How to Initialize a Database on Application Startup.

by Fyodor Kupolov

NOTICE: Work in progress

This How-To describes the steps for automated database schema initialization.

Table of contents

1 Intended Audience

Developers/DBAs.

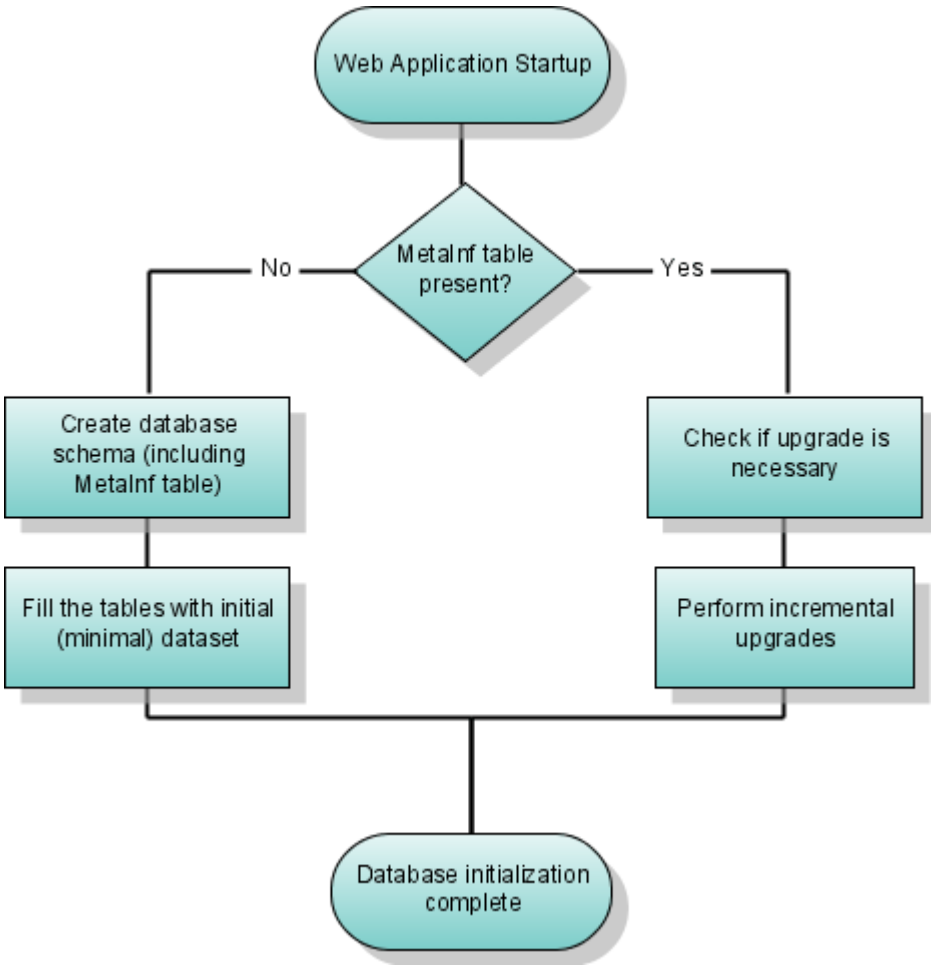
2 Purpose

Database initialization is an important part of application deployment. Typically a DBA applies a set of SQL scripts to initialize a database or perform an upgrade. While this manual step is reasonable for large applications with complex deployment scenarios there are plenty of projects where DB can be initialized automatically on application startup. Examples:

- Demo applications. Many example applications require the user to run db init scripts thus making an entry level more complex.
- Applications with automated installation procedure. The database can be initialized during a setup process or on application startup.
- Small to medium-sized projects where deployment is performed by a customer and has to be as simple as possible.

This how-to we describes one of the possible approaches to automate DB initialization. "Image Database" example from the Spring Framework distribution is used as a base application.

The following diagram demonstates a simplified database initialization/upgrading scenario:



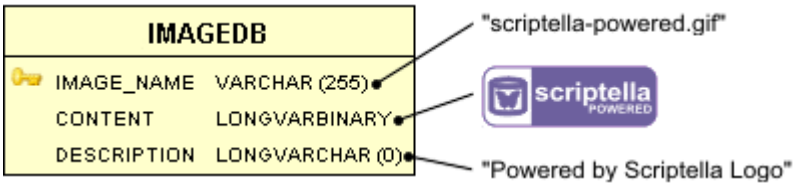
MetaInf table is used as a flag to check if the database has already been created. Additionally this table stores the database model meta information, e.g. build number or an application version. For simple deployment cases which do not require an upgrade procedure, this table may be omitted.

3 Prerequisites

If you want to reproduce the how-to steps manually download Spring 2.0 binary distribution with dependencies and unpack it. The *ImageDB* example files are located in a *samples/imagedb* folder.

4 Steps

4.1 Database Schema Initialization Script



The ImageDB database model consists of the only one table. Nevertheless filling the database can be tricky due to several reasons:

- DB script contains vendor specific SQL data types, e.g. BLOB/LONGVARBINARY or LONGTEXT/CLOB etc.
- It's problematic to upload BLOB content in a database neutral way.

