Liquibase

Version Control For Your Schema

Nathan Voxland
April 3, 2014
nathan@liquibase.org
@nvoxland

PERCONA LIVE

Agenda

LIQUIBASE

- Why Liquibase
- Standard Usage
- Tips and Tricks
- Q&A

Why Liquibase?

You would **never** develop code without version control

Why do you treat your database differently??

Why Liquibase?

- Show of hands:
  - Who uses Git, Subversion, CVS, whatever to manage their code? EVERYONE
    - Nobody emails word docs with Java/Python/C#/whatever to the a guy who applies them to the real codebase
  - But, manages SQL to deploy via word docs in email?
    - Maybe fancy and manages them in your bug database?

Why Liquibase?

Why???

Why Liquibase?

- The Database is as important as your code
  - Maybe more important: algorithms come and go but data is forever
- But you treat it in ways that you would never treat your source code
  - You have separate and unrelated processes for code and schema deployments that you hope work correctly in the end
  - You have no visibility into how your data model has changed over time or why

Why Liquibase?

Fear leads to Anger

Anger leads to Hate

Hate leads to Suffering

Why Liquibase?

- Just think of all the things you take for granted with Git or even CVS
  - commit, update, revert, branches, history log
- Without these tools you would **fear** making changes to your code
  - When you fear making changes, you don't make changes and instead angrily develop work arounds
- Those ever more elaborate work-arounds slow you down and **anger** you.
  - They introduce bugs. They confuse new developers and DBAs.
- The database becomes something everyone **hates** to deal with
- In the end everyone **suffers** because the database is core of your application

Why Liquibase?

Store database changes **with** your source code

Changes are **consistently** applied to every database

Why Liquibase?

- **Liquibase breaks this cycle with a simple structure:**

- #1: Liquibase allows you to track changes in a simple text format
  - Automatically works with all your existing source control tooling
    - Branch, merge, log all work
  - Human readable so you can always look at your change log and know what is going on
- #2: Liquibase allows you to apply changes consistently to every database from development through production
  - Every database knows what has already been run so Liquibase will only run new changes
  - Many ways to run Liquibase:
    - manually, chef/puppet, build server, scripted, embedded in your application
    - however your run liquibase, it ensures every database is updated the same way

Why Liquibase?


```xml
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
            http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

    <changeSet id="1" author="nathan">
        <createTable tableName="employee">
            <column name="id" type="int" autoIncrement="true">
                <constraints primaryKey="true"/>
            </column>
            <column name="first_name" type="varchar(255)"/>
            <column name="last_name" type="varchar(255)">
                <constraints nullable="false"/>
            </column>
        </createTable>
    </changeSet>

</databaseChangeLog>
```

Why Liquibase?

- That is what a Liquibase databasechangelog file looks like
  - Showing XML but it can also be YAML, JSON or SQL for all you XML-haters out there.
- Designed to be easy to read and work with
- Things to notice:
  - ChangeSet is what liquibase tracks
    - ID can be integer, bug number, whatever makes sense
- ChangeSet contains the change you want to make
  - Low level changes like “createTable”
  - High level refactoring like “introduce lookup table”
  - Raw SQL if you need it

Why Liquibase?


```
~$ liquibase update
```

Why Liquibase?

- Once you have a changelog file
  - run Liquibase update
    - pass in your connection url, username, password, changelog path and other flags as needed
- Can run as many times as you want, Liquibase tracks what needs to run

Why Liquibase?


```
10                  <constraints primaryKey="true"/>
11              </column>
12              <column name="first_name" type="varchar(255)"/>
13              <column name="last_name" type="varchar(255)">
14                  <constraints nullable="false"/>
15              </column>
16          </createTable>
17      </changeSet>
18
19      <changeSet id="2" author="nathan">
20          <addColumn tableName="employee">
21              <column name="title" type="varchar(50)"/>
22          </addColumn>
23      </changeSet>
24
25  </databaseChangeLog>
```

Why Liquibase?

- When you have another schema change to make: just append to the existing changelog
  - Don't modify the existing changeSet
- When you run `liquibase update` the next time, only changeSet 2 will run.

Why Liquibase?

