**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **Particulars** | **Page No** |
|  | Introduction | 2 |
| 1 | How Flyway DB Works | 3 |
| 2 | Configuration of Flyway DB | 9 |
| 3 | Flyway DB for Fresh and Existing DB |  |
|  | 3.1 Empty Database | 11 |
|  | 3.2 For Existing DB | 12 |

**Introduction**

This documentation describes key concepts of [**Flyway**](https://flywaydb.org/) and how we can use this framework to continuously remodel our application’s database schema reliably and easily.

Flyway updates a database from one version to a next using migrations. We can write migrations either in SQL with database specific syntax or in Java for advanced database transformations.

**1. How FLYWAY Works**

With Flyway all changes to the database are called **migrations.** To keep track of which migrations have already been applied, when and by whom, it adds a special bookkeeping table to your schema. This metadata table also tracks migration checksums and whether or not the migrations were successful.

The framework performs the following steps to accommodate evolving database schemas:

1. It checks a database schema to locate its metadata table (*SCHEMA\_VERSION* by default). If the metadata table does not exist, it will create one
2. It scans an application classpath for available migrations
3. It compares migrations against the metadata table. If a version number is lower or equal to a version marked as current, it is ignored
4. It marks any remaining migrations as pending migrations. These are sorted based on version number and are executed in order
5. As each migration is applied, the metadata table is updated accordingly

**Schema History Table**

To keep track of which migrations have already been applied when and by whom, Flyway adds a special **schema history table** to your schema. You can think of this table as a complete audit trail of all changes performed against the schema. It also tracks migration checksums and whether or not the migrations were successful.

**Commands used in FlywayDB**

| **Name** | **Description** |
| --- | --- |
| [migrate](https://flywaydb.org/documentation/commandline/migrate) | Migrates the database |
| [clean](https://flywaydb.org/documentation/commandline/clean) | Drops all objects in the configured schemas |
| [info](https://flywaydb.org/documentation/commandline/info) | Prints the details and status information about all the migrations |
| [validate](https://flywaydb.org/documentation/commandline/validate) | Validates the applied migrations against the ones available on the classpath |
| [baseline](https://flywaydb.org/documentation/commandline/baseline) | Baselines an existing database, excluding all migrations up to and including baselineVersion |
| [repair](https://flywaydb.org/documentation/commandline/repair) | Repairs the schema history table |

**Versioned migrations**

The most common type of migration is a **versioned migration**. Each versioned migration has a *version*, a *description* and a *checksum*. The version must be unique. The description is purely informative for you to be able to remember what each migration does. The checksum is there to detect accidental changes. Versioned migrations are applied in order exactly once.

Versioned migrations are typically used for:

* Creating/altering/dropping tables/indexes/foreign keys/enums/UDTs/…
* Reference data updates
* User data corrections

Here is a small example:

CREATE TABLE EMPLOYEE (

Eid INT NOT NULL PRIMARY KEY,

Ename VARCHAR NOT NULL,

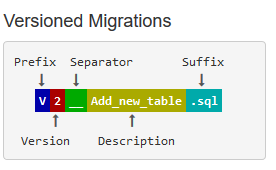
Designation VARCHAR NOT NULL

);

ALTER TABLE EMPLOYEE ADD Address VARCHAR;

INSERT INTO EMPLOYEE VALUES ('XXXX');

Versioned migrations are applied in the order of their versions. Versions are sorted numerically as you would normally expect.



The file name consists of the following parts:

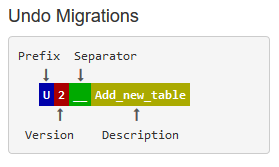
* **Prefix**: V for versioned, U for undo and R for repeatable migrations.
* **Version**: Version with dots or underscores separate as many parts as you like (Not for repeatable migrations)
* **Separator**: \_\_ (two underscores)
* **Description**: Underscores or spaces separate the words
* **Suffix**: .sql

Each versioned migration must be assigned a **unique version**. Any version is valid as long as it conforms to the usual dotted notation. For most cases a simple increasing integer should be all you need. However Flyway is quite flexible and all these versions are valid versioned migration versions:

* 1
* 001
* 5.2
* 1.2.3.4.5.6.7.8.9
* 205.68
* 20130115113556
* 2013.1.15.11.35.56
* 2013.01.15.11.35.56

Versioned migrations are applied in the order of their versions. Versions are sorted numerically as you would normally expect.

**Undo migrations** are the opposite of regular versioned migrations. An undo migration is responsible for undoing the effects of the versioned migration with the same version. Undo migrations are optional and not required to run regular versioned migrations.



**Repeatable migrations** are very useful for managing database objects whose definition can then simply be maintained in a single file in version control. Instead of being run just once, they are (re-)applied every time their checksum changes.

