IBM

developerWorks.

Automation for the people: Hands-free database migration

Use LiquiBase to manage database changes

Skill Level: Introductory

Paul Duvall (paul.duvall@stelligent.com)
CTO
Stelligent Incorporated

05 Aug 2008

Databases are often out of sync with the applications they support, and getting the database and data into a known state is a significant challenge to manage. In this installment of *Automation for the people*, automation expert Paul Duvall demonstrates how the open source LiquiBase database-migration tool can reduce the pain of managing the constant of change with databases and applications.

Most of the applications I've worked with over the years have been enterprise applications requiring the management of lots of data. Development teams working on such projects often treat the database as a completely separate entity from the application. This sometimes stems from an organizational structure that separates the database team from the application-development teams. Other times, it's simply what teams are used to doing. Either way, I've found that this separation leads to some of the following practices (or lack thereof):

- Manually applying changes to the database
- Not sharing database changes with other members of the team
- Inconsistent approaches to applying (database or data) changes
- Ineffective manual mechanisms for managing changes from one database version to the next

These are inefficient practices that leave developers out of sync with data changes.

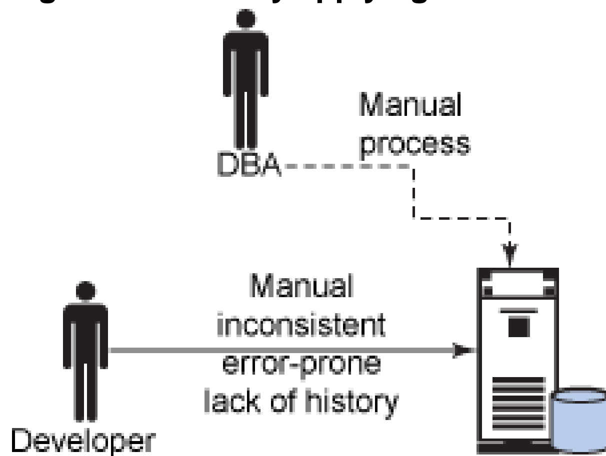
© Copyright IBM Corporation 1994, 2008. All rights reserved.

Moreover, they can cause the application's *users* to experience problems with inconsistent or corrupt data.

Figure 1 illustrates the manual approach that's often used on software development projects. Manual changes are often inconsistently applied and error prone, and they can make it difficult to undo what's already been done or analyze the history of database changes over time. For example, a DBA might remember to apply changes to the lookup data on one occasion, but a developer might forget to insert this data later into the same table.

> **About this series**
> As developers, we work to automate processes for end-users; yet, many of us overlook opportunities to automate our own development processes. To that end, *Automation for the people* is a series of articles dedicated to exploring the practical uses of automating software development processes and teaching you *when* and *how* to apply automation successfully.

Figure 1. Manually applying database changes



You can avoid the manual approach's pitfalls by implementing a database change strategy that minimizes human intervention. Through a combination of practices and tools, you can use a consistent and repeatable process for applying changes to your database and data. In this article, I'll cover:

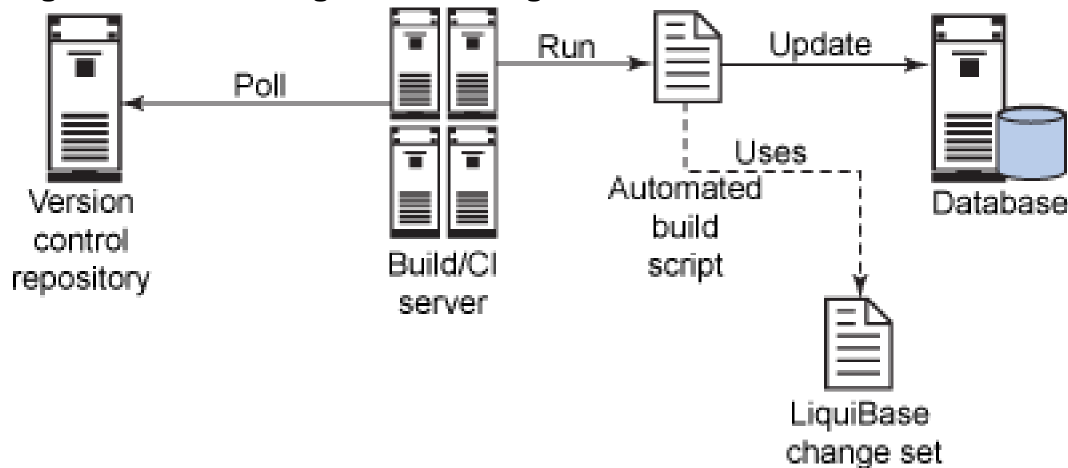
- Using a tool called LiquiBase to migrate between database versions
- How to run database migrations automatically
- Practices for consistently applying database changes
- Applying database refactorings using LiquiBase

