**Liquibase**

* It is open source database independent library for tracking, managing and applying database schema changes.
* No version control available for database changes. For application changes we have the version control normally and easily. That is the reason we have liquibase solution.
* It basically acts as a version control for our DB scripts.
* It supports the usual branching and merging concepts.
* Support XML, YAML, JSON and SQL format.
* Liquibase is going to convert the XML format into SQL scripts and it will be executed against the DB.
* Support content dependent logic – execution is done only against specific version.
* Embedded in your application or can be integrated with ant/maven/gradle build tools.
* The generated SQL scripts can be reviewed.

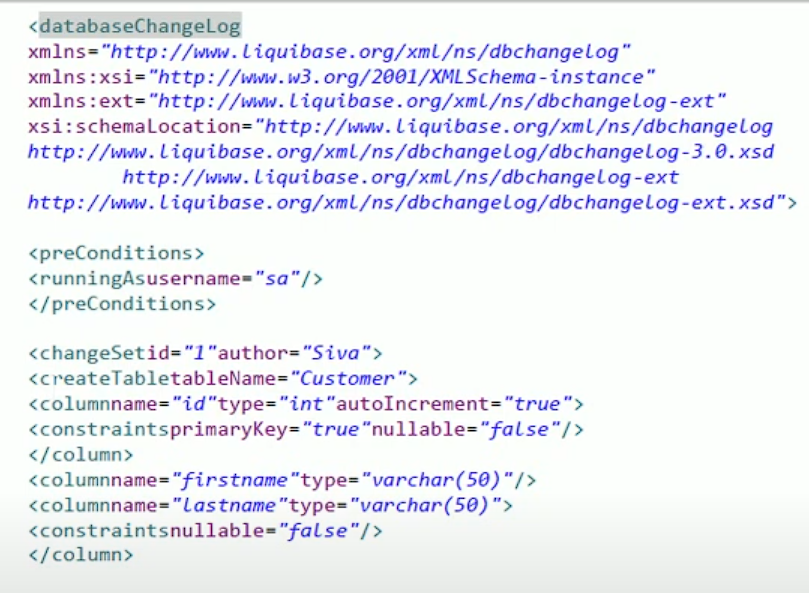
Commercial version of liquibase is called Datical. Datical DB features available– maintain versions, rollback, execute stored procedures, deployment simulation and error reporting, compliance enforcement, automated reporting.

* Liquibase works on root element – DB changelog file (format - XML, YAML, JSON or SQL)



* The changeLog file contains the set of changeSet. ChangeSet is the smallest unit, the fundamental block of change which has to be carried out in the DB.

Sample changelog file



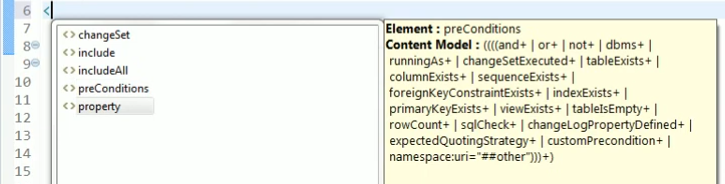
* changeSet takes two parameters, which is ‘id’ and ‘author’
* Type of DB – In the build.gradle file or gradle.properties file in case of Maven Pom.xml or in properties file. We will specify the type of DB, the JDBC driver, DB URL n password.
* Based on these parameters it will identify the corresponding DB. You can also config multiple DBs.
* For each changeSet a corresponding SQL statement is prepared and executed against the DB.

DDL and DML

Data Definition Language commands are used to define and manage the structure of database objects. They deal with the schema of the database, rather than the data itself.

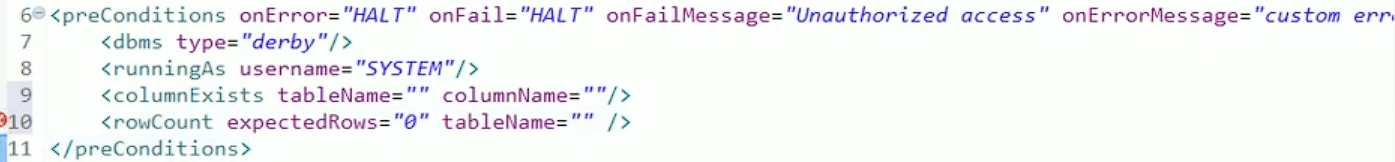
Data Manipulation Language commands are used to manage and manipulate the data within the database objects defined by DDL. They interact with the actual records stored in tables.