They are typically used for

* (Re-)creating views/procedures/functions/packages/…
* Bulk reference data reinserts

Now let’s create a repeatable migration to manage a view of the person table. With Flyway’s default naming convention, the filename will be similar to the regular migrations, except for the V prefix which is now replaced with a R and the lack of a version.

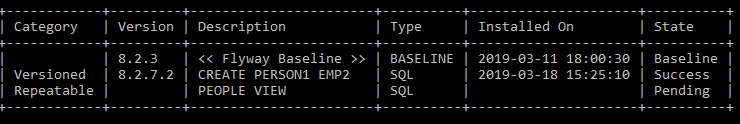
So go ahead and create R\_\_PEOPLE\_VIEW.sql in the /sql directory:

CREATE OR REPLACE VIEW people AS

SELECT id, name FROM person1;

This is now the status

flyway-5.2.4> flyway **info**



Note the new pending repeatable migration.

## Executing the migration

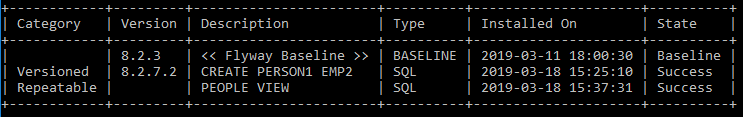
It’s time to execute our new migration.

So go ahead and invoke

flyway-5.2.4> flyway **migrate**

And you can check that this is indeed the new status:

flyway-5.2.4> flyway **info**



As expected we can see that the repeatable migration was applied successfully.

## Modifying the migration

Now let’s see what happens when we modify our migration file in place.

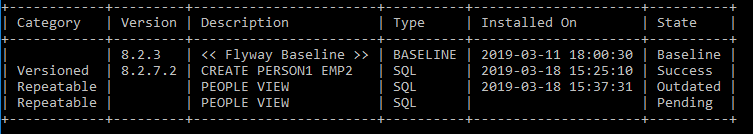
Update R\_\_PEOPLE\_VIEW.sql in the /sql directory as follows:

CREATE OR REPLACE VIEW people AS

SELECT id, name FROM person1 WHERE name like 'M%';

And check the status again:

flyway-5.2.4> flyway **info**

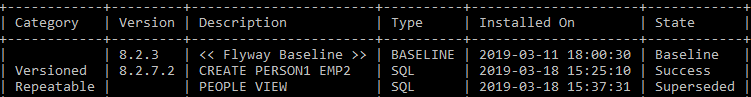


Our audit trail now clearly shows that the repeatable migration that was previously applied has become outdated and is now marked as pending again, ready to be reapplied.

So let’s do exactly that:

flyway-5.2.4> flyway **migrate**

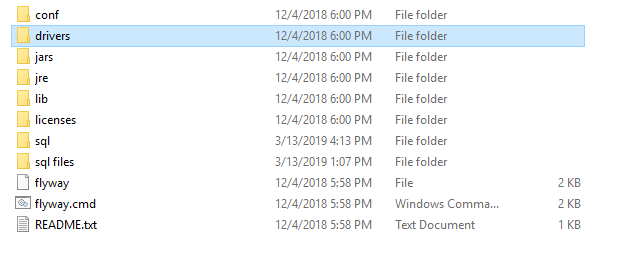
And the status is now



Our initial run has now been superseded by the one we just did. And so whenever the object you are managing (the people view in our example) needs to change, simply update the file in place and run migrate again

**2. Installing and Configuring FlywayDB**

Un-tar flywayDB<version>.zip, we will be provided with below folder structure:



**Conf:**

Configuration file (flyway.conf) of conf folder, we need to specify the below properties,

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | flyway.user=databaseUser  flyway.password=databasePassword  flyway.schemas=app-db  flyway.url= jdbc:db2://<host>:<port>/<database>  flyway.locations=filesystem:db/migration | |
| 2 |
| 3 |
| 4 |
| 5 |
|  |  |  |

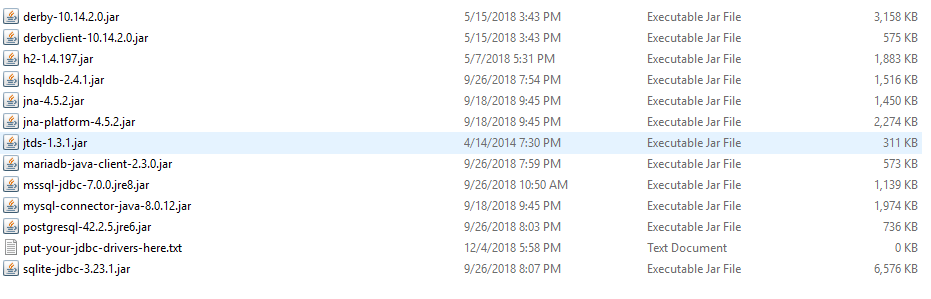
The above configuration specifies that our migration scripts are located in the ***db/migration*** directory, by default it will be stored in sql directory of flywaydb directory. It connects to DB2 instance using ***databaseUser*** and ***databasePassword*.**

The application database schema is *app-db*. Please replace *flyway.user, flyway.password, flyway.url* with your database username, database password and database host/port appropriately.