Not Diff Based

Liquibase will run only what you tell it to run

Why Liquibase?

- Notice Liquibase does not look at the existing structure of the database
  - It just asks the database what has already ran and runs whatever is new.
- A common approach when trying to manage schemas is to store what you want your database to look like, then automatically fix your target database to match the model
  - Good in theory but quickly falls apart when there are multiple paths to the same end schema that can do bad things to your data
    - Example: rename
  - Diff based tools also can't easily handle things like data transformations or contextual expected differences
  - Diff tools force you to hope the tool is doing what you expect on every database
- Liquibase has some diff-related features, but it is not core to the standard workflow or what you should be relying in in your processes

Why Liquibase?

Change Sets are tracked independently

Why Liquibase?

- Another common approach to managing schemas is having a single “database version”.
- Falls apart with multiple developers, multiple branches, hotfixes
- Liquibase tracks and applies each change separately

Normal Use

Every time you update code:

```
~$ liqubase update
```

Standard Usage

- Standard Usage: Whenever you update or deploy your code, run `liquibase update` to update your database
- Do this consistantly across dev, QA, production
  - Can be different methods of running Liquibase, but always run through liquibase

Normal Use

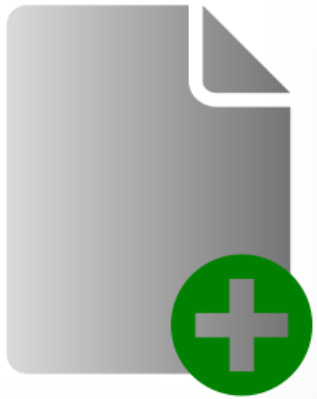
1. Append to the change log
2. `~$ liquibase update`
3. Test
4. Commit changelog with rest of code
5. GOTO 1

Standard Usage

- When you need to make an update to your database, follow those 5 steps

Best Practices

- Only append to your change log



Standard Usage

- Liquibase runs changeSets in the order defined from top to bottom, even if nesting change logs
- Adding change sets in the middle can introduce dependency issues
  - If you have changeSets A,B,C and that have ran against a database then insert to have A, D, B, C the original database will have ran as A,B,C,D but new databases will run as A,D,B,C
    - Can be OK, can be a problem
- Do not modify change sets
  - If an old version already ran, who knows how it is different and will exit because what is being requested is different than what already ran

Best Practices

- Pay attention to your change log path

Standard Usage

- Change log path is part of the change set identifier
  - Allows you to not have to worry about “id” uniqueness across changelog files
- What path gets stored can depend on how you run liquibase
  - Be careful to not have system dependent absolute paths
  - Use relative paths
- Select from databasechangelog to see what is being stored

Best Practices

- One change per change set

3!

Standard Usage

- Liquibase tries to run changeSets as transactions but Mysql and other databases autocommit DDL statements
- If a changeSet has a second create table that fails but the first passed, you are stuck because Liquibase will keep trying to run the first create table that will fail with “table exists”

Best Practices

- All changes managed through Liquibase

Standard Usage

- Always use Liquibase
  - In development: avoids “works on my machine” syndrome caused by a who forgot to mention a change they made
  - In production: all changes in production should be reflected in QA and dev environments to avoid release-time surprises

Best Practices

- Individual development databases

Standard Usage

- Shared database goes against many of the advantages Liquibase is trying to give you
  - Concurrent branch development
- Changes other made may not be compatible with your code
- Can use different database type for local dev as production if licensing issues

Tips and Tricks: Bootstrapping

Greenfield Project: **Easy!**

Create an empty changelog file to match your empty database

Tips and Tricks: Bootstrapping

- Starting Liquibase with a new project is easy
  - Just create an empty changelog file and it matches the current state of your database. Done!
- You start building your application and when you need your first database object you create your first changeSet. Nothing to it!

