Tcl Package for Sqlite3 database schema migration

Urmita Banerjee, Yili Zhang, Güneş Koru, Clif Flynt, Stephen Huntley, and Dae-young Kim

Health IT Lab at UMBC
Department of Information System
University of Maryland, Baltimore County, Baltimore, MD

***Abstract*—Health IT Lab at UMBC leveraged the easy relationship between SQLite and Tcl/Tk to develop a package which maintains schema migration by using Tcl on the sqlite3 database. The package allows to update or revert the database's schema to a newer or older version with evolution of the database.**

I. Introduction

Database schema migration refers to a multiphase process which facilitates incremental or reversal changes of a relational database schema [1]. Application modernization over time includes code and schema changes [2]. When developing a database dependent application software the development of the source code occurs in tandem with the evolution of the database [1]. The program code typically has fixed expectations of what elements should be present in the database while interacting with it. So only the database schema version against which the code was developed is considered compatible to run as part of the program [1]. Efficiently tracking and navigating around the schema parts without affecting the existing database and program can often be challenging. To ease this process, automated migration of schema comes into view which basically allows to adapt the database as per requirement and track the granular changes affecting it. To assist this cause while developing an application using Tcl and Sqlite3, a migration package was developed using Tcl. This package allows Sqlite3 database schema migration by generating forward and backward schema modification scripts and executing them to handle the database schema change.

II. Methods

A. About the Migration Package

The migration package developed using Tcl programming language caters to basic database operations like table creation, table deletion, adding a column, removing a column, and table renaming. Functions were created in the migration package to fulfill the requirements of these changes, with generation of Tcl script files for each of these data schema changes. Each function takes up an action of database schema modification, and the script generated along with the function execution includes an ”Up” function and a ”Down” function. The ”Up” function performs forward change in the database while the ”Down” function brings about a backward change in the database. For instance, an addTable Up function for a certain table creates the table and an addTable Down function removes the table and takes the database back to the state prior to the table creation.

Sourcing the migration package in an application code sanctions desired database migration functions to be run. The execution of the migration functions creates migration script

files. The "Up" function or "Down" function of the necessary migration script files can be invoked from the application source code to perform desired database changes. The name of these script files are timestamped along with the action taken and the name of table or column modified. These timestamped migration files help track the database schema modification version. The migration package allows the timestamped script files to execute a series of data schema changes in time sequence, both in forward and inverted order. Thus, executing scripts serially with "Up" fucntions can establish the database, and executing scripts in reversed order with "Down" functions can degenerate the database.

In our application during initial stages of software development, the Up scripts for database modification were run to establish the data model. With evolution of the application, the data model also evolved. This progress was easily handled by running necessary Up and Down scripts. In our system we also maintain a record of the execution of the migration files in a table called migration so that over the time we can study how the database changed and emerged in time.

B. Utilizing the Migration Package

Below are some sample steps carried out to utilize the migration package in achieving database schema migration:

```
package require Migration
package require sqlite3

Migration::dbName test.db

Migration::addTable tbl1 {id integer \
primary key} {name text} {age text}

Migration::addTable tbl2 {id integer \
primary key} {schoolname text} \
{schooladdress text} {studentid integer} \
{foreign key(studentid) references \
tbl1(id)}
```

The above code steps create two migration script files named M00180924115537_addtable_tbl1.tcl and M00180924115540_addtable_tbl2.tcl. The files will have both Up and Down functions in them. Typically the contents of the files look like this:

```
proc up {} {
 mig transaction {

   mig eval "CREATE TABLE IF NOT EXISTS \
   tbl1 (id integer primary key, \
   name text,age text)"

   mig eval "INSERT INTO migration \
   (DQT_Version, Time, Migration_File, \
   Action) values ('$::DQTVersion', \
   '[clock format [clock seconds] \
   -format %y%m%d-%H:%M:%S]', \
   '$Migration::MigFile','Up')"

   }
}

proc down {} {
 mig transaction {

    mig eval "DROP TABLE IF EXISTS tbl1"

    mig eval "INSERT INTO migration \
    (DQT_Version, Time, Migration_File, \
    Action) values ('$::DQTVersion', \
    '[clock format [clock seconds] \
    -format %y%m%d-%H:%M:%S]', \
    '$Migration::MigFile','Down')"

    }
}


proc up {} {
 mig transaction {

    mig eval "CREATE TABLE IF NOT EXISTS \
    tbl2 (id integer primary key, \
    schoolname text, schooladdress text, \
    studentid integer, foreign key(studentid) \
    references tbl1(id))"

    mig eval "INSERT INTO migration \
    (DQT_Version, Time, Migration_File, \
    Action) values ('$::DQTVersion', \
    '[clock format [clock seconds] \
    -format %y%m%d-%H:%M:%S]', \
    '$Migration::MigFile','Up')"
```

```
    }
}

proc down {} {
 mig transaction {

    mig eval "DROP TABLE IF EXISTS tbl2"

    mig eval "INSERT INTO migration \
    (DQT_Version, Time, Migration_File, \
    Action) values ('$::DQTVersion', \
    '[clock format [clock seconds] \
    -format %y%m%d-%H:%M:%S]', \
    '$Migration::MigFile','Down')"

    }
}
```