databaseChangeLog has the following ‘Elements’

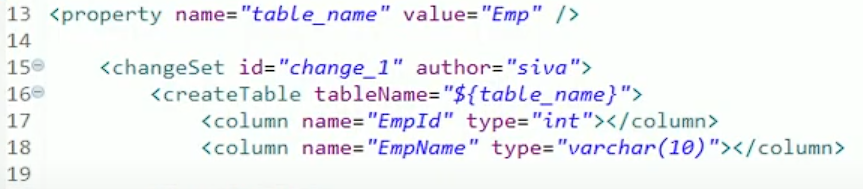


**preConditions** – Info on what basis/assumptions the changeLog has been created. It will impose these assumptions, basically conditions and it there is any violations it will stop processing the task in action.

Eg:

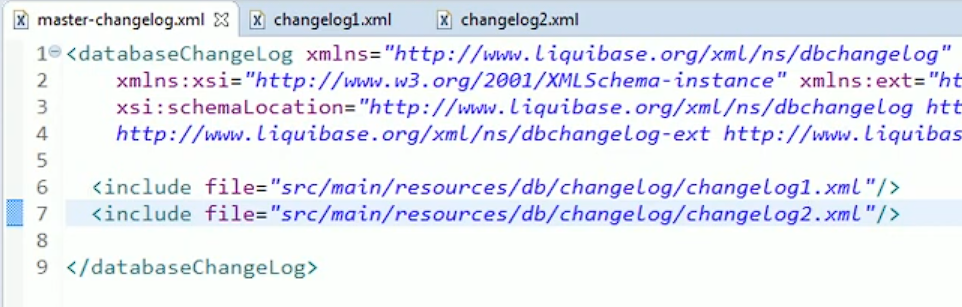


**property** – It is like, you can define a global name or an alias name and you can use it in subsequent changeSet



**include** – As the project gets bigger and bigger, maintaining all the changeSets in single file is going to be difficult. You can have multiple databasechangelog files and include them at the root in the master changelog file.

For a module you can create one master changelog file at root level and then include all these sub database changelog files into it.



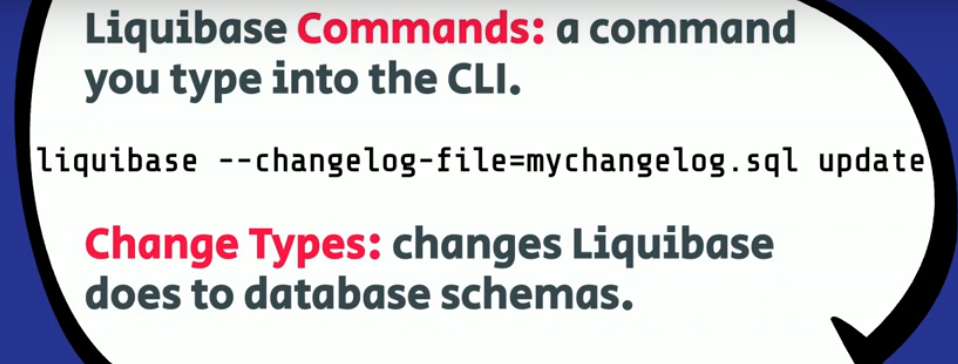
OCT 28th

For Liquibase support questions, visit the Liquibase Forum at <https://forum.liquibase.org>.

<https://docs.liquibase.com/>

<https://www.youtube.com/channel/UC5qMsRjObu685rTBq0PJX8w>

Liquibase also includes features such as rollback functionality and database drift detection.

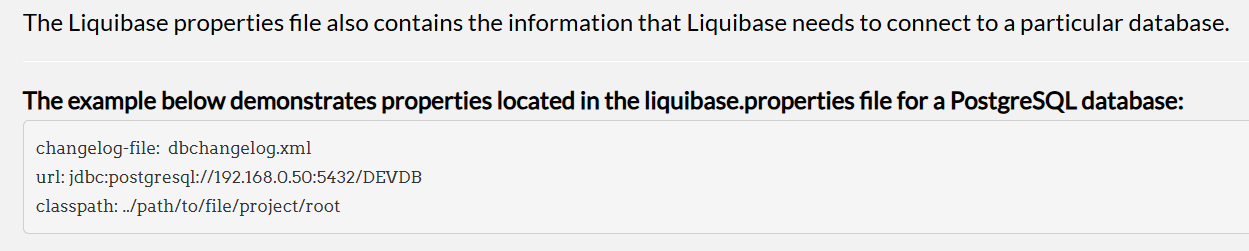


Tracking Tables – These are used to track, version and deploy DB schema changes. Liquibase tracks the deployment transactions in this tracking table.

Two type of tables: 1) DATABASECHANGELOG 2) DATABASECHANGELOGLOCK

* Liquibase automatically creates both tables when the first changelog is executed.
* **DATABASECHANGELOG** tracks each ‘changeset’ in your ‘changelog’ by ID, author and the file where the changeset resides. Tracks each successfully deployed changeset as a single row identified by a combination of changeset id, author, and the filename specified in the changelog.
* Liquibase compares the changelog against the tracking table to determine which changesets need to run. (the change is a combination of ID, author and file path)
* **DATABASECHANGELOGLOCK** locks others out so that they cant make updates to the same database schema at the same time.
* If Liquibase does not exit cleanly, the row may be left locked. The lock can be cleared by running the command liquibase releaseLocks which runs UPDATE DATABASECHANGELOGLOCK SET LOCKED=0
* With Liquibase you can migrate DB states to other databases with ease.
* Generally, there should be only one change per changeset to avoid failed auto-commit statements that can leave the database in an unexpected state.

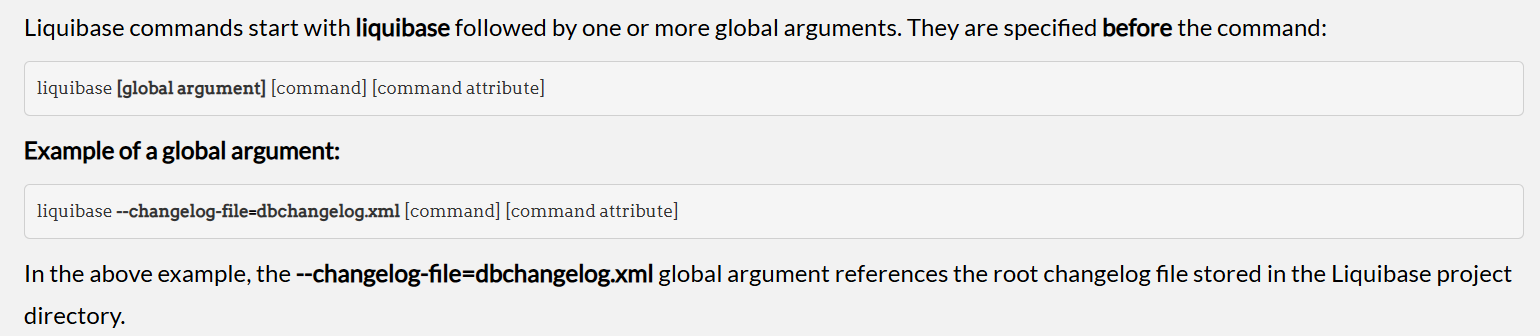
**Change Types** 🡪 They describe a type of change or action to be executed against a database.

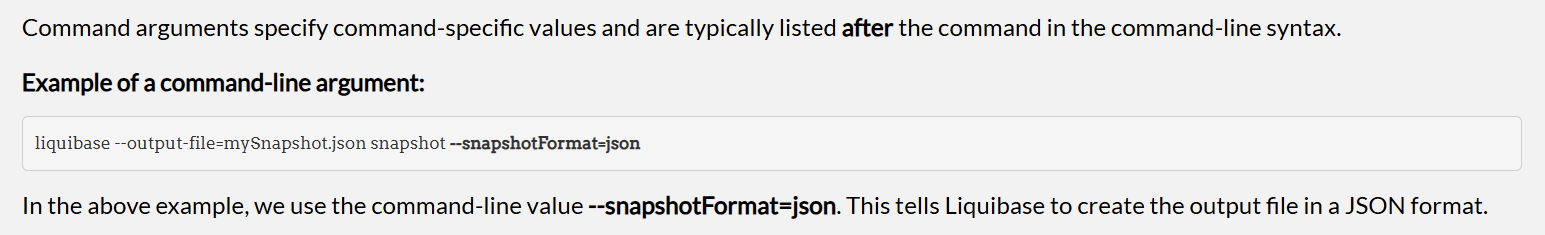


* The values in the Liquibase properties file must be specific to the database that it describes. The classpath can be set to the path where the Liquibase project is running or to the location of the JDBC driver.
* Values specified on the command line will always override the values in the Liquibase properties file.
* While it is allowed for **user authentication** information to be stored in the Liquibase properties file, it is strongly recommended that user credentials are stored in a **secure** credential repository or entered in the CLI when running a command.