In Figure 2, a Build/Continuous Integration server polls for changes to the version-control repository (such as Subversion). When it finds a change in the

© Copyright IBM Corporation 1994, 2008. All rights reserved.

repository, it runs an automated build script that uses LiquiBase to update the database.

Figure 2. Automating database migration



By using a process like the one illustrated in Figure 2, anyone on the team can apply the same changes to the database — either locally or on a shared database server. Moreover, because the process uses automated scripts, these changes can be applied in different environments without anyone laying a finger on the database and its contents.

Which is it? DDL, DML, schemas, databases, or data

In this article, I use the term *database changes* to refer to the structural changes to a database through the application of Data Definition Language (DDL) scripts. (Some database vendors refer to this as a *schema*.) And I'll refer to changes applied to a database via Data Manipulation Language (DML) scripts simply as *data* changes.

Managing database changes with LiquiBase

LiquiBase — available since 2006 — is an open source, freely available tool for migrating from one database version to another (see Resources). A handful of other open source database-migration tools are on the scene as well, including openDBcopy and dbdeploy. LiquiBase supports 10 database types, including DB2, Apache Derby, MySQL, PostgreSQL, Oracle, Microsoft® SQL Server, Sybase, and HSQL.

To install LiquiBase, download the compressed LiquiBase Core file, extract it, and place the included liquibase-*version*.jar file in your system's path.

Getting started with LiquiBase takes four steps:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

1. Create a database *change log* file.
2. Create a *change set* inside the change log file.
3. Run the change set against a database via the command line or a build script.
4. Verify the change in the database.

Creating a change log and change set

The first step to running LiquiBase, as demonstrated in Listing 1, is to create an XML file known as the database change log:

Listing 1. Defining a change set in a LiquiBase XML file

```
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog/1.7"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog/1.7
         http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-1.7.xsd">
  <changeSet id="2" author="paul">
    <createTable tableName="brewer">
      <column name="id" type="int">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="name" type="varchar(255)">
        <constraints nullable="false"/>
      </column>
      <column name="active" type="boolean" defaultValue="1"/>
    </createTable>
  </changeSet>
</databaseChangeLog>
```

XML ... again

Some live in a world of XML scripts and others do not. Many developers have grown accustomed even to programming using XML scripts (for example, with Apache Ant), but this isn't necessarily the environment DBAs live in. Recently, I was excited to show a DBA colleague some of LiquiBase's features. He liked many of its powerful tools for managing database changes but was skeptical whether DBAs would embrace the XML-based syntax. I assured him that LiquiBase also supports calling custom SQL scripts with its `sqlFile` and `sql` custom refactorings.

As you can see, the database change log file includes a reference to an XML schema (the dbchangelog-1.7.xsd file included in the LiquiBase installation). In the change log file, I create a `<changeSet>`. Within the `<changeSet>`, I apply changes to the database in a structured manner, as defined in the LiquiBase schema.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Running LiquiBase from the command line

After defining the change set, I can run LiquiBase from the command line, as shown in Listing 2:

Listing 2. Running LiquiBase from the command line

```
liquibase
--driver=org.apache.derby.jdbc.EmbeddedDriver \
--classpath=derby.jar \
--changeLogFile=database.changelog.xml \
--url=jdbc:derby:brewery;create=true \
--username= --password= \
update
```

In this example, I run LiquiBase passing in:

- The database driver
- The classpath for the location of the database driver's JAR file
- The name of the change log file that I created (shown in Listing 1) called database.changelog.xml
- The URL for the database
- A username and password

Finally, Listing 2 calls the `update` command to tell LiquiBase to make my changes to the database.

