MATERIALIZED VIEWS IN FIREBIRD

ALEXEY KOVYAZIN, FIREBIRD FOUNDATION

DENIS SIMONOV, IBSURGEON

AVAILABILITY?

- Materialized Views will be available in vanilla Firebird 6
- Already available in HQbird with Firebird 5
  - More details about HQbird https://firebirdsql.org/hqbird

Materialized Views
Materialized View (Hereinafter MV)

- A hybrid of a view (**VIEW**) and a table (**TABLE**). Stores the result of query execution.
- Read-only (**SELECT**). You cannot directly perform **INSERT**, **UPDATE**, **DELETE**
- Updated using the command **REFRESH MATERIALIZED VIEW** or **REFRESH MATERIALIZED VIEW CONCURENTLY**

These are DDL commands, so they cannot be called from within PSQL.

- For MV you can create indexes, including unique ones
- A regular view can be converted into a materialized one and vice versa
- Not automatically refreshed when the underlying tables change (similar to Oracle)

Materialized Views or Regular Views

Property	Regular VIEW	MATERIALIZED VIEW
Querying the view	Slow if the query is complex.	Fast, since the inner query does not need to be re-executed.
Tracking changes in the underlying tables	Yes. Automatic	No. To refresh the data, you need to call the command REFRESH MATERIALIZED VIEW
Modifying the underlying tables via INSERT/UPDATE/DELETE	Yes. Corresponding triggers must be written (except for the simplest cases).	No.
Indexing	No. In some cases the optimizer can use the indexes of the underlying tables.	Yes. Any field or several fields can be indexed, including creating a unique index.

How Did We Live Before?

- A regular table was created with fields identical in type and composition to the fields in the query
- In HQBIRD SERVER 5.0 a simpler option can be used: **CREATE TABLE … AS SELECT**
- A stored procedure was created that refreshed this table
- Downsides of this approach:
  - Clearing the table is expensive if a full refresh is required (DELETE)
  - You need to keep track of the table column types and their composition
  - Several metadata objects instead of one – hard to make changes to business logic
- Upsides:
  - Triggers can be written on the underlying tables that update the data in the storage table

Use Cases for MV

- Stored aggregates for closed periods
- Complex calculations on rarely changing data
- Data marts for further analysis
- Scheduled refresh via CRON

Syntax

```
CREATE [OR ALTER] MATERIALIZED VIEW <view_name> [(columns_names)]
  AS <query>
  [WITH [NO] DATA]

ALTER MATERIALIZED VIEW <view_name> [(columns_names)]
  AS <query>
  [WITH [NO] DATA]

RECREATE MATERIALIZED VIEW <view_name> [(columns_names)]
  AS <query>
  [WITH [NO] DATA]

DROP VIEW <view_name>
```

View <-> Materialized View

```
ALTER MATERIALIZED VIEW <view_name> [(columns_names)]
  AS <query>
  TO NOT MATERIALIZED

ALTER VIEW <view_name> [(columns_names)]
  AS <query>
  TO MATERIALIZED
```

Refreshing Data in MV

```
REFRESH MATERIALIZED VIEW <view_name>
[CONCURRENTLY | DROP DATA] [CASCADE]
```

Refresh Materialized View <view_name>

- Full refresh of MV in **exclusive** mode

1. All indexes are deactivated
2. All data is deleted (similar to **TRUNCATE TABLE**)
3. The table is populated with new data
4. Indexes are rebuilt

- This is the fastest way to refresh a large amount of data in MV

Refresh Materialized View <view_name> Concurrently

- Refreshing MV while allowing queries against MV to continue running during the refresh
- Requires a **UNIQUE INDEX** on the MV
- Requires the view source code (**RDB$VIEW_SOURCE IS NOT NULL**)
- Performs something like a MERGE (new rows are inserted, existing ones are updated, and non-existing ones are deleted)

- Useful when you need to refresh a small number of rows in MV and MV is being actively used (SELECT queries)

Refresh Materialized View <view_name> Drop Data

- Deletes the materialized view data and makes its indexes inactive
- Requires exclusive mode

- Useful when you need to free up database space occupied by the data and indexes of the MV

Refresh Materialized View … Cascade

- Refreshes MV and the MV that the current view depends on
- Can be used together with CONCURRENTLY or DROP DATA
  - CONCURRENTLY CASCADE – requires a unique index on all MV that the current view depends on
  - DROP DATA CASCADE – deletes the data of the current view and all views that the current view depends on

Materialized View in GBAK

- The data of MV is not included in the backup
- During restore: the MV object is restored, and at the end of the restore process, **REFRESH CASCADE** is run for all MV in the correct order
- The switch **-NO_MATVIEWS** or **-NO_M** allows you to skip refreshing MV during restore. In Services, the equivalent parameter is **isc_spb_res_no_matviews**
- If you need to restore into Firebird that does not support MV, you must first convert the MV in the database into regular views using

