Java DB Migrator Documentation

INTRO

Java Migration is related to Evolutionary Database Design which is very important capability for agile methodologies. In agile methodologies developer wants to release their changes into database easily and in a good and fashionable manner.

The whole story briefly is that developers in consulting with DBAs create changing scripts and release them with software versions and their changes should be affect the database in a true way.

There are some advantages and disadvantages in comparison to Traditional Database Design and many articles exist that describes each of which designs from different aspects and from different view of different developers, DBAs and so on.

Java DB Migrator Documentation

JAVA MIGRATION PROJECT PROCESS

The whole process is as below:

1. Firstly an empty Database Migration File should be created by “generate()” method in Main Class or by generate task of Gradle
   The name of file contains specified name defined when generated (by default: migration) followed by a simple timestamp string

2. Related Database Migration File Should be filled by simple sample commands, I have provided Command Templates in file “templates.dbmgr”.

   Examples:

```
ADD                                TABLE                                table_name
ADD           COLUMN           table_name          column_name          column_type
CHG        COLUMN_NAME        table_name        column_old_name        column_new_name
CHG        COLUMN_TYPE        table_name        column_old_type        column_new_type
ADD ROW table_name columns#values
```

   Commands are in uppercase and lowercase parts should be filled by appropriate values. It’s very simple to define and parse commands containing regular expression for affecting some numbers of tables, columns, etc. we have a test method (dynamicTables()) in project for testing this capability.

3. Creating “schema_history” table via first time call of “migrate()” method from program or through calling related Gradle Task.

   “shema_history” table is a table for keeping logs of executed migrations and preventing them to be executed again. It also can keep valuable data like the time which migration is installed, execution time and many other things in a real production product like the person who has done the migration and so on.

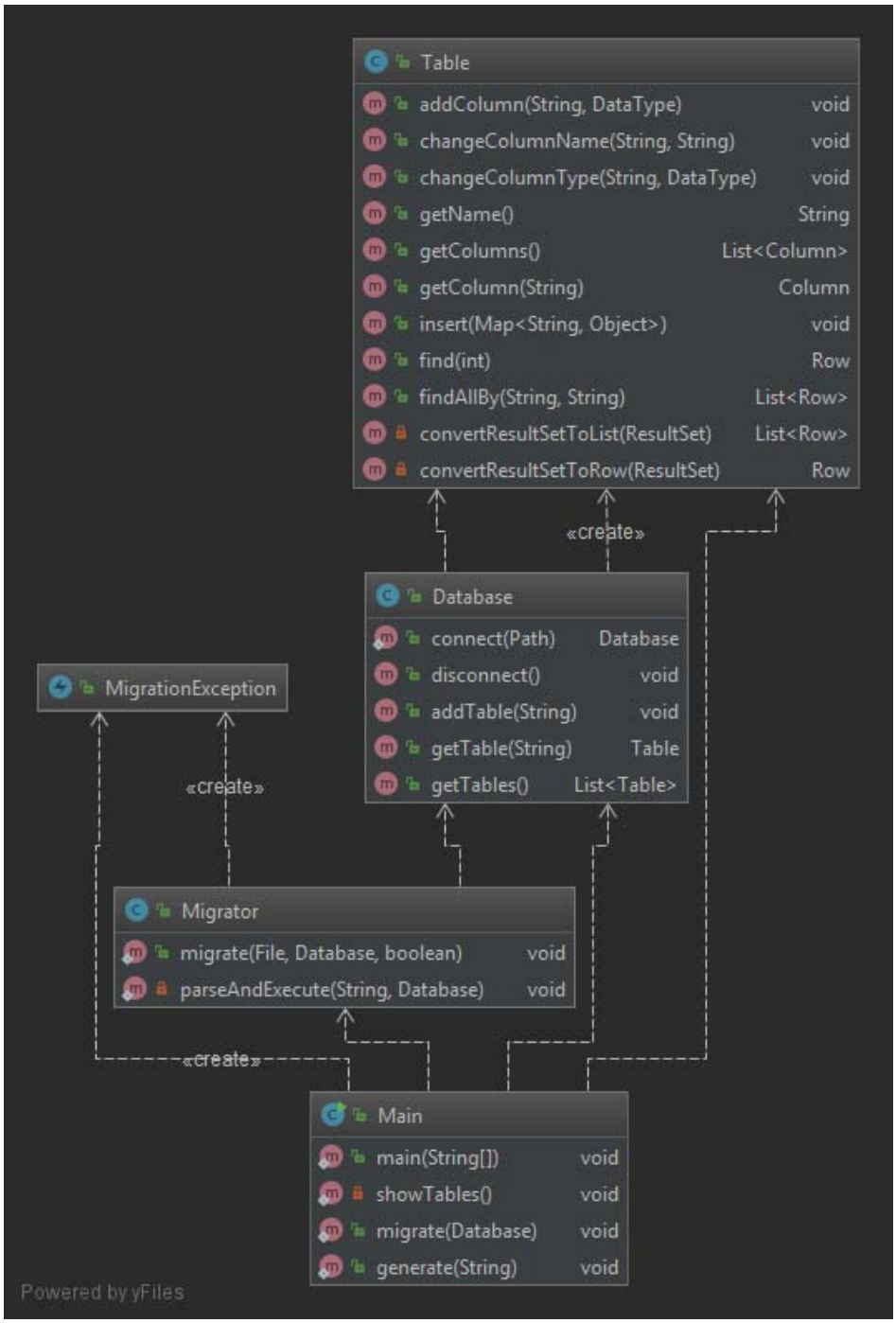
In my case table has three columns: version, installed_on, execution_time, Version is the name of the file which is generated in generation phase and make a migration unique from others.

4. Final step is calling “migration()” again, in this step all the generated changes inside related migration files which has not done before (based on information in schema_history table) will affect defined database.

Java DB Migrator Documentation

CODE DESCRIPTION

Class Diagram of Main Classes of Project is as below:



There are two main methods in Main class, As I mentioned before “generate()” creates related migration files and “migrate()” migrates changes into database. “migrate()” calls “migrate()” method of Migrator class in its body and “migrate()” method of Migrator class, uses “parseAndExecute()” method for parsing related command defined in migrations files and execute commands via database API (dbapi) to database.

Adding other commands is very easy in this way for instance for even NoSQL databases we can simply define commands and related parser and extending database API (dbapi) for executing related database command on database.

Java DB Migrator Documentation

FURTHER WORKS

- It is possible to extend database API for different kind of databases with different kind of scripts or structure like NoSQL databases
- We can have a roll back strategy for rolling back changes in the case of failures or something else
- It is applicable to keep extra information in schema_history table for logging and tracking beside its main responsibility of preventing migration again
- Another feature is scheduling migration for a specific time or condition in the future
- The parsing strategy and migration commands templates can also be extended and optimized in a better and convenient way.