Running LiquiBase in an automated build

Instead of using the command-line option, I can make the database changes as part of the automated build by calling the Ant task provided by LiquiBase. Listing 3 shows an example of this Ant task:

Listing 3. Ant script to execute the updateDatabase Ant task

```
<target name="update-database">
  <taskdef name="updateDatabase" classname="liquibase.ant.DatabaseUpdateTask"
    classpathref="project.class.path" />
  <updateDatabase changeLogFile="database.changelog.xml"
    driver="org.apache.derby.jdbc.EmbeddedDriver"
    url="jdbc:derby:brewery"
    username=""
    password=""
    dropFirst="true"
    classpathref="project.class.path"/>
</target>
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

In Listing 3, I create a target called `update-database`. In it, I define the specific LiquiBase Ant task I wish to use, calling it `updateDatabase`. I pass the required values, including the `changeLogFile` (which specifies the change log file defined in Listing 1) and connection information for the database. The classpath defined in `classpathref` must contain liquibase-*version*.jar.

Before and after

Figure 3 shows the state of the database before I run the change set in Listing 1:

Figure 3. Database state before running the LiquiBase change set

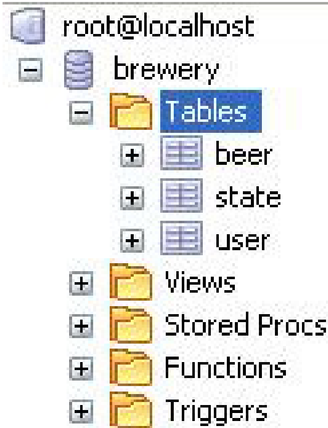
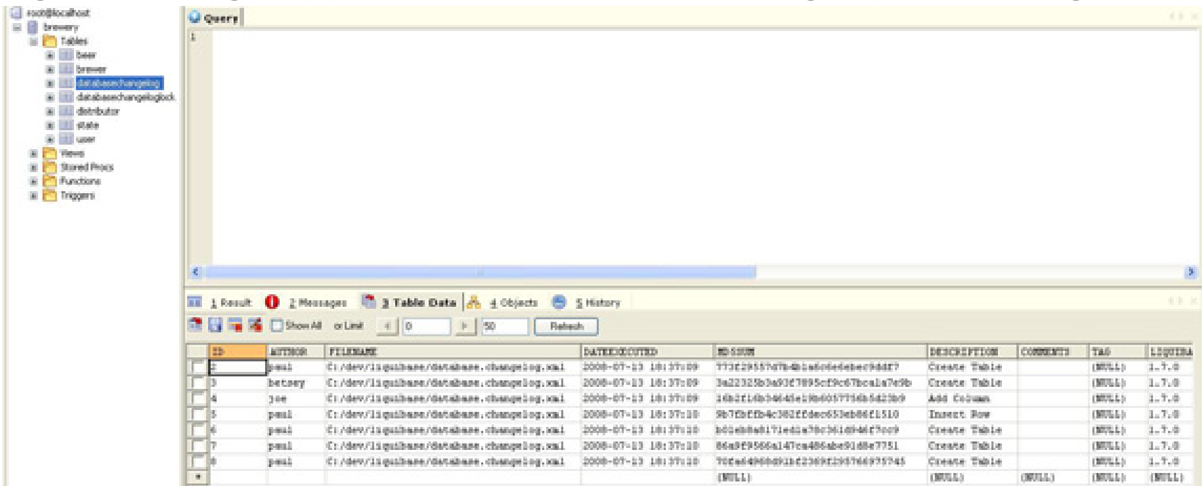


Figure 4 shows the results of running the database change set either through the command line (as shown in Listing 2) or from Ant (as shown in Listing 3):

Figure 4. Changes applied to a database after running LiquiBase change set



See the full figure here.

Several aspects of Figure 4 are worth mentioning. Two LiquiBase-specific tables were created, along with the new tables that were created based on the change set definition from Listing 1. The first LiquiBase-specific table, called

© Copyright IBM Corporation 1994, 2008. All rights reserved.

`databasechangelog`, keeps track of all of the changes applied to the database — useful for tracking who made a database change and why. The second LiquiBase-specific table, `databasechangelock`, identifies the user that has a lock on making changes to the database.

You can run LiquiBase in many other ways too, but I've given you most of what you need to know to apply database changes. As you use LiquiBase, you'll spend much of your time learning the different ways to apply database refactorings, along with the complexities in making changes to your particular database. For example, LiquiBase provides support for database rollbacks, which can present numerous challenges. Before I show you examples of database refactorings, I'll quickly share some of the basic principles and practices of database integration that will help you get the most from database migration.

Integrating database changes often

In recent years, development teams have been applying principles and practices for managing their database assets that are similar to the ones they follow when working with source code. So, it's a natural progression to apply practices such as scripting database changes, sharing these assets in a source code repository, and integrating the changes into the build and Continuous-Integration process. Table 1 provides an overview of the key practices that development teams should follow when making database changes a part of an automated process:

Automate your DBA

On some of the projects I've worked on, DBAs have created unnecessary bottlenecks by controlling changes to a development database. DBAs should be spending time on creative, nonrepetitive activities such as monitoring and improving database performance, not doing unnecessary repetitive work that is sold to management under the guise of control and consistency.

Table 1. Database-integration practices

Practice	Description
Scripting all DDL and DML	Database changes should be capable of being run from the command line.
Source control for data assets	A version-control repository is used to manage all database-related changes.
Local database sandbox	Each developer uses a local database sandbox to make changes.
Automated database integration	Database-related changes are applied as part of the build process.

These practices ensure better consistency and prevent changes from getting lost

© Copyright IBM Corporation 1994, 2008. All rights reserved.

from one software release to the next.

Applying refactorings to an existing database

As new features are added to an application, the need often arises to apply structural changes to a database or modify table constraints. LiquiBase provides support for more than 30 database refactorings (see Resources). This section covers four of these refactorings: Add Column, Drop Column, Create Table, and manipulating data.

Add Column

It's sometimes next to impossible to consider all of the possible columns in a database at the beginning of a project. And sometimes users request new features — such as collecting more data for information stored in the system — that can require new columns to be added. Listing 4 adds a column to the `distributor` table in the database, using the LiquiBase `addColumn` refactoring:

Listing 4. Using the Add Column database refactoring in a LiquiBase change set

