
<td>INTEGER</td>
<td>Идентификатор пользователя в POSIX</td>
</tr>
<tr>
<td>PLG$GID</td>
<td>INTEGER</td>
<td>Идентификатор группы в POSIX</td>
</tr>
<tr>
<td>PLG$PASSWD</td>
<td>VARBINARY(64)</td>
<td>Хеш пароля</td>
</tr>
<tr>
<td>PLG$COMMENT</td>
<td>BLOB SUB_TYPE TEXT</td>
<td>Текстовый комментарий</td>
</tr>
<tr>
<td>PLG$FIRST_NAME</td>
<td>VARCHAR(32)</td>
<td>Первое имя (имя)</td>
</tr>
<tr>
<td>PLG$MIDDLE_NAME</td>
<td>VARCHAR(32)</td>
<td>Среднее имя (отчество)</td>
</tr>
<tr>
<td>PLG$LAST_NAME</td>
<td>VARCHAR(32)</td>
<td>Последнее имя (фамилия)</td>
</tr>
</tbody>
</table>

Данное представление хранит следующий SQL запрос

```sql
SELECT
    PLG$USER_NAME,
    PLG$GROUP_NAME,
    PLG$UID,
    PLG$GID,
    PLG$PASSWD,
    PLG$COMMENT,
    PLG$FIRST_NAME,
    PLG$MIDDLE_NAME,
    PLG$LAST_NAME
FROM
    PLG$USERS
WHERE CURRENT_USER = 'SYSDBA'
    OR CURRENT_ROLE = 'RDB$ADMIN'
```
OR CURRENT_USER = PLG$USERS.PLG$USER_NAME
## Appendix H: Наборы символов и порядки сортировки

### Table 332. Наборы символов и порядки сортировки

<table>
<thead>
<tr>
<th>Название</th>
<th>ID</th>
<th>Байтов на символ</th>
<th>Порядок сортировки</th>
<th>Язык</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>2</td>
<td>1</td>
<td>ASCII</td>
<td>Английский</td>
</tr>
<tr>
<td>BIG_5</td>
<td>56</td>
<td>2</td>
<td>BIG_5</td>
<td>Китайский, Вьетнамский, Корейский</td>
</tr>
<tr>
<td>CP943C</td>
<td>68</td>
<td>2</td>
<td>CP943C</td>
<td>Японский.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>CP943C_UNICODE</td>
<td>Японский.</td>
</tr>
<tr>
<td>CYRL</td>
<td>50</td>
<td>1</td>
<td>CYRL</td>
<td>Русский.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_RUS</td>
<td>Русский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>PDOX_CYRL</td>
<td>Русский Paradox.</td>
</tr>
<tr>
<td>DOS437</td>
<td>10</td>
<td>1</td>
<td>DOS437</td>
<td>Английский — США.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_DEU437</td>
<td>Немецкий dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_ESP437</td>
<td>Испанский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_FIN437</td>
<td>Финский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_FRA437</td>
<td>Французский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_ITA437</td>
<td>Итальянский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_NLD437</td>
<td>Голландский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_SVE437</td>
<td>Шведский dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_UK437</td>
<td>Английский (Великобритания) dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_US437</td>
<td>Английский (США) dBase.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>PDOX_ASCII</td>
<td>Кодовая страница Paradox — ASCII.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>PDOX_SWEDFIN</td>
<td>Шведский / Финский Paradox.</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>PDOX_INTL</td>
<td>Международный английский Paradox.</td>
</tr>
<tr>
<td>DOS737</td>
<td>9</td>
<td>1</td>
<td>DOS737</td>
<td>Греческий.</td>
</tr>
<tr>
<td>DOS775</td>
<td>15</td>
<td>1</td>
<td>DOS775</td>
<td>Балтийский.</td>
</tr>
<tr>
<td>DOS850</td>
<td>11</td>
<td>1</td>
<td>DOS850</td>
<td>Латинский I (нет символа Евро).</td>
</tr>
<tr>
<td>//</td>
<td>//</td>
<td>//</td>
<td>DB_DEU850</td>
<td>Немецкий.</td>
</tr>
</tbody>
</table>
### Наборы символов и порядки сортировки