Tips and Tricks: Bootstrapping

Existing Project:

You Have Options

Option 1: Make it look like you've always been using Liquibase

Tips and Tricks: Bootstrapping

- If you have an existing project with an existing database, things are more complicated.
  - Often times a good idea is to find a brand new project as a trial run to get used to Liquibase and better understand it and then apply it to existing projects.
- Unfortunately there is no single “this is how you do it” for existing projects because there are so much variation in projects, processes, and requirements
  - Liquibase provides many tools to help but it is up to you to decide how to string them together

Tips and Tricks: Bootstrapping

- I usually suggest making it look like you've been using Liquibase since the beginning.
  - This can be more work up front, but makes things simpler once it is all set up
- You need to have a changelog file that recreates the database as it is now
  - You may have different databases in different states, need to determine what "now" is
  - "Liquibase generateChangeLog" command is your friend
- Also need to make sure the already ran changeSets are marked as ran
  - Multiple options depending on your project size, environments, etc
    - changeLogSync, changeLogSyncSQL
    - Preconditions
    - Separate changeLog that is exectuted conditionally
  - If you have multiple databases in different states, you may need a mixture
  - If you have multiple databases in different states, you may also need to do cleanup to standardize them across the board.

Tips and Tricks: Bootstrapping

Existing Project:

You Have Options

Option 2: We are going to use Liquibase starting…NOW!

Tips and Tricks: Bootstrapping

- Alternatively, you can just say “From now on we are using Liquibase”
  - Advantage to this is that it is much easier to set up--you just mandate it.
  - No generating changeLogs, no changeLogSync
  - Still need to determine what “now” is
- Disadvantage: you cannot bootstrap your database with only Liquibase for new developers, new customers, etc.
  - Easy workaround is to use standard backup tool to create your starting seed database
- Still want to standardize database variations with preconditions etc. going forward

Tips and Tricks: Bootstrapping

People and Processes

Tips and Tricks: Bootstrapping

- Starting to use Liquibase isn't just about you your bootstrap your change log file
  - Also a question of how you introduce it into your existing process and culture
- For many companies, not a problem
  - Know they need to fix process
  - Everyone is on board with advantages
- For others, there can be entrenched interests and strong resistance
  - Similar to any other processes changes
  - Introduce slowly

Tips and Tricks: Bootstrapping

- Your company has existing processes and workflows that are there for a reason and you want to make sure introducing Liquibase doesn't jeopradise them.
- Liquibase has features to fit into many workflows
  - updateSQL
    - Allow DBAs and others to review and document changes
      - Can even update if needed
    - Works when you don't have direct access to production
  - diff & diffChangeLog
    - If developers want to update database directly, can use to watch for changes and pull change sets out
    - Start with someone's job. Then push to developers to use. Then have them just create them in the first place
    - Remember: Diff logic is not primary and so doesn't catch everything
      - Don't slide into a one-step process
  - Formatted SQL
    - For those who prefer SQL
    - Miss out on some features many people prefer
      - Lose cross-database support, dbdoc functions, auto-rollback generation

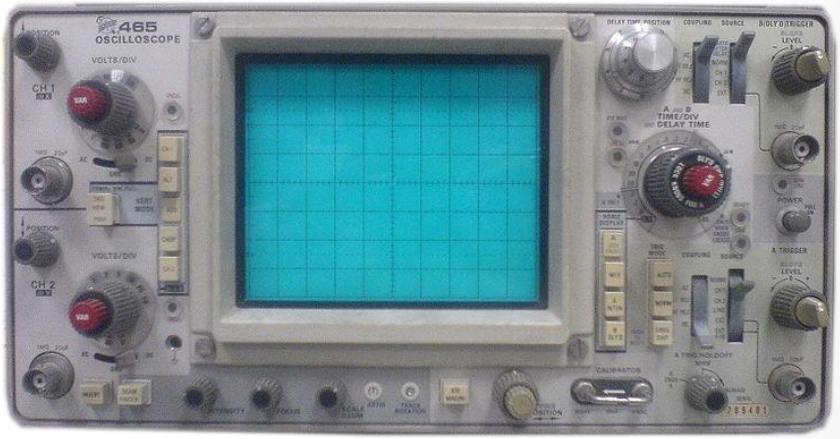
Rollback

Tips and Tricks

- Liquibase provides a way to manage rollback logic with update logic
  - Use during development
    - spelled something wrong or shouldn't have created a table
  - Production
    - OMG!!!!!1111!!!11!!!!
- Can roll back a set number of steps, to a given date, or to a tag
- How does Liquibase roll back changes?
  - The transaction is already done.
  - XML/YAML/JSON know how to rollback create* and rename* changes
  - Others and raw SQL need <rollback> tag
- Additional modes that can be helpful:
  - `rollbackSQL`
  - `futureRollbackSQL`

