Creating logs for data auditing

Carlos H. Cantu

www.firebaseapp.com.br
www.firebirdnews.org
Who am I?

- Maintainer of www.firebase.com.br and www.firebirdnews.org
- Author of 2 Firebird books published in Brazil
- Software developer for about 30 years
- Organizer of the Firebird Developers Day conference
- Firebird consultant
Why logging?

- To know what, when, who and where information was inserted, deleted or altered.
- Technical information: transaction number, isolation, protocol, etc..
- Avoid (once for all) allegations like:
  - The record disappeared!
  - I didn’t change anything!
Proposed solution

• Log routines implemented using native features of Firebird >= 2.1 (i.e.: PSQL, triggers and procedures).

• Two log tables used:
  – Operations executed
  – Data associated with those operations

• The creation and maintenance of the log triggers will be entirely done by a single store procedure.
Log operations table structure

CREATE TABLE LOG_OPERATIONS (  
  IDLOGOPER       BIGINT NOT NULL,  -- Primary key  
  TABLE_NAME      VARCHAR(31) NOT NULL COLLATE WIN_PTBR,  
  OPERATION       CHAR(1) NOT NULL,  
  USER_NAME       VARCHAR(64) COLLATE WIN_PTBR,  
  SYSTIME         TIMESTAMP DEFAULT CURRENT_TIMESTAMP(0),  
  TRANSACTIONID   INTEGER,  
  CLIENT_ADDRESS  VARCHAR(255) COLLATE WIN_PTBR,  
  NETWORK_PROTOCOL VARCHAR(255) COLLATE WIN_PTBR,  
  TR_ISOLATION    VARCHAR(255) COLLATE WIN_PTBR,  
  PK1             VARCHAR(50) COLLATE WIN_PTBR,  
  PK2             VARCHAR(50) COLLATE WIN_PTBR,  
  PK3             VARCHAR(50) COLLATE WIN_PTBR  
);
### Log data table

```sql
CREATE TABLE LOG_DATA ( 
    ID BIGINT NOT NULL, -- Primary key
    IDLOGOPER BIGINT NOT NULL, -- Foreign key
    COLUMN_NAME VARCHAR(31) COLLATE WIN_PTBR,
    OLD_VALUE VARCHAR(2046) COLLATE WIN_PTBR,
    NEW_VALUE VARCHAR(2046) COLLATE WIN_PTBR,
    OLD_BLOB BLOB SUB_TYPE 0 SEGMENT SIZE 80,
    NEW_BLOB BLOB SUB_TYPE 0 SEGMENT SIZE 80
);
```
Pay attention...

- Rapid growing of the database file.
- Performance of the operations.
- Easy way to use the logged information.
- Maintenance of the log routines in the case of changes in the metadata.
- Blob columns.
- Varchar columns.
- Float or Double precision columns.
Rapid growing of the database file

• Logged data occupies space in the database.

• Tips:
  – Put the log in a separate database (speed up the production database backups, etc.).
  – Store the log database in another hard drive.
  – Purge of the old logs from time to time.
  – Transfer the old log data to archived files.
Operations performance

• In normal usage conditions, the performance degradation is not noticed by the users.

• Batch operations can show perceptible performance loss.

• Take care of combination of $FW = ON + \text{Barrier active}$ in Linux systems!
Easy way to use logged data

- The logged information are stored in “normal” tables in the database, so they can be accessed using *selects*.

- You can create user friendly GUI in your app, allowing users to make their own searches in the logged data.
Log routines maintenance

• Any change in the database table’s metadata needs the log trigger of that table to be updated.

• Updating the log trigger is quick and easy (ie: just run the procedure and it will recreate the log trigger).
Blob columns

• Blob can has “any” size.

• Null blobs occupies only a few bytes in the database page.
VARCHAR columns

- Varchar and char columns are stored RLE compressed.
- Content can vary from 1 to 32,767 (char) and 32,765 (varchar) “bytes”.
- You can set a limit (trunc) to the size of char/varchars stored in the log tables.
Float or Double Precision columns

- Take care with the precision!
- The “string” version of the values may not be exactly equal to the original value (IEEE standard inherited “problem”).
- Always when possible, prefer to use `decimal` or `numeric` (with dialect 3) to avoid inaccurate values problem.
Creation of the log triggers

• DDL (Data Definition Language) statements are not direct available inside procedures and/or triggers.

• **Solution:** Use `execute statement` to run DDL statements.

• **Warning:** There is a **64kb limit** in the source code of procedures and triggers

• Use `IS DISTINCT FROM` instead of

  ```sql
  if ((new.afield<> old.afield) or ((new.afield is null) and (old.afield is not null)) or ((new.afield is not null) and (old.afield is null)))
  ```
Storing the log in external databases

- Firebird >= 2.5 brought some enhancements to execute statement
- It allows to access external databases!
- EXECUTE STATEMENT <query_text> [(<input_parameters>)]
  [ON EXTERNAL [DATA SOURCE] <connection_string>]
  [WITH {AUTONOMOUS | COMMON} TRANSACTION]
  [AS USER <user_name>]
  [PASSWORD <password>]
  [ROLE <role_name>]
  [WITH CALLER PRIVILEGES]
  [INTO <variables>]}
Example database
Benchmarks

• Batch operations:
  100,000 inserts
  20,000 updates
  10,000 deletes

• Firebird 2.5.2 SS

• Windows 8.1 Pro 64bits

• Intel QuadCore + 16GB RAM

Obs: All operations were executed inside a single transaction.
Firebird 2.5.2 SuperServer

• No logging (log inactive):
  Prepare time = 0ms
  **Execute time = 1m 30s 954ms**
  Current memory = 1.336.460
  Max memory = 2.214.544
  Memory buffers = 75
  Reads from disk to cache = 70.670
  Writes from cache to disk = 51.483
  Fetches from cache = 1.444.617

• Log active (External log DB):
  Prepare time = 0ms
  **Execute time = 2m 45s 141ms (1,83x increase)**
  Current memory = 1.743.324
  Max memory = 2.620.112
  Memory buffers = 75
  Reads from disk to cache = 70.448
  Writes from cache to disk = 51.200
  Fetches from cache = 1.444.727
Firebird 2.5.2 SuperServer

- **log active** (internal log tables)
  - Prepare time = 0ms
  - **Execute time = 4m 57s 375ms (3.3x increase)**
  - Current memory = 2.016.788
  - Max memory = 3.191.832
  - Memory buffers = 90
  - Reads from disk to cache = 84.734
  - Writes from cache to disk = 83.495
  - Fetches from cache = 13.645.639
  - **Log DB size = ~118MB**
Possible improvements

- Allow to specify if want to log only inserts, deletes or updates (or a combination of them).

- Use internal function `rdb$get_context` to retrieve the logged user, instead of the connected user (`current_user`)

- Reduce the name of the log procedures, to save bytes in the triggers source code.
Tips

• In extreme cases, to increase log performance in batch operations, you can deactivate the indexes of the log tables before executing them, and activate them after it is finished.

• Connect to the external database using an “embedded” connection.

• Configure the log database with the “-use full” option.
Questions?

www.firebase.com.br
www.firebirdnews.org

Download the scripts at
http://www.firebase.com.br/fb/imgdocs/cantu_log_bds.zip

Thanks to all Conference sponsors: