* **DB Migration Overview**
  + **Issues writing applications that persist data to a RDBMS**
    - What is current state of database?
    - Has db already been updated with appropriate sql scripts
    - If db doesn't exist, how do you create the new from scratch?
    - Is the software compatible with the current state of the database?
    - There are multiple dbs that need to be kept in sync. Causes issues when db needs to be modified.
    - These issues become even more prevalent when dealing with multiple environments on a project
  + **Meh Solutions**
    - Manually write sql statements on each machine
    - You can create .sql scripts, then send to developers, testers, etc. and have them execute scripts on their machine
    - But even if this works initially, over time it can become difficult to know which versions are applied? And if not updated, in what order should your scripts be updated.
  + **Database migration tools**
    - Assist in managing a database across multiple environments (development, testing, production, etc.)
    - Recreating database from scratch
    - Ensures that your software will run with the current state of database
    - The state of a database is clear at all times
    - Review changes to database
* **Flyway**
  + **Overview**
    - Flyway tackles these problems in a reliable way.
    - Deployed with application, so there is no concern with state of db.
    - Written in java, but can be integrated as a jar, standalone command line client, or plugin
  + **How Flyway Works?**
    - Flyway creates a meta table (bookkeeping table) to keep track of what's been applied and whether or not it was successful
    - At runtime (or whenever migrate is invoked) is able to look at all available migrations, and compare them with the flyway metadata table to determine which migrations need to be performed (if any).
      * *Does this by scanning file system or classpath (based on location you specified), sorted by version number, then migrates in order.*
    - If a migration has already been run, it will be skipped
    - Entry point is usually in SQL (but can also be written in java.. per flyway docs, raw sql should be first choice)
* **Notable Configuration / Options**
  + **flyway.schemas**
    - Comma separated list of schemas to manage
    - It will clean all schemas in the order they are placed in the list
    - Only migrates first item in list (?)
  + **Flyway.table**
    - Name of the table inside your db which stores the migration/versioning info
  + **Flyway.locations**
    - Comma separated list that specifies location of migrations
      * Filesystem: - looks for path on file system, must be sql files
      * If location is unprefixed or begins with classpath: it will look on the applications classpath (can be either sql or java files)
    - *Flyway.properties Example:*
      * *flyway.locations*=filesystem:C:/Users/lfallon/Desktop/Flyway/flyway-cmd-3.1/sql
      * Spring Config Example where the folder **migrations** is directly on the classpath

<bean id="flyway" class="org.flywaydb.core.Flyway" init-method="migrate">

**<property name="locations" value="migrations/" />**

<property name="dataSource" ref="dataSource" />

<property name="cleanOnValidationError" value="true"></property>

</bean>

* + **Flyway SQL Migration Naming**
    - Naming Structure: **prefixVERSIONseparatorDESCRIPTIONsuffix**
      * **Example V1\_3\_\_AddedDepartmentTable.sql**
      * Prefix: V
      * Version: 1\_3
      * Separator: \_\_
      * Description: AddedDepartmentTable
      * Suffix: .sql
  + **Flyway Placeholders**
    - TODO
  + **Flyway Baseline Options**
    - Enables user to tag an existing database with a “baseline version” that specifies what versions migrations should begin at.
    - Used in cases where you haven an existing db without a metadata schema
    - i.e., if you have a non-empty database without a metadata schema, and you know this db is fully up to date with v2.4 of your migrations, you can set the baselineVersion to 2.4. When you run baseline, a metadata table will be created in your db beginning with version 2.4 (and a type of BASELINE). Now when flyway migrate is executed, it will begin executing the .sql / java files starting at version 2.5
  + **Flyway Validate Options**
    - Flyway.cleanOnValidateError: Specifies whether or not to clear db if an error results from executing validate command
    - This should never be enabled in production (per the Flyway docs this is solely intended as a convenience for developers). The flyway migrations are managing db structure. The only values in a db migration would be those that are static/readonly tables/fields. Because of this, the production db will have many records that your dev db does not. And wiping clean would permanently erase that data.
* **Commands**
  + **Info-** Returns a summary of all applied, failed, or pending migrations, by comparing the databases’s metadata table to the locations directory of your scripts.
  + **Migrate-** Runs a migration
  + **Repair-** If a migration failed, this is flyway’s attempt to revert your database to the previous valid version (note: that this operation is not guaranteed to work 100%. There may need to be addl work)
  + **Clean-** Wipes your db clean. Very useful in development. Obviously, never use in production.
  + **Validate-** Validates the current migrations to available ones
  + **Baseline-** If a database already exists and you know that it’s current up to a current version, you can set a “baseline” in your metadata table that basically says “Hey, I’m good up to version 2.3, so when you migrate start at 2.4”

* **Configuration Steps / Execution Modes**
  + Command Line
    - Navigate to flyway.bat file in terminal, and execute **flyway *<<command\_name>> <<options>>***
  + Java Spring

<bean id="flyway" class="org.flywaydb.core.Flyway" init-method="migrate">

<property name="locations" value="migrations/" />

<property name="dataSource" ref="dataSource" />

<property name="cleanOnValidationError" value="true"></property>

</bean>

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource" depends-on="flyway">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://127.0.0.1:3306/my\_database" />

<property name="username" value="user123" />

<property name="password" value="pass123" />

</bean>

<dependency>

<groupId>org.flywaydb</groupId>

<artifactId>flyway-core</artifactId>

<version>3.0</version>

</dependency>

* + Java
    - Show in flywaydemo-spring
  + Maven
    - Show in flywaydemo-spring

**Addl Notes**

- The version of your JDBC driver can effect migrations due to sql syntax. Be sure that the versions in your development are same as in production

- Appears as though for some db’s, a schema must already exist (MySql), and others it will be automatically created (H2)

- Flyway 3.0 has error if you set the baseline, then attempt to migrate from there (upgrading to 3.1 fixed this issue)

- **Testing Scenario**: Place your sql scripts in two locations. One location will have only your db structure (your dev db), while the other (your db for testing env) can have the same scripts but with DML statements to populate with test data

- **Liquibase**: offers an abstraction, using xml files at runtime it translates the xml files into the appropriate sql statements based on your db. But many (or most) projects will be dealing with the same RDBMS, so if you can live without that feature, flyway's simplicity may be the better choice.

- **Backwards Migration**: not really supported (though it will wrap entire sql script in transaction, and rollback if a migration fails). Destructive changes cause problems with "reverting". (NOTE: Can write migrations in a backward compatible way, with the last version in production you would like to roll back to.)