<table>
<thead>
<tr>
<th>Название</th>
<th>ID</th>
<th>Байтов на символ</th>
<th>Порядок сортировки</th>
<th>Язык</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_ESP850</td>
<td></td>
<td></td>
<td>Испанский</td>
<td></td>
</tr>
<tr>
<td>DB_FRA850</td>
<td></td>
<td></td>
<td>Французский</td>
<td></td>
</tr>
<tr>
<td>DB_FRC850</td>
<td></td>
<td></td>
<td>Французский — Канада</td>
<td></td>
</tr>
<tr>
<td>DB_ITA850</td>
<td></td>
<td></td>
<td>Итальянский</td>
<td></td>
</tr>
<tr>
<td>DB_NLD850</td>
<td></td>
<td></td>
<td>Голландский</td>
<td></td>
</tr>
<tr>
<td>DB_PTB850</td>
<td></td>
<td></td>
<td>Португальский — Бразилия</td>
<td></td>
</tr>
<tr>
<td>DB_SVE850</td>
<td></td>
<td></td>
<td>Шведский</td>
<td></td>
</tr>
<tr>
<td>DB_UK850</td>
<td></td>
<td></td>
<td>Английский — Великобритания</td>
<td></td>
</tr>
<tr>
<td>DB_US850</td>
<td></td>
<td></td>
<td>Английский — США</td>
<td></td>
</tr>
<tr>
<td>DOS852</td>
<td>45</td>
<td>1</td>
<td>Латинский II</td>
<td></td>
</tr>
<tr>
<td>DB_CSY</td>
<td></td>
<td></td>
<td>Чешский dBase</td>
<td></td>
</tr>
<tr>
<td>DB_PLK</td>
<td></td>
<td></td>
<td>Польский dBase</td>
<td></td>
</tr>
<tr>
<td>DB_SLO</td>
<td></td>
<td></td>
<td>Словацкий dBase</td>
<td></td>
</tr>
<tr>
<td>PDOX_CSY</td>
<td></td>
<td></td>
<td>Чешский Paradox</td>
<td></td>
</tr>
<tr>
<td>PDOX_HUN</td>
<td></td>
<td></td>
<td>Венгерский Paradox</td>
<td></td>
</tr>
<tr>
<td>PDOX_PLK</td>
<td></td>
<td></td>
<td>Польский Paradox</td>
<td></td>
</tr>
<tr>
<td>PDOX_SLO</td>
<td></td>
<td></td>
<td>Словацкий Paradox</td>
<td></td>
</tr>
<tr>
<td>DOS857</td>
<td>46</td>
<td>1</td>
<td>Турецкий</td>
<td></td>
</tr>
<tr>
<td>DB_TRK</td>
<td></td>
<td></td>
<td>Турецкий dBase</td>
<td></td>
</tr>
<tr>
<td>DOS858</td>
<td>16</td>
<td>1</td>
<td>Латинский I с символом Евро</td>
<td></td>
</tr>
<tr>
<td>DOS860</td>
<td>13</td>
<td>1</td>
<td>Португальский</td>
<td></td>
</tr>
<tr>
<td>DB_PTG860</td>
<td></td>
<td></td>
<td>Португальский dBase</td>
<td></td>
</tr>
<tr>
<td>DOS861</td>
<td>47</td>
<td>1</td>
<td>Исландский</td>
<td></td>
</tr>
<tr>
<td>PDOX_ISL</td>
<td></td>
<td></td>
<td>Исландский Paradox</td>
<td></td>
</tr>
<tr>
<td>DOS862</td>
<td>17</td>
<td>1</td>
<td>Иврит</td>
<td></td>
</tr>
<tr>
<td>DOS863</td>
<td>14</td>
<td>1</td>
<td>Французский — Канада</td>
<td></td>
</tr>
<tr>
<td>DB_FRC863</td>
<td></td>
<td></td>
<td>Французский dBase — Канада</td>
<td></td>
</tr>
<tr>
<td>DOS864</td>
<td>18</td>
<td>1</td>
<td>Арабский</td>
<td></td>
</tr>
<tr>
<td>DOS865</td>
<td>12</td>
<td>1</td>
<td>Скандинавские</td>
<td></td>
</tr>
<tr>
<td>DB_DAN865</td>
<td></td>
<td></td>
<td>Датский dBase</td>
<td></td>
</tr>
<tr>
<td>DB_NOR865</td>
<td></td>
<td></td>
<td>Норвежский dBase</td>
<td></td>
</tr>
<tr>
<td>Название</td>
<td>ID</td>
<td>Байтов на символ</td>
<td>Порядок сортировки</td>
<td>Язык</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
<td>-----------------</td>
<td>----------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>PDOX_NORDAN4</td>
<td>48</td>
<td>1</td>
<td>DOS866</td>
<td>Норвегия и Дания.</td>
</tr>
<tr>
<td>DOS869</td>
<td>49</td>
<td>1</td>
<td>DOS869</td>
<td>Современный греческий.</td>
</tr>
<tr>
<td>EUCJ_0208</td>
<td>6</td>
<td>2</td>
<td>EUCJ_0208</td>
<td>Японские EUC.</td>
</tr>
<tr>
<td>GB_2312</td>
<td>57</td>
<td>2</td>
<td>GB_2312</td>
<td>Упрощенный китайский (Гонконг, Корея).</td>
</tr>
<tr>
<td>GB18030</td>
<td>69</td>
<td>4</td>
<td>GB18030</td>
<td>Китайский.</td>
</tr>
<tr>
<td>GB18030_UNICODE</td>
<td></td>
<td></td>
<td>GB18030_UNICODE</td>
<td>Китайский.</td>
</tr>
<tr>
<td>GBK</td>
<td>67</td>
<td>2</td>
<td>GBK</td>
<td>Китайский.</td>
</tr>
<tr>
<td>GBK_UNICODE</td>
<td></td>
<td></td>
<td>GBK_UNICODE</td>
<td>Китайский.</td>
</tr>
<tr>
<td>ISO8859_1</td>
<td>21</td>
<td>1</td>
<td>ISO8859_1</td>
<td>Латинский I.</td>
</tr>
<tr>
<td>DA_DA</td>
<td></td>
<td></td>
<td>DA_DA</td>
<td>Датский.</td>
</tr>
<tr>
<td>DE_DE</td>
<td></td>
<td></td>
<td>DE_DE</td>
<td>Немецкий.</td>
</tr>
<tr>
<td>DU_NL</td>
<td></td>
<td></td>
<td>DU_NL</td>
<td>Голландский.</td>
</tr>
<tr>
<td>EN_UK</td>
<td></td>
<td></td>
<td>EN_UK</td>
<td>Английский, Великобритания.</td>
</tr>
<tr>
<td>EN_US</td>
<td></td>
<td></td>
<td>EN_US</td>
<td>Английский, США.</td>
</tr>
<tr>
<td>ES_ES</td>
<td></td>
<td></td>
<td>ES_ES</td>
<td>Испанский.</td>
</tr>
<tr>
<td>ES_ES_CI_AI</td>
<td></td>
<td></td>
<td>ES_ES_CI_AI</td>
<td>Испанский не чувствительный к регистру символов и к акцентам (ударению).</td>
</tr>
<tr>
<td>FI_FI</td>
<td></td>
<td></td>
<td>FI_FI</td>
<td>Финский.</td>
</tr>
<tr>
<td>FR_CA</td>
<td></td>
<td></td>
<td>FR_CA</td>
<td>Французский, Канада.</td>
</tr>
<tr>
<td>FR_FR</td>
<td></td>
<td></td>
<td>FR_FR</td>
<td>Французский.</td>
</tr>
<tr>
<td>FR_FR_CI_AI</td>
<td></td>
<td></td>
<td>FR_FR_CI_AI</td>
<td>Французский — не чувствительный к регистру символов и к акцентам (ударению).</td>
</tr>
<tr>
<td>IS_IS</td>
<td></td>
<td></td>
<td>IS_IS</td>
<td>Исландский.</td>
</tr>
<tr>
<td>IT_IT</td>
<td></td>
<td></td>
<td>IT_IT</td>
<td>Итальянский.</td>
</tr>
<tr>
<td>NO_NO</td>
<td></td>
<td></td>
<td>NO_NO</td>
<td>Норвежский.</td>
</tr>
<tr>
<td>PT_PT</td>
<td></td>
<td></td>
<td>PT_PT</td>
<td>Португальский.</td>
</tr>
<tr>
<td>PT_BR</td>
<td></td>
<td></td>
<td>PT_BR</td>
<td>Португальский, Бразилия.</td>
</tr>
<tr>
<td>SV_SV</td>
<td></td>
<td></td>
<td>SV_SV</td>
<td>Шведский.</td>
</tr>
<tr>
<td>Название</td>
<td>ID</td>
<td>Байтов на символ</td>
<td>Порядок сортировки</td>
<td>Язык</td>
</tr>
<tr>
<td>------------------</td>
<td>----</td>
<td>------------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>ISO8859_2</td>
<td>22</td>
<td>1</td>
<td>ISO8859_2</td>
<td>Латинский 2 — Центральная Европа (хорватский, чешский, венгерский, польский, румынский, сербский, словацкий, словенский).</td>
</tr>
<tr>
<td>ISO8859_3</td>
<td>23</td>
<td>1</td>
<td>ISO8859_3</td>
<td>Латинский 3 — Южная Европа (мальтийский, эсперанто).</td>
</tr>
<tr>
<td>ISO8859_4</td>
<td>34</td>
<td>1</td>
<td>ISO8859_4</td>
<td>Латинский 4 — Северная Европа (эстонский, латвийский, литовский, грэнландский, саамский).</td>
</tr>
<tr>
<td>ISO8859_5</td>
<td>35</td>
<td>1</td>
<td>ISO8859_5</td>
<td>Кириллица (русский).</td>
</tr>
<tr>
<td>ISO8859_6</td>
<td>36</td>
<td>1</td>
<td>ISO8859_6</td>
<td>Арабский.</td>
</tr>
<tr>
<td>ISO8859_7</td>
<td>37</td>
<td>1</td>
<td>ISO8859_7</td>
<td>Греческий.</td>
</tr>
<tr>
<td>ISO8859_8</td>
<td>38</td>
<td>1</td>
<td>ISO8859_8</td>
<td>Иврит.</td>
</tr>
<tr>
<td>ISO8859_9</td>
<td>39</td>
<td>1</td>
<td>ISO8859_9</td>
<td>Латинский 5.</td>
</tr>
<tr>
<td>ISO8859_13</td>
<td>40</td>
<td>1</td>
<td>ISO8859_13</td>
<td>Латинский 7 — Балтика.</td>
</tr>
<tr>
<td>KOI8R</td>
<td>63</td>
<td>1</td>
<td>KOI8R</td>
<td>Русский. Словарное упорядочение.</td>
</tr>
<tr>
<td>KOI8U</td>
<td>64</td>
<td>1</td>
<td>KOI8U</td>
<td>Украинский. Словарное упорядочение.</td>
</tr>
<tr>
<td>KSC_5601</td>
<td>44</td>
<td>2</td>
<td>KSC_5601</td>
<td>Корейский.</td>
</tr>
<tr>
<td>NEXT</td>
<td>19</td>
<td>1</td>
<td>NEXT</td>
<td>Кодирование NeXTSTEP.</td>
</tr>
<tr>
<td>KOI8U_UA</td>
<td></td>
<td></td>
<td>KOI8U_UA</td>
<td>Корейский — словарный порядок сортировки.</td>
</tr>
<tr>
<td>KOI8R_RU</td>
<td></td>
<td></td>
<td>KOI8R_RU</td>
<td>Корейский — словарный порядок сортировки.</td>
</tr>
<tr>
<td>NXT_DEU</td>
<td></td>
<td></td>
<td>NXT_DEU</td>
<td>Немецкий.</td>
</tr>
</tbody>
</table>