These add table migration files can then be invoked from the source code for the desired database modification.

```
Migration::dbName finaldb.sqlite3

Migration::changeSchema up -s \
M00180924115537_addtable_tbl1.tcl \
M00180924115537_addtable_tbl1.tcl
```

A series of script files can also be mentioned in the above statement to execute a sequential pattern of database modifications as shown below.

```
Migration::dbName finaldb.sqlite3

Migration::changeSchema up -s \
M00180924115537_addtable_tbl1.tcl \
M00180924120501_addtable_tbl3.tcl
```

All scripts starting from M00180924115537_addtable_tbl1.tcl to M00180924115537_addtable_tbl3.tcl are serially executed. For the sake of this example we can assume that the above statements introduce three new tables into the database namely, tbl1, tbl2, tbl3. Similarly other database operations like renaming a table or adding a column can also be performed by invoking the desired modification scripts in the source code.

III. Results

The migration package was used in an application developed in Health IT Lab, UMBC. The migration scripts obtained from the package were integrated into the code as part of the requirement. The migration package helped construct the expected database scenarios, and enabled data schema change without affecting the existing database. With progress and evolution of the application, the data model also evolved and it was handled easily by running migration scripts.

Some of the implemented migration examples are as follows: The Up Add Table function for a particular entity introduces that table into the data model while the Down Add Table function removes the particular table from the schema if it exists and reverts the database to an older version. Similarly an Up Delete Table function for a particular entity successfully deletes the table from the database while Down Delete Table function inserts the table back into the database and constructs an earlier state of the database. The Up Rename Table function allows to give an existing table a new name while the Down Rename Table function allows the table to switch back to an older name.

In situations where the data model underwent major changes that altered the existing entity relationships i.e., deleted many existing tables and created new tables and relationships, the migration package enabled a more systematic and controlled establishment of the new data model without having to develop from scratch.

The below example explains the above case:

```
Migration::dbName finaldb.sqlite3

Migration::changeSchema up -s \
M00180925120511_deletetable_tbl1.tcl \
M00180925123012_deletetable_tbl3.tcl

Migration::changeSchema up -s \
M00180925155537_addtable_newtbl1.tcl \
M00180925162534_addtable_newtbl3.tcl
```

The above statements integrated in the source code runs the scripts from M00180925120511_deletetable_tbl1.tcl to M00180925123012_deletetable_tbl3.tcl and M00180925155537_addtable_newtbl1.tcl to

M00180925162534_addtable_newtbl3.tcl and drops the tables tbl1, tbl2 and tbl3 from the database and creates three new tables namely, newtbl1, newtbl2, and newtbl3. Thus we have a new data model implemented without having to develop it from the beginning. During software testing, we could easily test the harmony between the database structure and the application code by using the schema migration process on test databases [1].

IV. Conclusion

Database Schema Migration is an essential process in agile software development. It helps adapting database evolution and keeps the database state compatible with the program code. Schema Migration allows the database schema to be updated to a new state or reverted to an earlier state and its evolution can be tracked. It is a time efficient process and its utilization removes the need to fully redesign data models up-front with every little alterations in the database [1]. The primary purpose of schema migration is to handle database evolutions without impacting the existing data in the database. For system like ours where the database structure needs to be set and compatible with the code expectations, the migration scripts allowed to tackle changes in the database structure without any failure in running the application. However, there are certain risks associated with migration. Preservation of data in general is a concern when it comes to migration and it is not guaranteed as schema changes like column deletion can affect data negatively. In cases of large databases, migration can lead to unexpected problems if there is still data introduced by old software that was not removed properly or if the relationships between the entities are not well thought before executing the migration steps which can lead to integrity failures [1]. Thus it can be concluded that schema migration is a very useful process and this Tcl migration package helps achieve schema migration soundly in a Tcl-Sqlite3 environment.

References

[1] Schema Migration Wikipedia;. (Accessed on 09/24/2018). http://www.webcitation.org/72gGFm7xa.

[2] Database Migration Blog;. (Accessed on 09/24/2018). http://www.webcitation.org/72gGjYfEc.