ALTER MATERIALIZED VIEW ... TO NOT MATERIALIZED

Hands-On. Product Stock Balance

The database stores 10 years of product sales and receipts
The example is synthetic
PRODUCT – 10000 rows
INVOICE – 1100000 rows
INVOICE_LINE – 11545586 rows
PRODUCT_STORE – 4810985 rows

SQL query calculating current stock balance

```
WITH
  T AS (
    SELECT S.PRODUCT_ID, SUM(S.QUANTITY) AS QUANTITY
    FROM PRODUCT_STORE S
    GROUP BY 1
    UNION ALL
    SELECT L.PRODUCT_ID, -SUM(L.QUANTITY) AS QUANTITY
    FROM INVOICE_LINE L
    GROUP BY 1
  ),
  REMAIN_PRODUCTS AS (
    SELECT PRODUCT_ID, SUM(QUANTITY) AS QUANTITY
    FROM T
    GROUP BY 1
  )
SELECT
  REMAIN_PRODUCTS.PRODUCT_ID,
  PRODUCT.NAME AS PRODUCT_NAME,
  REMAIN_PRODUCTS.QUANTITY
FROM
  REMAIN_PRODUCTS
  JOIN PRODUCT ON PRODUCT.PRODUCT_ID = REMAIN_PRODUCTS.PRODUCT_ID
ORDER BY PRODUCT.NAME;
```

SQL query calculating current stock balance

```
Elapsed time = 29.398 sec
Buffers = 131072
Reads = 628296
Writes = 1
Fetches = 49079938

Per table statistics:
-------------------------------+---------+---------+---------+---------+---------+
 Table name                    | Natural | Index   | Insert  | Update  | Delete  |
-------------------------------+---------+---------+---------+---------+---------+
PRODUCT                        |         |    10000|         |         |         |
INVOICE_LINE                   |         | 11545586|         |         |         |
PRODUCT_STORE                  |         |  4810985|         |         |         |
-------------------------------+---------+---------+---------+---------+---------+
```

Hands-On. Product Stock Balance. How to Improve?

We will calculate the change in stock balance only for the current quarter
We store the balance at the end of the previous quarter in the MV
We sum the stored balance and its change
We display the result to the user

SQL to Calculate the Balance at the End of the Previous Quarter

```
WITH
  T AS (
    SELECT S.PRODUCT_ID, SUM(QUANTITY) AS QUANTITY
    FROM
      PRODUCT_STORE S
    WHERE S.DATE_OF_RECEIPT < FIRST_DAY(OF QUARTER FROM CURRENT_DATE)
    GROUP BY 1
    UNION ALL
    SELECT L.PRODUCT_ID, -SUM(L.QUANTITY) AS QUANTITY
    FROM
      INVOICE_LINE L
      JOIN INVOICE ON INVOICE.INVOICE_ID = L.INVOICE_ID
    WHERE INVOICE.CREATE_DATE < FIRST_DAY(OF QUARTER FROM CURRENT_DATE)
    GROUP BY 1
  )
SELECT
  T.PRODUCT_ID, SUM(T.QUANTITY) AS QUANTITY
FROM T
GROUP BY 1;
```

SQL to Calculate the Balance at the End of the Previous Quarter
execution statistics

```
Elapsed time = 23.166 sec
Buffers = 131072
Reads = 222521
Writes = 1
Fetches = 31516649

Per table statistics:
```

Table name	Natural	Index	Insert	Update	Delete
INVOICE		1095272			
INVOICE_LINE		11495717			
PRODUCT_STORE		4794069			

MV for storing the prior quarter-end balance

```
CREATE MATERIALIZED VIEW V_REMAIN_PRODUCTS
AS
WITH
  T AS (
    SELECT S.PRODUCT_ID, SUM(QUANTITY) AS QUANTITY
    FROM PRODUCT_STORE S
    WHERE S.DATE_OF_RECEIPT < FIRST_DAY(OF QUARTER FROM CURRENT_DATE)
    GROUP BY 1
    UNION ALL
    SELECT L.PRODUCT_ID, -SUM(L.QUANTITY) AS QUANTITY
    FROM
      INVOICE_LINE L
      JOIN INVOICE ON INVOICE.INVOICE_ID = L.INVOICE_ID
    WHERE INVOICE.CREATE_DATE < FIRST_DAY(OF QUARTER FROM CURRENT_DATE)
    GROUP BY 1
  )
SELECT
  T.PRODUCT_ID, SUM(T.QUANTITY) AS QUANTITY
FROM T
GROUP BY 1
WITH DATA -- created with data; if WITH NO DATA (the default) is used, it is created without data
;
```

Creating MV. Statistics

Creation without data:

```
Elapsed time = 0.009 sec
Buffers = 131072
Reads = 0
Writes = 39
Fetches = 1718
```

Creation with data:

```
Elapsed time = 22.927 sec
Buffers = 131072
Reads = 222526
Writes = 161
Fetches = 31519545
```

```
-------------------------------+---------+---------+---------+---------+---------+
 Table name                    | Natural | Index   | Insert  | Update  | Delete  |
-------------------------------+---------+---------+---------+---------+---------+
INVOICE                        |         |  1095272|         |         |         |
INVOICE_LINE                   |         | 11495717|         |         |         |
PRODUCT_STORE                  |         |  4794069|         |         |         |
V_REMAIN_PRODUCTS              |         |         |   10000|         |         |
-------------------------------+---------+---------+---------+---------+---------+
```