**Liquibase Commands**

Liquibase command parameters include command values, global attributes, and command-line attributes.

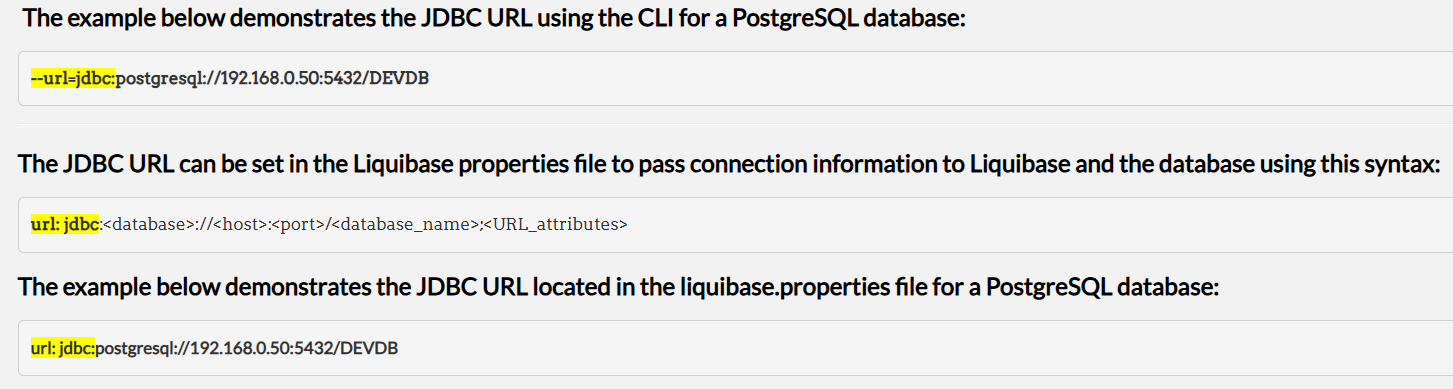
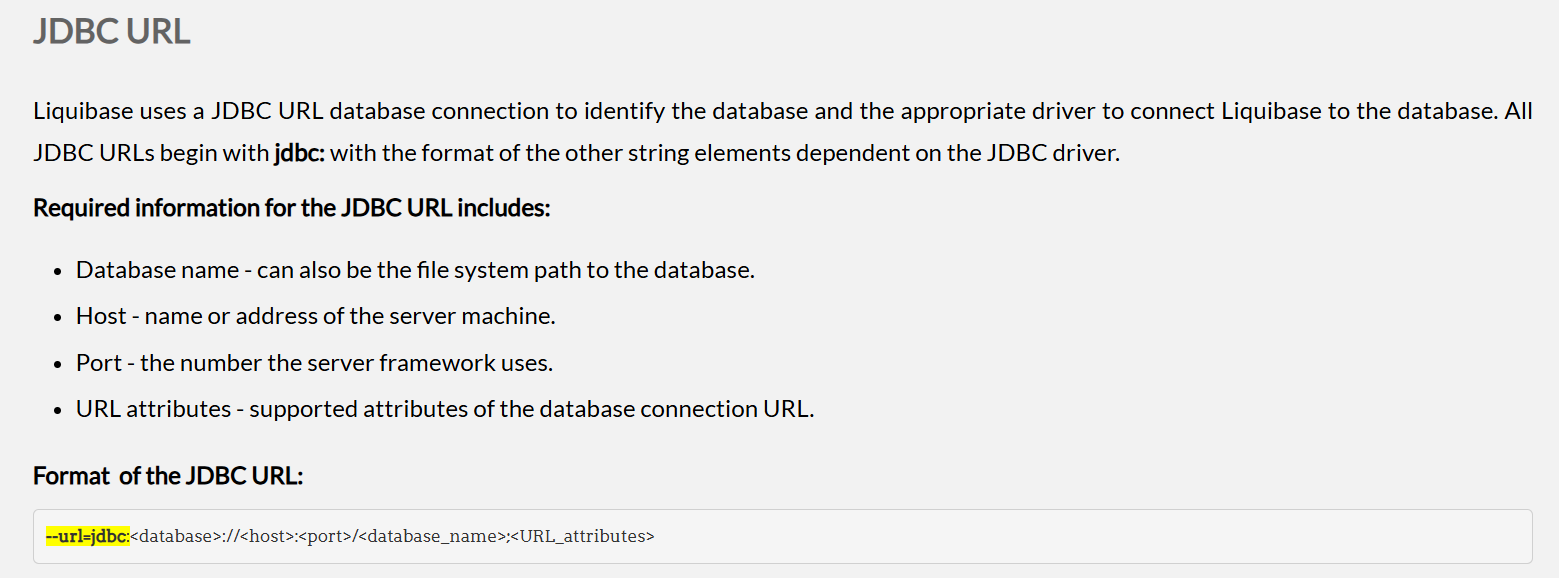