Appendix H: Наборы символов и порядки сортировки
<table>
<thead>
<tr>
<th>Название</th>
<th>ID</th>
<th>Байтов на символ</th>
<th>Порядок сортировки</th>
<th>Язык</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXT_ESP</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Испанский.</td>
</tr>
<tr>
<td>NXT_FRA</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Французский.</td>
</tr>
<tr>
<td>NXT_ITA</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Итальянский.</td>
</tr>
<tr>
<td>NXT_US</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Английский, США.</td>
</tr>
<tr>
<td>NONE</td>
<td>0</td>
<td>1</td>
<td>NONE</td>
<td>Нейтральная кодовая страница. Перевод в верхний регистр выполняется только для кодов ASCII 97–122. Постарайтесь сделать так, чтобы этот набор символов никогда не появлялся в столбцах ваших баз данных.</td>
</tr>
<tr>
<td>OCTETS</td>
<td>1</td>
<td>1</td>
<td>OCTETS</td>
<td>Двоичные символы.</td>
</tr>
<tr>
<td>SJIS_0208</td>
<td>5</td>
<td>2</td>
<td>SJIS_0208</td>
<td>Японский.</td>
</tr>
<tr>
<td>TIS620</td>
<td>66</td>
<td>1</td>
<td>TIS620</td>
<td>Тайский.</td>
</tr>
<tr>
<td>UNICODE_FSS</td>
<td>3</td>
<td>3</td>
<td>UNICODE_FSS</td>
<td>UNICODE</td>
</tr>
<tr>
<td>UTF8</td>
<td>4</td>
<td>4</td>
<td>UTF8</td>
<td>UNICODE 4.0.</td>
</tr>
<tr>
<td>WIN1250</td>
<td>51</td>
<td>1</td>
<td>WIN1250</td>
<td>ANSI — Центральная Европа.</td>
</tr>
<tr>
<td>BS_BA</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Боснийский.</td>
</tr>
<tr>
<td>PXW_CSY</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Чешский.</td>
</tr>
<tr>
<td>PXW_HUN</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Венгерский — не чувствительный к регистру и чувствительный к акценту (ударению).</td>
</tr>
<tr>
<td>PXW_HUNDC</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Венгерский — словарная сортировка.</td>
</tr>
<tr>
<td>PXW_PLK</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Польский.</td>
</tr>
<tr>
<td>PXW_SLOV</td>
<td>//</td>
<td>//</td>
<td>//</td>
<td>Словенский.</td>
</tr>
<tr>
<td>Название</td>
<td>ID</td>
<td>Байтов на символ</td>
<td>Порядок сортировки</td>
<td>Язык</td>
</tr>
<tr>
<td>----------</td>
<td>----</td>
<td>-----------------</td>
<td>---------------------</td>
<td>------</td>
</tr>
<tr>
<td>WIN1251</td>
<td>52</td>
<td>1</td>
<td>WIN1251</td>
<td>ANSI кириллица.</td>
</tr>
<tr>
<td>WIN1252</td>
<td>53</td>
<td>1</td>
<td>WIN1252</td>
<td>ANSI — Латинский I.</td>
</tr>
<tr>
<td>WIN1253</td>
<td>54</td>
<td>1</td>
<td>WIN1253</td>
<td>ANSI греческий.</td>
</tr>
<tr>
<td>WIN1254</td>
<td>55</td>
<td>1</td>
<td>WIN1254</td>
<td>ANSI турецкий.</td>
</tr>
<tr>
<td>WIN1255</td>
<td>58</td>
<td>1</td>
<td>WIN1255</td>
<td>ANSI иврит.</td>
</tr>
<tr>
<td>WIN1256</td>
<td>59</td>
<td>1</td>
<td>WIN1256</td>
<td>ANSI арабский.</td>
</tr>
<tr>
<td>WIN1257</td>
<td>60</td>
<td>1</td>
<td>WIN1257</td>
<td>ANSI балтийский.</td>
</tr>
<tr>
<td>WIN1258</td>
<td>65</td>
<td>1</td>
<td>WIN1258</td>
<td>Вьетнамский.</td>
</tr>
</tbody>
</table>
Appendix I: License notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the “License”); you may only use this Documentation if you comply with the terms of this License. Copies of the License are available at https://www.firebirdsql.org/pdfmanual/pdl.pdf (PDF) and https://www.firebirdsql.org/manual/pdl.html (HTML).

The Original Documentation is titled Firebird 4.0 Language Reference. This Documentation was derived from Firebird 2.5 Language Reference.

The Initial Writers of the Original Documentation are: Paul Vinkenoog, Dmitry Yemanov, Thomas Woinke and Mark Rotteveel. Writers of text originally in Russian are Denis Simonov, Dmitry Filippov, Alexander Karpeykin, Alexey Kovyazin and Dmitry Kuzmenko.

Copyright © 2008-2021. All Rights Reserved. Initial Writers contact: paul at vinkenoog dot nl.

Writers and Editors of included PDL-licensed material are: J. Beesley, Helen Borrie, Arno Brinkman, Frank Ingermann, Vlad Khorsun, Alex Peshkov, Nickolay Samofatov, Adriano dos Santos Fernandes, Dmitry Yemanov.

Included portions are Copyright © 2001-2020 by their respective authors. All Rights Reserved.

Contributor(s): Mark Rotteveel, Denis Simonov.

Portions created by Mark Rotteveel and Denis Simonov are Copyright © 2018-2021. All Rights Reserved. (Contributor contact(s): mrotteveel at users dot sourceforge dot net).
(Contributor contact(s): sim-mail at list dot ru).
Алфавитный указатель

A
ALL, 124
ALTER CHARACTER SET, 293
ALTER DATABASE, 134
ADD DIFFERENCE FILE, 135
ADD FILE, 135
BEGIN BACKUP, 136
DECRYPT, 139
DISABLE PUBLICATION, 139
DROP DIFFERENCE FILE, 136
DROP LINGER, 138
ENABLE PUBLICATION, 139
ENCRYPT WITH, 138
END BACKUP, 136
EXCLUDE … FROM PUBLICATION, 140
   ALL, 140
   TABLE, 140
INCLUDE … TO PUBLICATION, 139
   ALL, 140
   TABLE, 140
SET DEFAULT CHARACTER SET, 137
SET DEFAULT SQL SECURITY, 137
SET LINGER, 137
ALTER DOMAIN, 150
   ADD CONSTRAINT CHECK, 152
   DROP DEFAULT, 152
   DROP NOT NULL, 153
   SET DEFAULT, 152
   SET NOT NULL, 152
   TO name, 152
   TYPE, 152
ALTER EXCEPTION, 285
ALTER EXTERNAL CONNECTIONS POOL CLEAR
   ALL, 702
ALTER EXTERNAL CONNECTIONS POOL CLEAR
   OLDEST, 702
ALTER EXTERNAL CONNECTIONS POOL SET
   LIFETIME, 701
ALTER EXTERNAL CONNECTIONS POOL SET
   SIZE, 701
ALTER EXTERNAL FUNCTION, 270
ALTER FUNCTION, 245
ALTER GENERATOR, 277
   INCREMENT BY, 278
   RESTART, 278
ALTER INDEX, 191
   ACTIVE, 192
   INACTIVE, 192
ALTER MAPPING, 687
ALTER PACKAGE, 254
ALTER PROCEDURE, 233
ALTER ROLE, 664
   RDB$ADMIN, 665
   SET SYSTEM PRIVILEGES TO, 665
ALTER SEQUENCE, 277
   INCREMENT BY, 278
   RESTART, 278
   RESTART WITH, 278
ALTER SESSION RESET, 705
ALTER TABLE, 175
   ADD, 180
   ADD CONSTRAINT, 180
   ALTER [COLUMN], 181
   COMPUTED BY, 185
   DROP DEFAULT, 183
   DROP IDENTITY, 185
   DROP NOT NULL, 184
   GENERATED ALWAYS AS, 185
   POSITION, 182
   RESTART, 184
   RESTART WITH, 184
   SET DEFAULT, 183
   SET GENERATED, 184
   SET INCREMENT, 185
   SET NOT NULL, 183
   TO, 182
   TYPE, 182
ALTER SQL SECURITY, 186
ALTER TRIGGER, 221
   POSITION, 222
ALTER USER, 647
   ACTIVE, 648
   GRANT ADMIN ROLE, 648
   INACTIVE, 648
PASSWORD, 648
REVOKE ADMIN ROLE, 649
TAGS, 648
USING PLUGIN, 649
ALTER VIEW, 201
ANY, 125
AT, 100