MV for storing the prior quarter-end balance Creating an Index

```
CREATE UNIQUE INDEX IDX_V_REMAIN_PRODUCTS_PRODUCT_ID
ON V_REMAIN_PRODUCTS(PRODUCT_ID);

Elapsed time = 0.013 sec
Buffers = 131072
Reads = 2
Writes = 31
Fetches = 10542
```

SQL Query Calculating Current Stock Balance

```
WITH
  T AS (
    SELECT S.PRODUCT_ID, SUM(S.QUANTITY) AS QUANTITY
    FROM PRODUCT_STORE S
    WHERE S.DATE_OF_RECEIPT >= FIRST_DAY(OF QUARTER FROM CURRENT_DATE)
    GROUP BY 1
    UNION ALL
    SELECT L.PRODUCT_ID, -SUM(L.QUANTITY) AS QUANTITY
    FROM
      INVOICE_LINE L
      JOIN INVOICE ON INVOICE.INVOICE_ID = L.INVOICE_ID
    WHERE INVOICE.CREATE_DATE >= FIRST_DAY(OF QUARTER FROM CURRENT_DATE)
    GROUP BY 1
  ),
  REMAIN_PRODUCTS AS (
    SELECT PRODUCT_ID, SUM(QUANTITY) AS QUANTITY
    FROM T
    GROUP BY 1
    UNION ALL
    SELECT PRODUCT_ID, QUANTITY
    FROM V_REMAIN_PRODUCTS
  ),
  T2 AS (SELECT PRODUCT_ID, SUM(QUANTITY) AS QUANTITY FROM REMAIN_PRODUCTS GROUP BY 1)
SELECT
  T2.PRODUCT_ID, PRODUCT.NAME AS PRODUCT_NAME, T2.QUANTITY
FROM T2 JOIN PRODUCT ON PRODUCT.PRODUCT_ID = T2.PRODUCT_ID
ORDER BY PRODUCT.NAME;
```

SQL Query Calculating Current Stock Balance Execution Statistics

```
Elapsed time = 0.308 sec
Buffers = 131072
Reads = 0
Writes = 0
Fetches = 167190

Per table statistics:
```

Table name	Natural	Index	Insert	Update	Delete
PRODUCT		10000			
INVOICE		4728			
INVOICE_LINE		49869			
PRODUCT_STORE		16916			
V_REMAIN_PRODUCTS	10000				

Let's compare the performance: 29.398 sec vs 0.308 sec. That is ~ 95 times faster.

Exclusive Refresh of MV

```
-- Updated 1168 rows.
UPDATE INVOICE_LINE
SET QUANTITY = QUANTITY + 1
WHERE PRODUCT_ID = 5000;

REFRESH MATERIALIZED VIEW V_REMAIN_PRODUCTS;

Elapsed time = 23.120 sec
Buffers = 131072
Reads = 217749
Writes = 2529
Fetches = 31549835

Per table statistics:
--------------------------------+---------+---------+---------+---------+---------+
 Table name                     | Natural | Index   | Insert  | Update  | Delete  |
--------------------------------+---------+---------+---------+---------+---------+
INVOICE                         |         |  1095272|         |         |         |
INVOICE_LINE                    |         | 11495717|         |         |         |
PRODUCT_STORE                   |         |  4794069|         |         |         |
V_REMAIN_PRODUCTS               |         |         |    10000|         |         |
--------------------------------+---------+---------+---------+---------+---------+
```

Concurrent Refresh of MV

```
-- Updated 1168 rows.
UPDATE INVOICE_LINE
SET QUANTITY = QUANTITY - 1
WHERE PRODUCT_ID = 5000;

REFRESH MATERIALIZED VIEW V_REMAIN_PRODUCTS CONCURRENTLY;

Elapsed time = 25.927 sec
Buffers = 131072
Reads = 222121
Writes = 4
Fetches = 31536792

Per table statistics:
--------------------------------+---------+---------+---------+---------+---------+
 Table name                     | Natural | Index   | Insert  | Update  | Delete  |
--------------------------------+---------+---------+---------+---------+---------+
INVOICE                         |         |  1095272|         |         |         |
INVOICE_LINE                    |         | 11495717|         |         |         |
PRODUCT_STORE                   |         |  4794069|         |         |         |
V_REMAIN_PRODUCTS               |         |    10000|         |        1|         |
--------------------------------+---------+---------+---------+---------+---------+
```

Deleting Data from MV

```
MATERIALIZED VIEW V_REMAIN_PRODUCTS DROP DATA;

Elapsed time = 0.005 sec
Buffers = 131072
Reads = 0
Writes = 18
Fetches = 367
```

QUESTIONS?

AK@FIREBIRDSQL.ORG