Keeping DB metadata in sync with your application

Carlos H. Cantu
www.firebase.com.br – www.firebirdnews.org – www.dbfreemagazine.com.br

Environment

- **Several customers located in diferent places (sites), using the same application.**
- **Each customer has its own application .EXE, network, and its own Firebird server and database.**
- **Database is used in network environment (multiple simultaneous connections)**

The Problem

- Many times the database structure (metadata) needs to be modified (**upgraded**), to be compatible with a new version of the application.
- Different customers can upgrade in **different times**.
- Customers may be **using different (old) versions of the database**.
- It is better to **make metadata changes in "exclusive" connection**, to avoid "*object is in use*" or metadata cache problems…
- If metadata **update fails, database must be recovered to the old (working) version**.

Proposed solution

- Use of multiple “diff” **SQL scripts** to upgrade metadata to each of the “database versions”.
- Application should **check the actual database version** and apply (or not) any upgrades.
- Application should take care to **keep the actual database safe**, since if something goes wrong, there will be a working copy of it to be restored.

Important!

New DB version = New application executable

Database version flag

- Use a simple table inside the database with a **flag field** used to identify the currently metadata version structure.

  Ex:
  create table **CONTROL** (
  DB_VERSION integer not null,
  InMaintenance char(1));

Diff SQL scripts

- Each script will contain the **DDL/DML instructions** to upgrade the metadata to some specific version of the database.
- The update scripts can be built using specific **tools**, or can be built **manually**.

Database comparer tools: IBComparer, DBComparer, IBExpert, etc. *Obs: Do not trust 100% in the generated script! Test it before distributing!*

File (script) comparer tools: Araxis Merge, Beyond Compare.

Algorithm

- After connection, application check the database metadata version reading the *version flag field* in the control table, and compare it to its own “hardcoded” version:
  - If database version **is equal** application version, **no upgrade is needed**.
  - If database version **is lower** than application version, **upgrade is needed**.
  - If database version **is higher** than application version, warn the user and **abort the application**.

DB version is lower than App version

1. App generates backup of the database.
2. App restores the just generated backup using temporary file name.
3. App applies every upgrade script, starting with the first one after the actual database version until the last one.
4. If no exception was generated, currently database is erased and updated database is renamed to correct DB name.

PROBLEM!!!!

FB has no database **RENAME** statement ☹

Possible solutions:

1. Using operational system functions to rename the file, but it would need physical access rights to DB file ☹
2. Using UDF to rename the file (*weird!*)
3. Using external “custom” service to rename the file

DB version is lower than App version

Second approach

1. App creates a backup of the currently DB
   InMaintenance flag is set to "Y".
   *Extra option would be to **test** the backup doing a restore.*
2. App applies the update SQL scripts to currently DB
   At the end, *DB_Version field in Control table is updated*
3. If something goes wrong during *2nd pass*, old database is restored and application exits.
4. If everything was fine, *InMaintenance* flag is set to "N"

- *Auxiliar field "InMaintenance" in the Control table is set to **(Y)**es or **(N)**o to avoid other users to "use" the database while it is being updated.* ***In FB 2.0 we could put the DB is Single Shutdown mode for this.***

Extra Safety....

- If something goes wrong, we can **leave the backup file** saved in the hard drive so, in the worse option, DBA can restore, correct/fix or update it manually.

- Periodically, a scheduled process, or DBA can **delete the old backup files** to free up space in the HD.

- **Problem 1**
  While restoring a backup (because something went wrong during the update process), if someone connects to the DB before the restore is completed, DB can get corrupted if FB version is less than 2.0.

  Solution: Move to FB 2.0 – it blocks new connections to the DB while it is being restored

- **Problem 2**
  During the update process, users can connect to the DB using third party applications or tools that are not aware of the *InMaintenance* flag. This may cause problems if some of the operations need “exclusive” access to be done.

  Solution: Move to FB 2.0 and use ***Single Shutdown*** mode instead of the *InMaintenance* flag.

- How to **minimize** “Problem 2” using FB 1.x
  Answer: Make the application login as an user different from SYSDBA or DB owner for normal usage. When update is needed, put DB file in *Multi Shutdown* mode, connect with SYSDBA/Owner and apply the changes. After that, put DB back online.
  Obs: This is not 100% solution, since users can still connect to DB using third party utils, logging as SYSDBA/Owner.

- Never mix DLL instructions with DML instructions in the same transaction or be prepared to face some weird side-effects.

- Always name your constraints. Auto-named constraints can change its name when backup is restored.

The example application was implemented in Delphi 7 using IBObjects and IBOAdmin components.

The End

- **Any questions???**