B
BEGIN, 421
BETWEEN, 107
BREAK, 426

C
CASE, 101
CLOSE, 454
COMMENT ON, 294
COMMIT, 630
RETAIN, 630
CONTAINING, 111
CONTINUE, 428
CREATE COLLATION, 288
CREATE DATABASE, 126
DEFAULT CHARACTER SET, 130
DIFFERENCE FILE, 131
LENGTH, 130
PAGE_SIZE, 130
PASSWORD, 129
ROLE, 129
SET NAMES, 130
STARTING AT, 130
USER, 129
CREATE DOMAIN, 145
CHECK, 148
COLLATE, 149
DEFAULT, 148
NOT NULL, 148
VALUE, 148
CREATE EXCEPTION, 283
CREATE FUNCTION, 237
DETERMINISTIC, 240
SQL SECURITY, 241
CREATE GENERATOR, 275
INCREMENT BY, 276
STARTING WITH, 276
CREATE GLOBAL TEMPORARY TABLE, 171
ON COMMIT DELETE ROWS, 171
ON COMMIT PRESERVE ROWS, 171
CREATE INDEX, 188
ASCENDING, 189
COMPUTED BY, 189
DESCENDING, 189
UNIQUE, 189
CREATE MAPPING, 683
CREATE OR ALTER EXCEPTION, 285
CREATE OR ALTER FUNCTION, 247
CREATE OR ALTER GENERATOR, 279
CREATE OR ALTER MAPPING, 688
CREATE OR ALTER PACKAGE, 255
CREATE OR ALTER PROCEDURE, 234
CREATE OR ALTER SEQUENCE, 279
CREATE OR ALTER TRIGGER, 224
CREATE OR ALTER USER, 650
CREATE OR ALTER VIEW, 203
CREATE PACKAGE, 249
SQL SECURITY, 252
CREATE PACKAGE BODY, 259
CREATE PROCEDURE, 227
SQL SECURITY, 230
CREATE ROLE, 661
SET SYSTEM PRIVILEGES TO, 662
CREATE SEQUENCE, 275
INCREMENT BY, 276
STARTING WITH, 276
CREATE SHADOW, 142
AUTO, 143
CONDITIONAL, 143
MANUAL, 143
CREATE TABLE, 155
CHECK, 166
COMPUTED BY, 161
CONSTRAINT, 163
DEFAULT, 159
DISABLE PUBLICATION, 167
ENABLE PUBLICATION, 167
EXTERNAL FILE, 172
FOREIGN KEY, 164
GENERATED ALWAYS AS, 161
IDENTITY, 160
GENERATED ALWAYS, 160
GENERATED BY DEFAULT, 160
INCREMENT BY, 161
START WITH, 161
NOT NULL, 159
PRIMARY KEY, 163
SQL SECURITY, 166
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIQUE, 163</td>
<td></td>
</tr>
<tr>
<td>CREATE TRIGGER, 206</td>
<td></td>
</tr>
<tr>
<td>ACTIVE, 209</td>
<td></td>
</tr>
<tr>
<td>AFTER, 210</td>
<td></td>
</tr>
<tr>
<td>BEFORE, 210</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL, 210</td>
<td></td>
</tr>
<tr>
<td>INACTIVE, 209</td>
<td></td>
</tr>
<tr>
<td>POSITION, 210</td>
<td></td>
</tr>
<tr>
<td>SQL SECURITY, 208</td>
<td></td>
</tr>
<tr>
<td>CREATE USER, 644</td>
<td></td>
</tr>
<tr>
<td>ACTIVE, 646</td>
<td></td>
</tr>
<tr>
<td>GRANT ADMIN ROLE, 646</td>
<td></td>
</tr>
<tr>
<td>INACTIVE, 646</td>
<td></td>
</tr>
<tr>
<td>PASSWORD, 645</td>
<td></td>
</tr>
<tr>
<td>TAGS, 646</td>
<td></td>
</tr>
<tr>
<td>USING PLUGIN, 646</td>
<td></td>
</tr>
<tr>
<td>CREATE VIEW, 196</td>
<td></td>
</tr>
<tr>
<td>WITH CHECK OPTIONS, 198</td>
<td></td>
</tr>
<tr>
<td>DECLARE CURSOR, 413</td>
<td></td>
</tr>
<tr>
<td>NO SCROLL, 414</td>
<td></td>
</tr>
<tr>
<td>SCROLL, 414</td>
<td></td>
</tr>
<tr>
<td>DECLARE EXTERNAL FUNCTION, 266</td>
<td></td>
</tr>
<tr>
<td>DECLARE FILTER, 272</td>
<td></td>
</tr>
<tr>
<td>DECLARE FUNCTION, 418</td>
<td></td>
</tr>
<tr>
<td>DECLARE PROCEDURE, 416</td>
<td></td>
</tr>
<tr>
<td>DECLARE VARIABLE, 410</td>
<td></td>
</tr>
<tr>
<td>DELETE, 382</td>
<td></td>
</tr>
<tr>
<td>ORDER BY, 384</td>
<td></td>
</tr>
<tr>
<td>PLAN, 384</td>
<td></td>
</tr>
<tr>
<td>RETURNING, 385</td>
<td></td>
</tr>
<tr>
<td>ROWS, 384</td>
<td></td>
</tr>
<tr>
<td>WHERE, 383</td>
<td></td>
</tr>
<tr>
<td>DROP COLLATION, 292</td>
<td></td>
</tr>
<tr>
<td>DROP DATABASE, 141</td>
<td></td>
</tr>
<tr>
<td>DROP DOMAIN, 154</td>
<td></td>
</tr>
<tr>
<td>DROP EXCEPTION, 286</td>
<td></td>
</tr>
<tr>
<td>DROP EXTERNAL FUNCTION, 271</td>
<td></td>
</tr>
<tr>
<td>DROP FILTER, 274</td>
<td></td>
</tr>
<tr>
<td>DROP FUNCTION, 248</td>
<td></td>
</tr>
<tr>
<td>DROP GENERATOR, 280</td>
<td></td>
</tr>
<tr>
<td>DROP INDEX, 193</td>
<td></td>
</tr>
<tr>
<td>DROP MAPPING, 688</td>
<td></td>
</tr>
<tr>
<td>DROP PACKAGE, 257</td>
<td></td>
</tr>
<tr>
<td>DROP PACKAGE BODY, 263</td>
<td></td>
</tr>
<tr>
<td>DROP PROCEDURE, 235</td>
<td></td>
</tr>
<tr>
<td>DROP ROLE, 665</td>
<td></td>
</tr>
<tr>
<td>DROP SEQUENCE, 280</td>
<td></td>
</tr>
<tr>
<td>DROP SHADOW, 144</td>
<td></td>
</tr>
<tr>
<td>DELETE FILE, 144</td>
<td></td>
</tr>
<tr>
<td>PRESERVE FILE, 144</td>
<td></td>
</tr>
<tr>
<td>DROP TABLE, 186</td>
<td></td>
</tr>
<tr>
<td>DROP TRIGGER, 225</td>
<td></td>
</tr>
<tr>
<td>DROP USER, 651</td>
<td></td>
</tr>
<tr>
<td>USING PLUGIN, 652</td>
<td></td>
</tr>
<tr>
<td>DROP VIEW, 204</td>
<td></td>
</tr>
<tr>
<td>END, 421</td>
<td></td>
</tr>
<tr>
<td>EXCEPTION, 458</td>
<td></td>
</tr>
<tr>
<td>USING, 459</td>
<td></td>
</tr>
<tr>
<td>EXECUTE BLOCK, 393</td>
<td></td>
</tr>
<tr>
<td>EXECUTE PROCEDURE, 391</td>
<td></td>
</tr>
<tr>
<td>EXECUTE STATEMENT, 432</td>
<td></td>
</tr>
<tr>
<td>ON EXTERNAL, 436</td>
<td></td>
</tr>
<tr>
<td>AS USER, 439</td>
<td></td>
</tr>
<tr>
<td>PASSWORD, 439</td>
<td></td>
</tr>
<tr>
<td>ROLE, 439</td>
<td></td>
</tr>
<tr>
<td>WITH AUTONOMOUS TRANSACTION, 436</td>
<td></td>
</tr>
<tr>
<td>WITH CALLER PRIVILEGES, 436</td>
<td></td>
</tr>
<tr>
<td>WITH COMMON TRANSACTION, 436</td>