```
<changeSet id="4" author="joe">
  <addColumn tableName="distributor">
    <column name="phonenumber" type="varchar(255)"/>
  </addColumn>
</changeSet>
```

The new `phonenumber` column is defined as a `varchar` datatype.

Drop Column

Let's say that a couple of releases later, you choose to remove the `phonenumber` column you added in Listing 4. This is as simple as calling the `dropColumn` refactoring, as shown in Listing 5:

Listing 5. Dropping a database column

```
<dropColumn tableName="distributor" columnName="phonenumber"/>
```

Create Table

Adding a new table to a database is also a common database refactoring. Listing 6 creates a new table called `distributor`, defining its columns, constraints, and default values:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 6. Creating a new database table in LiquiBase

```
<changeSet id="3" author="betsey">
  <createTable tableName="distributor">
    <column name="id" type="int">
      <constraints primaryKey="true" nullable="false"/>
    </column>
    <column name="name" type="varchar(255)">
      <constraints nullable="false"/>
    </column>
    <column name="address" type="varchar(255)">
      <constraints nullable="true"/>
    </column>
    <column name="active" type="boolean" defaultValue="1"/>
  </createTable>
</changeSet>
```

This example uses the `createTable` database refactoring as part of a change set (`createTable` was also used back in Listing 1).

Manipulating data

After applying structural database refactorings (such as Add Column and Create Table), you often need to insert data into tables affected by these refactorings. Furthermore, you might need to change the existing data in lookup tables or other types of tables. Listing 7 shows how to insert data using a LiquiBase change set:

Listing 7. Inserting data in a LiquiBase change set

```
<changeSet id="3" author="betsey">
  <code type="section" width="100%">
  <insert tableName="distributor">
    <column name="id" valueNumeric="3"/>
    <column name="name" value="Manassas Beer Company"/>
  </insert>
  <insert tableName="distributor">
    <column name="id" valueNumeric="4"/>
    <column name="name" value="Harrisonburg Beer Distributors"/>
  </insert>
</changeSet>
```

You may have already written SQL scripts to manipulate data, or the LiquiBase XML change set may be too limiting. And sometimes it's simpler to use SQL scripts to apply mass changes to the database. LiquiBase can accommodate these situations too. Listing 8 calls `insert-distributor-data.sql` within a change set to insert the `distributor` table data:

Listing 8. Running a custom SQL file from a LiquiBase change set

```
<changeSet id="6" author="joe">
  <sqlFile path="insert-distributor-data.sql"/>
</changeSet>
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

LiquiBase provides support for many other database refactorings, including Add Lookup Table and Merge Columns. You can define all of them in a manner similar to what I've shown in Listings 4 through 8.