Tips and Tricks

Manage Test Data



Tips and Tricks

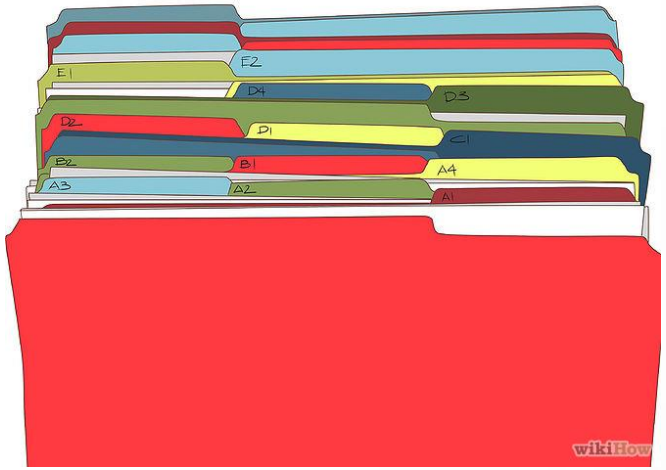
- How to best support your test data is a perennial problem.
  - You have test data for unit tests and integration tests that you keep in backups or SQL load scripts or CSV or excel
  - Take a long time to build up but and is very brittle
- Liquibase can reduce brittleness by loading test data within your changelog file
  - Loaded in at the point it matches the schema, then migrated just like production data would be
- Liquibase supports a <loadData> tag that takes a CSV file OR you can create a standard database dump snapshot and load it in
- Contexts can be used to control which data goes into what test environments
- Test data will also be managed in version control and work with branches and merging
  - as long as you don't use a binary format

Manage Stored Procedures

Tips and Tricks

- Stored procedures, functions and triggers are more like code artifacts than database objects because they don't have state to preserve
  - They also tend to be very large and changes are made to just a small portion of the code at a time
- Therefore, it often makes sense to break the normal "don't touch already ran changeSets" rule in favor of defining them as "create or replace" and use runOnChange
  - Allows you to modify the SQL in place and Liquibase will reexecute it if and only if it has changed
  - Keeps changelog files smaller but most importantly it works nicer with version control for history tracking

Organize Your Change Logs



Tips and Tricks

- A huge monolithic change log gets difficult to work with
- Break up by having a root changelog that <import>s other change logs
  - Can break up by version or feature or anything else
- Liquibase flattens the nested changelogs at run time, so make sure you are only appending to change log OR changes in change logs are independent
  - Otherwise run into ordering problem listed before

Tips and Tricks

DATACAL

- Enterprise Liquibase
- Graphical UI with Change set wizard
- Deployment plan wizard
- Forecasting
- Diff additional types
- Detailed reporting
- Build automation integration

Tips and Tricks

- If you are looking for functionality beyond what comes with Liquibase, check out Datical at datical.com
  - If liquibase is version control for your database, Datical is devops for your database
- Graphical UI
  - Designed to be easy to use. Gives you the right-click refactor menu instead of dealing with raw XML
- Deployment Plan Wizard
  - models and manages your logical deployment workflow
- Forecasting
  - Pre-deployment Analysis
  - Simulate unexecuted change sets before deploying to make sure you won't encounter errors or inadvertently truncate, modify or delete data
- Support for Additional Types
  - Adds support for Stored Procedures, Packages, Functions, Triggers, Check Constraints and more.
- Detailed Reporting
  - Historical reporting for all Forecast & Deployments keeps your entire organization informed about what is happening
  - Easy status on whether or not your individual environments are up to date.
- Build automation integration
  - Integrations with build automation tools: UrbanCode Deploy, Bladelogic, Jenkins, Puppet

Q & A

nathan@liquibase.org

@nvoxland

---

liquibase.org

datical.com