<td></td>
</tr>
<tr>
<td>EXISTS, 120</td>
<td></td>
</tr>
<tr>
<td>EXIT, 430</td>
<td></td>
</tr>
<tr>
<td>FETCH, 449</td>
<td></td>
</tr>
<tr>
<td>FOR EXECUTE STATEMENT, 445</td>
<td></td>
</tr>
<tr>
<td>FOR SELECT, 440</td>
<td></td>
</tr>
<tr>
<td>AS CURSOR, 441</td>
<td></td>
</tr>
<tr>
<td>INTO, 441</td>
<td></td>
</tr>
<tr>
<td>GRANT, 666</td>
<td></td>
</tr>
<tr>
<td>GRANTED BY, 670</td>
<td></td>
</tr>
<tr>
<td>WITH ADMIN OPTION, 676</td>
<td></td>
</tr>
<tr>
<td>WITH GRANT OPTION, 670</td>
<td></td>
</tr>
<tr>
<td>IF, 423</td>
<td></td>
</tr>
<tr>
<td>ELSE, 423</td>
<td></td>
</tr>
<tr>
<td>THEN, 423</td>
<td></td>
</tr>
<tr>
<td>IN, 121</td>
<td></td>
</tr>
<tr>
<td>IN AUTONOMOUS TRANSACTION, 454</td>
<td></td>
</tr>
<tr>
<td>INSERT, 366</td>
<td></td>
</tr>
<tr>
<td>DEFAULT VALUES, 370</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>OVERRIDING</td>
<td>370</td>
</tr>
<tr>
<td>OVERRIDING SYSTEM VALUE</td>
<td>370</td>
</tr>
<tr>
<td>OVERRIDING USER VALUE</td>
<td>371</td>
</tr>
<tr>
<td>RETURNING</td>
<td>371</td>
</tr>
<tr>
<td>SELECT</td>
<td>369</td>
</tr>
<tr>
<td>VALUES</td>
<td>368</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>368</td>
</tr>
<tr>
<td>IS</td>
<td>118</td>
</tr>
<tr>
<td>IS FALSE</td>
<td>118</td>
</tr>
<tr>
<td>IS NULL</td>
<td>119</td>
</tr>
<tr>
<td>IS TRUE</td>
<td>118</td>
</tr>
<tr>
<td>IS UNKNOWN</td>
<td>118</td>
</tr>
<tr>
<td>IS DISTINCT FROM</td>
<td>117</td>
</tr>
<tr>
<td>LATERAL</td>
<td>312</td>
</tr>
<tr>
<td>LEAVE</td>
<td>427</td>
</tr>
<tr>
<td>LIKE</td>
<td>108</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>108</td>
</tr>
<tr>
<td>MERGE</td>
<td>386</td>
</tr>
<tr>
<td>RETURNING</td>
<td>390</td>
</tr>
<tr>
<td>NEXT VALUE FOR</td>
<td>100</td>
</tr>
<tr>
<td>OPEN</td>
<td>446</td>
</tr>
<tr>
<td>OVER</td>
<td>578</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>582</td>
</tr>
<tr>
<td>PARTITION BY</td>
<td>581</td>
</tr>
<tr>
<td>RANGE</td>
<td>584</td>
</tr>
<tr>
<td>ROWS</td>
<td>584</td>
</tr>
<tr>
<td>POST_EVENT</td>
<td>456</td>
</tr>
<tr>
<td>RECREATE TABLE</td>
<td>187</td>
</tr>
<tr>
<td>RECREATE TRIGGER</td>
<td>226</td>
</tr>
<tr>
<td>RECREATE USER</td>
<td>653</td>
</tr>
<tr>
<td>RECREATE VIEW</td>
<td>204</td>
</tr>
<tr>
<td>RELEASE SAVEPOINT</td>
<td>634</td>
</tr>
<tr>
<td>RETURN</td>
<td>457</td>
</tr>
<tr>
<td>REVOKE</td>
<td>677</td>
</tr>
<tr>
<td>ADMIN OPTION FOR</td>
<td>681</td>
</tr>
<tr>
<td>ALL ON ALL</td>
<td>682</td>
</tr>
<tr>
<td>GRANT OPTION FOR</td>
<td>680</td>
</tr>
<tr>
<td>GRANTED BY</td>
<td>681</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>631</td>
</tr>
<tr>
<td>RETAIN</td>
<td>632</td>
</tr>
<tr>
<td>ROLLBACK TO SAVEPOINT</td>
<td>632</td>
</tr>
<tr>
<td>SAVEPOINT</td>
<td>633</td>
</tr>
<tr>
<td>SELECT</td>
<td>297</td>
</tr>
<tr>
<td>FETCH</td>
<td>355</td>
</tr>
<tr>
<td>FIRST, SKIP</td>
<td>298</td>
</tr>
<tr>
<td>FOR UPDATE</td>
<td>356</td>
</tr>
<tr>
<td>OF</td>
<td>357</td>
</tr>
<tr>
<td>FROM</td>
<td>304</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>330</td>
</tr>
<tr>
<td>HAVING</td>
<td>334</td>
</tr>
<tr>
<td>INTO</td>
<td>360</td>
</tr>
<tr>
<td>JOIN</td>
<td>314</td>
</tr>
<tr>
<td>CROSS JOIN</td>
<td>324</td>
</tr>
<tr>
<td>FULL JOIN</td>
<td>317</td>
</tr>
<tr>
<td>INNER</td>
<td>317</td>
</tr>
<tr>
<td>LEFT JOIN</td>
<td>317</td>
</tr>
<tr>
<td>NATURAL</td>
<td>322</td>
</tr>
<tr>
<td>ON</td>
<td>319</td>
</tr>
<tr>
<td>OUTER</td>
<td>316</td>
</tr>
<tr>
<td>RIGHT JOIN</td>
<td>317</td>
</tr>
<tr>
<td>USING</td>
<td>320</td>
</tr>
<tr>
<td>OFFSET</td>
<td>355</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>348</td>
</tr>
<tr>
<td>ASC</td>
<td>349</td>
</tr>
<tr>
<td>COLLATE</td>
<td>350</td>
</tr>
<tr>
<td>DESC</td>
<td>349</td>
</tr>
<tr>
<td>NULLS FIRST</td>
<td>350</td>
</tr>
<tr>
<td>NULLS LAST</td>
<td>350</td>
</tr>
<tr>
<td>PLAN</td>
<td>337</td>
</tr>
<tr>
<td>ROWS</td>
<td>352</td>
</tr>
<tr>
<td>UNION</td>
<td>346</td>
</tr>
<tr>
<td>ALL</td>
<td>346</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>346</td>
</tr>
</tbody>
</table>
WHERE, 327
WINDOW, 336
WITH, 362
WITH LOCK, 357
SET BIND, 691
SET DECFLOAT
    ROUND, 694
    TRAPS, 696
SET GENERATOR, 282
SET ROLE, 702
SET SESSION IDLE TIMEOUT, 700
SET STATEMENT TIMEOUT, 698
SET STATISTICS, 194
SET TIME ZONE, 704
SET TRANSACTION, 619
    AUTO COMMIT, 627
    IGNORE LIMBO, 627
    ISOLATION LEVEL, 622
    NO AUTO UNDO, 627
    NO WAIT, 622
    READ COMMITTED, 624
        NO RECORD_VERSION, 625
        READ CONSISTENCY, 625
        RECORD_VERSION, 625
    READ ONLY, 621
    READ WRITE, 621
    RESERVING, 628
    SNAPSHOT, 623
        AT NUMBER, 623
    SNAPSHOT TABLE STABILITY, 624
    WAIT, 622
SET TRUSTED ROLE, 703
SIMILAR TO, 112
SINGULAR, 123
SOME, 125
SQL SECURITY, 655
SQL диалект, 30
STARTING WITH, 110
SUSPEND, 430
SYSDBA, 637