Constantly keeping data in sync

In software development, if something hurts, you should do it more often and not wait until the later development cycles to perform these operations manually when they are more costly and painful. Migrating a database is not a trivial exercise, and it can benefit tremendously from automation. In this article, I've:

- Demonstrated how to use LiquiBase to script database migrations and make these changes a part of the automated build process
- Described the principles and practices of database integration that lead to consistency
- Showed how to apply database refactorings such as Add Column, Create Table, and updating data through the use of LiquiBase

Table 2 summarizes the list of some of the features that LiquiBase provides:

Table 2. Summary of some of LiquiBase features

Feature	Description
Support for multiple databases	Supports DB2, Apache Derby, MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Sybase, HSQL, and others.
View a history of changes applied to a database	Using the `databasechangelog` table, you can view every change applied to the database.
Generates database diff logs	Learn about changes applied to the database outside the LiquiBase change sets.
Capable of running custom SQL scripts	Use LiquiBase to call SQL scripts that you've already written.
Utilities to roll back database changes	Supports a process for rolling back any changes that were applied to the database.

You can now see that when applied appropriately through automated scripts, database migration can be a less painful and more repeatable process that many members of your team can run.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Resources

Learn

- LiquiBase: Access LiquiBase resources at the project Web site, including the complete list of database refactorings that LiquiBase supports.
- "Incremental Migration": (Martin Fowler, martinfowler.com, July 2008): Fowler argues that because data migration difficult, we should do it more often.
- *Refactoring Databases: Evolutionary Database Design*: (Scott W. Ambler and Pramod J. Sadalage, Addison-Wesley Professional, 2006): Learn how to evolve database schemas in step with source code.
- *Continuous Integration: Improving Software Quality and Reducing Risk* (Paul Duvall, Steve Matyas, and Andrew Glover, Addison-Wesley Signature Series, 2007): Examples in Chapter 5, "Continuous Database Integration," demonstrate how to incorporate database integration into the automated build process.
- *Recipes for Continuous Database Integration: Evolutionary Database Development*: (Pramod Sadalage, Addison-Wesley Professional, 2007): Find out how the database can be brought under the purview of Continuous Integration.
- "Evolutionary Database Design": (Fowler and Sadalage, martinfowler.com, 2003): Read about applying Continuous Integration and automated refactoring to database development.
- "Database Integration in your Build scripts": (Paul Duvall, testearly.com, June 2006): Learn about running DDL and DML scripts as part of an automated build process.
- Browse the technology bookstore for books on these and other technical topics.
- developerWorks Java™ technology zone: Hundreds of articles about every aspect of Java programming.

Get products and technologies

- LiquiBase: Download LiquiBase to begin performing automated database migrations
- Ant: Download Ant and start building software in a predictable and repeatable manner.

Discuss

- Improve Your Code Quality discussion forum: Regular developerWorks contributor Andrew Glover brings his considerable expertise as a consultant focused on improving code quality to this moderated discussion forum.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- Accelerate development space: Regular developerWorks contributor Andrew Glover hosts a one-stop portal for all things related to developer testing, Continuous Integration, code metrics, and refactoring.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Paul Duvall
Paul Duvall is the CTO of Stelligent Incorporated, an Agile consultancy that helps development teams deliver production-ready software. He is the co-author of the Addison-Wesley Signature Series book *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley Professional, 2007; Jolt Award 2008 winner). He also contributed to the *UML 2 Toolkit* (Wiley, 2003) and the *No Fluff Just Stuff Anthology* (Pragmatic Programmers, 2007).

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

© Copyright IBM Corporation 1994, 2008. All rights reserved.