Liquibase will validate if the specified global or command-line argument is allowed for the command being executed. If the command argument is incorrect, or not allowed, the user will receive the following error message: ***Unexpected argument(s):***

**Connecting to the Database**

* Liquibase is built on top of standard **JDBC** to connect to a database management system (DBMS). Liquibase can connect to a DBMS as long as a JDBC driver exists for that DBMS.
* To **establish a connection** between Liquibase and the database, the JDBC drivers must be stored as a **.jar file in the liquibase/lib** folder.



Quiz question

Liquibase needs to find the changelogs and JAR files to operate.From the available answers, where does Liquibase look for those files?

Answer:

Any paths specified in the classpath setting.

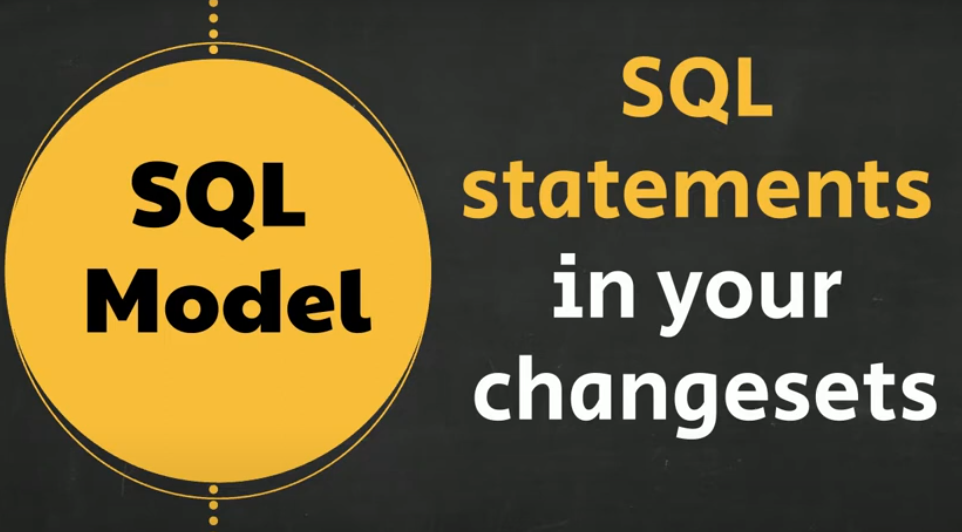
The current working directory.

Inside any .zip or .jar files in the liquibase\_libs or lib directories.

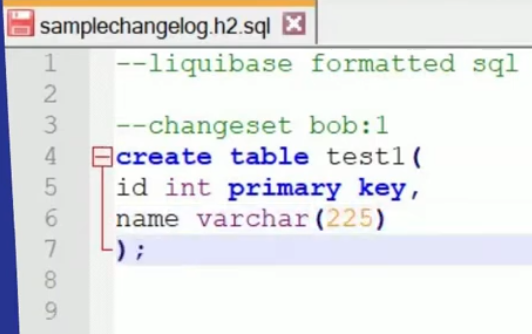
. Liquibase finds files by looking in all these locations plus a liquibase\_libs directory in the current working directory and a lib directory in the Liquibase install location.