SET, 375
    DEFAULT, 375
WHERE, 376
UPDATE OR INSERT, 379
    DEFAULT, 381
    MATCHING, 380
    RETURNING, 381

W
WHEN ... DO, 461
    ANY, 462
WHILE, 425

A
Агрегатные функции, 555
    FILTER, 555
Аналитические функции, 578

B
Выражение, 90

I
Идентификатор, 32

K
Комментарии, 34
Контекстные переменные, 603
    CURRENT_CONNECTION, 603
    CURRENT_DATE, 603
    CURRENT_ROLE, 604
    CURRENT_TIME, 605
    CURRENT_TIMESTAMP, 605
    CURRENT_TRANSACTION, 606
    CURRENT_USER, 607
    DELETING, 607
    GDSCODE, 608
    INSERTING, 609
    LOCALTIME, 610
    LOCALTIMESTAMP, 610
    NEW, 611
    OLD, 612
    RESETTING, 613
    ROW_COUNT, 613
    SQLCODE, 615
    SQLSTATE, 616
    UPDATING, 616
    USER, 617

U
UPDATE, 373
    INTO, 378
    ORDER BY, 377
    PLAN, 376
    RETURNING, 378
    ROWS, 377
Л
Литерал, 92

О
Общие табличные выражения, 362
Оконные функции, 578

П
План запроса, 337
Предикат, 106
Производная таблица, 308

Т
Типы данных, 36
  BIGINT, 39
  BINARY, 67
  BLOB, 72
  BOOLEAN, 69
  CHAR, 67
  CHAR VARYING, 69
  CHARACTER, 67
  CHARACTER VARYING, 69
  DATE, 53
  DECIMAL, 50
  DOUBLE PRECISION, 43
  FLOAT, 42
  INT128, 40
  INTEGER, 39
  LONG FLOAT, 43
  NATIONAL CHAR, 69
  NATIONAL CHARACTER, 69
  NCHAR, 69
  NUMERIC, 49
  REAL, 43
  SMALLINT, 39
  TIME, 54
    WITH TIME ZONE, 54
    WITHOUT TIME ZONE, 54
  TIMESTAMP, 54
    WITH TIME ZONE, 54
    WITHOUT TIME ZONE, 54
  VARBINARY, 68
  VARCHAR, 69