And now Scriptella comes into play. The following script creates a database and populates it with the initial dataset:

```
<!DOCTYPE etl SYSTEM "http://scriptella.org/dtd/etl.dtd">
<etl>
    <properties> <!-- Just include external properties -->
        <include href="webinit.etl.properties"/>
    </properties>
    <connection driver="$driver" url="$url" user="$user" password="$password" />
    <script>
        <!-- Metainf table stores version information -->
        CREATE TABLE Metainf (
            buildnum INTEGER
        );
        INSERT INTO Metainf VALUES (1);
        <!-- Conditional schema creation scripts-->
        <dialect name="hsql">
            <include href="hsqldb-schema.sql"/>
        </dialect>
        <dialect name="oracle">
            <include href="oracle-schema.sql"/>
        </dialect>
        <dialect name="mysql">
            <include href="mysql-schema.sql"/>
        </dialect>
        <!-- Fill the table with data -->
        <include href="data.sql"/>

        <!-- If Metainf present (table Metainf already exists),
             skip schema creation and continue -->
        <onerror message=".*Metainf.*"/>
    </script>
    <!-- Optional upgrade procedure -->
    <query>
        <!-- Selects current DB build -->
        SELECT * FROM Metainf
        <!-- Check if upgrade is necessary -->
        <script if="buildnum lt 1">
            <!--Upgrades DB to build 1 -->
            <!--...-->
            <!-- Update Metainf to confirm successful upgrade -->
            UPDATE Metainf SET buildnum=1;
```

```
        </script>
        <!-- upgrade scripts for subsequent builds -->
    </query>
</etl>
```

- The **-schema.sql* files are simple SQL files identical to the txt files located in a *db* folder of the ImageDB example.
- The *data.sql* file contains insert statements for initial dataset:

```
-- ImageDB initial dataset.
-- ?{file ... } SQL syntax extension allows referencing BLOBs in external files.
-- See Reference Manual for more details on JDBC escaping and other syntax extensions.

INSERT INTO imagedb(image_name, content,description) VALUES ('scriptella-logo.png',
      ?{file 'blobs/scriptella-logo.png'}, 'Scriptella ETL logo');
INSERT INTO imagedb(image_name, content,description) VALUES ('scriptella-powered.gif',
      ?{file 'blobs/scriptella-powered.gif'}, 'Powered by Scriptella Logo');
```

The database is created when a web application is started for the first time. For subsequent startups the initializing procedure is skipped and the information message similar to the following one is printed on the console:

```
INFO: Script /etl/script[1] failed: scriptella.jdbc.JdbcException:
Unable to execute statement. Error statement:
CREATE TABLE Metainf (
buildnum INTEGER
). Error codes: [S0001, -21]
Using onError handler: OnError{message=.*Metainf.*, codes=[], retry=false}
```

4.2 Integration with a web application

To integrate a DB initializing procedure with a web application put the following files into a folder inside the WAR file, e.g. */WEB-INF/db*:

- *webinit.etl.xml* – Scriptella database initialization file.
- *webinit.etl.properties* – configuration properties for webinit.etl.xml. Spring or JNDI managed data sources or driver classes can be specified here.
- *...-schema.sql* - schema creation scripts for different databases. These scripts are included by a main etl.xml file.
- *data.sql* – initial dataset.
- *blobs* – binary data referenced from data.sql file.

Additionally the *hsqldb.jar* (or other JDBC driver) should be copied to *WEB-INF/lib* dir.

4.3 Automatic initialization in a ServletContextListener

To execute *webinit.etl.xml* on web application startup create an implementation of `ServletContextListener`:

```
    public class WebDbInitializer implements ServletContextListener {
        private static final String WEBINIT_ETL_PATH = "webinit.etl.path";
        /**
         * Executes script which inits the database.
         * @param etlUrl ETL file URL.
         * @throws EtlExecutorException if script execution fails.
         */
        static void initDatabase(URL etlUrl) throws EtlExecutorException {
            EtlExecutor exec = EtlExecutor.newExecutor(etlUrl);
            exec.execute();
```

```
        }

        public void contextInitialized(ServletContextEvent servletContextEvent) {
            ServletContext ctx = servletContextEvent.getServletContext();
            try {
                initDatabase(ctx.getResource("/WEB-INF/db/webinit.etl.xml"));
                ctx.log("DB script executed");
            } catch (Exception e) {
                ctx.log("Unable to execute DB script", e);
            }
        }
    . . .
    }
```

This listener is registered in the web.xml file using the following snippet:

```
    <web-app>
        <listener>
            <listener-class>scriptella.imagedb.WebDbInitializer</listener-class>
        </listener>
    </web-app>
```

The *webinit.etl.properties* file has the following content for deployment on Tomcat:

```
    driver=hsqldb
    url=jdbc:hsqldb:file:${catalina.home}/db/imagedb
    user=sa
    password=
```

4.4 Integration with Spring Framework

Integration with Spring is even simpler, just add the following XML bean declaration to the application context xml file:

```
    <bean class="scriptella.driver.spring.EtlExecutorBean">
        <property name="configLocation" value="/WEB-INF/db/webinit.etl.xml"/>
        <property name="autostart" value="true"/>
    </bean>
```

In this case the *webinit.etl.properties* file has the following content:

```
driver=spring
url=dataSource
```

5 Resources

Sample application downloads:

- imagedb-spring.war - Modified ImageDB WAR file for Spring Framework. Simply deploy it to Tomcat and run, the database will be created automatically in a <TOMCAT_HOME>/db directory.
- imagedb.war - Demo WAR file which creates ImageDB database automatically (No Spring integration).

Articles:

- "Evolutionary Database Design" (Martin Fowler, Pramod Sadalage)