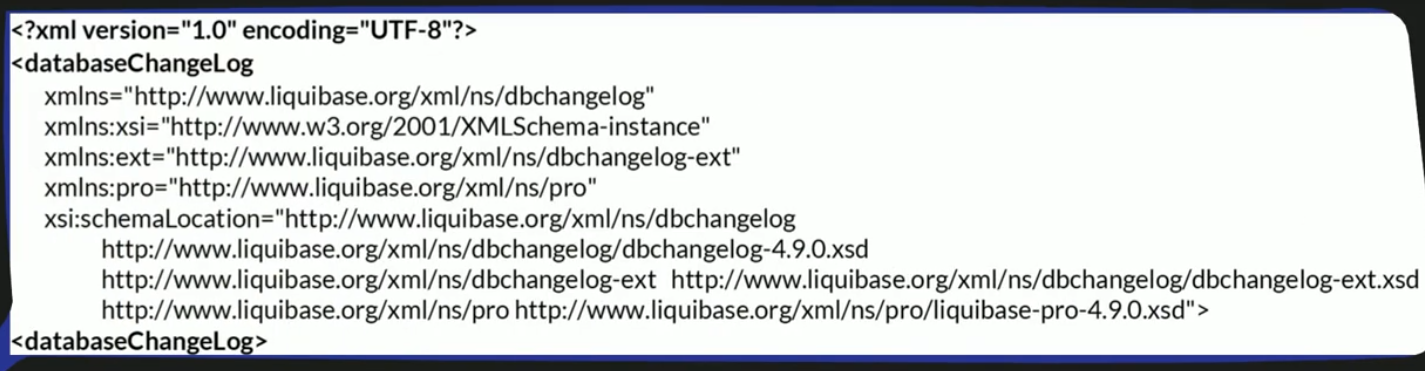
Liquibase is used in two models generally

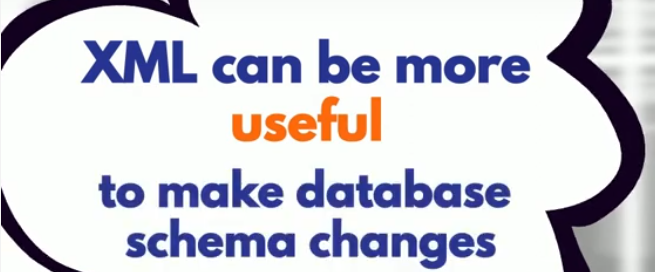
1. SQL model – written in SQL
2. Platform Agnostic model – written in XML, YAML or JSON.

Sample changelogs – SQL & XML versions







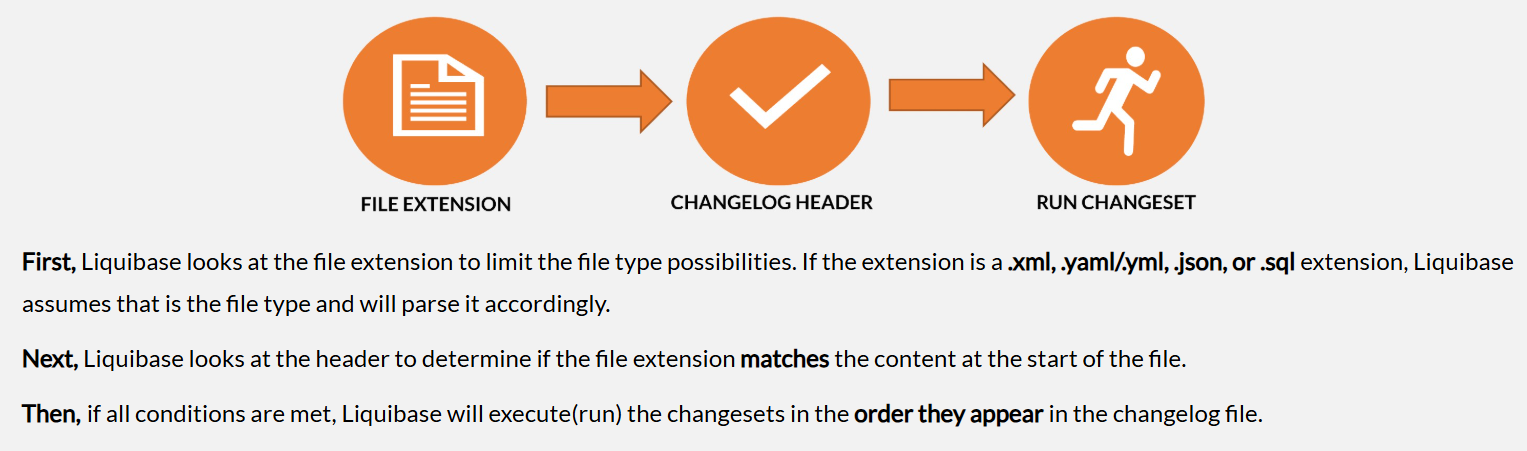
Especially when those changes are between diff types of databases, such as going from an Oracle DB to a postgreSQL DB