Ф
Функция, 466
  ABS(), 472
  ACOS(), 473
  ACOSH(), 473
  ASCII_CHAR(), 488
  ASCII_VAL(), 488
  ASIN(), 474
  ASINH(), 474
  ATAN(), 475
  ATAN2(), 475
  ATANH(), 476
  AVG(), 556
  BASE64_DECODE(), 489
  BASE64_ENCODE(), 490
  BIN_AND(), 536
  BIN_NOT(), 537
  BIN_OR(), 537
  BIN_SHL(), 538
  BIN_SHR(), 538
  BIN_XOR(), 539
  BIT_LENGTH(), 491
  BLOB_APPEND(), 517
  CAST(), 533
  CEIL(), 477
  CHAR_LENGTH(), 492
  CHAR_TO_UUID(), 539
  COALESCE(), 543
  COMPARE_DECFLOAT(), 520
  CORR(), 563
  COS(), 477
  COSH(), 478
  COT(), 478
  COUNT(), 557
  COVAR_POP(), 564
  COVAR_SAMP(), 564
  CRYPT_HASH(), 524
  CUME_DIST(), 591
  DATEADD(), 511
  DATEDIFF(), 512
  DECODE(), 544
  DECRYPT(), 524
  DENSE_RANK(), 588
  ENCRYPT(), 526
  EXP(), 479
  EXTRACT(), 514
  FIRST_DAY(), 515
  FIRST_VALUE(), 595
  FLOOR(), 479
  GEN_ID(), 542
  GEN_UUID(), 540
<table>
<thead>
<tr>
<th>Алфавитный указатель</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH(), 492</td>
</tr>
<tr>
<td>HEX_DECODE(), 493</td>
</tr>
<tr>
<td>HEX_ENCODE(), 494</td>
</tr>
<tr>
<td>IIF(), 545</td>
</tr>
<tr>
<td>LAG(), 596</td>
</tr>
<tr>
<td>LAST_DAY(), 516</td>
</tr>
<tr>
<td>LAST_VALUE(), 597</td>
</tr>
<tr>
<td>LEAD(), 598</td>
</tr>
<tr>
<td>LEFT(), 495</td>
</tr>
<tr>
<td>LIST(), 558</td>
</tr>
<tr>
<td>LN(), 479</td>
</tr>
<tr>
<td>LOG(), 480</td>
</tr>
<tr>
<td>LOG10(), 481</td>
</tr>
<tr>
<td>LOWER(), 496</td>
</tr>
<tr>
<td>LPAD(), 497</td>
</tr>
<tr>
<td>MAKE_DBKEY(), 548</td>
</tr>
<tr>
<td>MAX(), 560</td>
</tr>
<tr>
<td>MAXVALUE(), 546</td>
</tr>
<tr>
<td>MIN(), 561</td>
</tr>
<tr>
<td>MINVALUE(), 547</td>
</tr>
<tr>
<td>MOD(), 481</td>
</tr>
<tr>
<td>NORMALIZE_DECFLOAT(), 521</td>
</tr>
<tr>
<td>NTH_VALUE(), 599</td>
</tr>
<tr>
<td>NTILE(), 592</td>
</tr>
<tr>
<td>NULLIF(), 547</td>
</tr>
<tr>
<td>OCTET_LENGTH(), 498</td>
</tr>
<tr>
<td>OVERLAY(), 499</td>
</tr>
<tr>
<td>PERCENT_RANK(), 590</td>
</tr>
<tr>
<td>PI(), 482</td>
</tr>
<tr>
<td>POSITION(), 501</td>
</tr>
<tr>
<td>POWER(), 482</td>
</tr>
<tr>
<td>QUANTIZE(), 521</td>
</tr>
<tr>
<td>RAND(), 482</td>
</tr>
<tr>
<td>RANK(), 589</td>
</tr>
<tr>
<td>RDB$ERROR(), 550</td>
</tr>
<tr>
<td>RDB$GET_CONTEXT(), 466</td>
</tr>
<tr>
<td>RDB$GET_TRANSACTION_CN(), 552</td>
</tr>
<tr>
<td>RDB$ROLE_IN_USE(), 553</td>
</tr>
<tr>
<td>RDB$SET_CONTEXT(), 470</td>
</tr>
<tr>
<td>RDB$SYSTEM_PRIVILEGE(), 554</td>
</tr>
<tr>
<td>REGR_AVGX(), 570</td>
</tr>
<tr>
<td>REGR_AVGY(), 570</td>
</tr>
<tr>
<td>REGR_COUNT(), 571</td>
</tr>
<tr>
<td>REGR_INTERCEPT(), 572</td>
</tr>
<tr>
<td>REGR_R2(), 573</td>
</tr>
<tr>
<td>REGR_SLOPE(), 574</td>
</tr>
<tr>
<td>REGR_SXX(), 575</td>
</tr>
<tr>
<td>REGR_SYX(), 576</td>
</tr>
<tr>
<td>REPLACE(), 502</td>
</tr>
<tr>
<td>REVERSE(), 503</td>
</tr>
<tr>
<td>RIGHT(), 504</td>
</tr>
<tr>
<td>ROUND(), 483</td>
</tr>
<tr>
<td>ROW_NUMBER(), 593</td>
</tr>
<tr>
<td>RPAD(), 505</td>
</tr>
<tr>
<td>RSA_DECRYPT(), 530</td>
</tr>
<tr>
<td>RSA_ENCRYPT(), 529</td>
</tr>
<tr>
<td>RSA_PRIVATE(), 528</td>
</tr>
<tr>
<td>RSA_PUBLIC(), 529</td>
</tr>
<tr>
<td>RSA_SIGN_HASH(), 531</td>
</tr>
<tr>
<td>RSA_VERIFY_HASH(), 532</td>
</tr>
<tr>
<td>SIGN(), 484</td>
</tr>
<tr>
<td>SIN(), 484</td>
</tr>
<tr>
<td>SINH(), 485</td>
</tr>
<tr>
<td>SQRT(), 485</td>
</tr>
<tr>
<td>STANDARD Deviation POP(), 565</td>
</tr>
<tr>
<td>STANDARD Deviation SAMP(), 566</td>
</tr>
<tr>
<td>SUBSTRING(), 506</td>
</tr>
<tr>
<td>SUM(), 561</td>
</tr>
<tr>
<td>TAN(), 485</td>
</tr>
<tr>
<td>TANH(), 486</td>
</tr>
<tr>
<td>TOTALORDER(), 523</td>
</tr>
<tr>
<td>TRIM(), 509</td>
</tr>
<tr>
<td>TRUNC(), 486</td>
</tr>
<tr>
<td>UPPER(), 510</td>
</tr>
<tr>
<td>UUID_TO_CHAR(), 541</td>
</tr>
<tr>
<td>VAR_POP(), 567</td>
</tr>
<tr>
<td>VAR_SAMP(), 568</td>
</tr>
</tbody>
</table>

870