To add more ‘changesets’ to your XML changelogs, add them with a new changeset tag.

\*When non-SQL formats are used, Liquibase will generate the database-specific SQL syntax for the different databases.

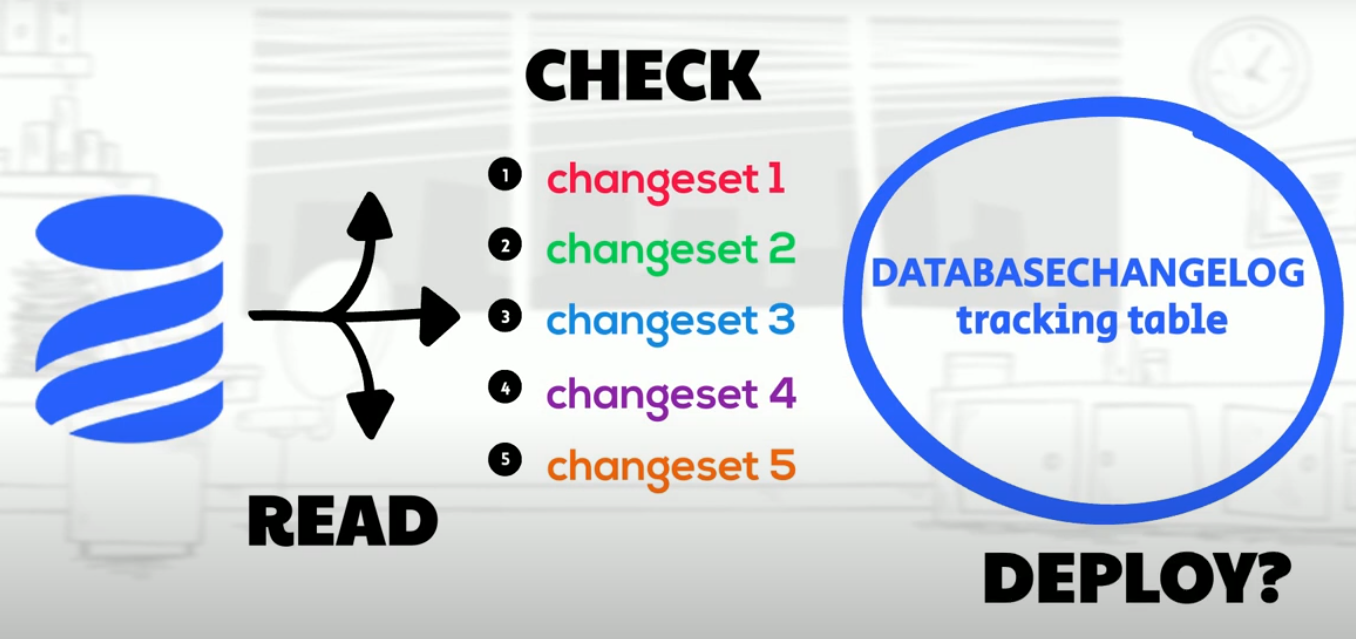
Components of changelog

Changelogs can include a number of components based on user specifics but there are three components that are essential for every changelog: The **changelog header**, **changesets**, and **changeset attributes**.



Changeset attributes are a way to provide information that determines **how** a changeset is identified and processed. Attributes provide a unique way to differentiate a changeset, and every changeset in Liquibase requires **unique identifiers**.

Changesets



The ID is just used as identifier it does not determine the order in which the changes are run.

Author tag – to identify who wrote the changesets.

Generally one change type per changeset is recommended, but that is not possible always. When you are having multiple change types in a changeset make sure to test it completely.