**Drivers:**

In order to connect with your database, Flyway needs the appropriate JDBC driver to be available in its drivers directory. By default below are the drivers which are present

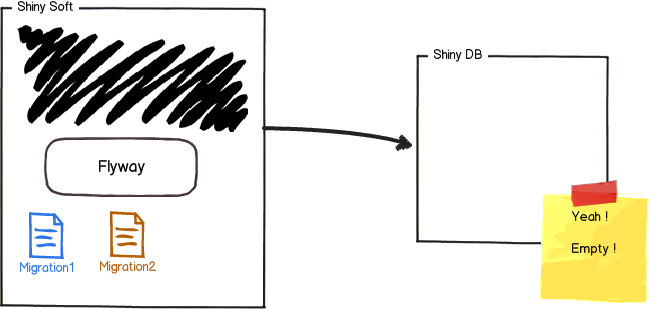
If your database is not listed here, you need to download its JDBC driver and place it in the *drivers* directory yourself. Instructions on where to download the drivers from are provided on the respective documentation pages for each database.



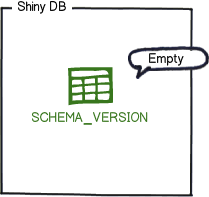
**3. Flyway DB for Fresh and Existing DB**

**3.1 Empty DATABASE**

The easiest scenario is when you point **Flyway** to an **empty database**.



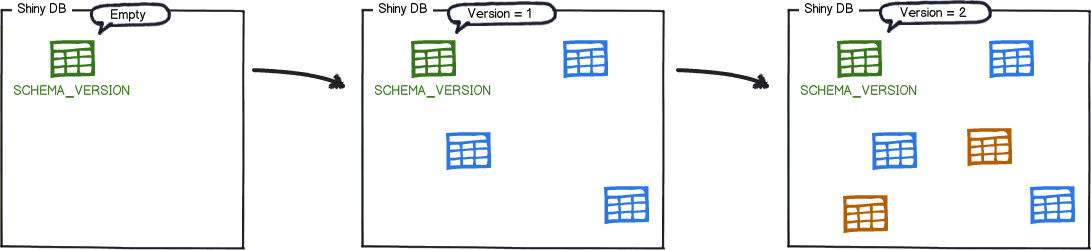
It will try to locate its **schema history table**. As the database is empty, Flyway won't find it and will **create** it instead.  
  
You now have a database with a single empty table called *flyway\_schema\_history* by default:



This table will be used to track the state of the database.

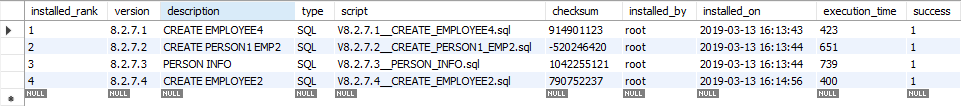
Immediately afterwards Flyway will begin **scanning** the file system or the classpath of the application for **migrations**. They can be written in either sql or Java.

The migrations are then **sorted** based on their **version number** and applied in order:



As each migration gets applied, the schema history table is updated accordingly:

##### flyway\_schema\_history

****

Every time the need to evolve the database arises, whether structure (DDL) or reference data (DML), simply create a new migration with a version number higher than the current one. The next time Flyway starts, it will find it and upgrade the database accordingly

**3.2 FOR EXISTING DB**

**Proposed versioning strategy**

* Versioning will be done using versioned migration.
* If the changes to be done to DB in 827 then the DDL files should be named or versioned as V\_8.2.7.1\_\_<description\_of\_the\_file>, V\_8.2.7.2\_\_<description\_of\_the\_file>, etc.
* The migration (change to DB) will be done in the same order.

Eg: If u want to create a table EMP in a file and insertion of values to the same table EMP in other sql file, then the sql files should be versioned as:

V\_8.2.7.1\_\_creation\_of\_EMP\_table.sql

V\_8.2.7.2\_\_insertion\_of\_values\_to\_EMP\_table.sql

The above two files will be migrated in the same order as we have maintained. So it is the responsible of DB team to maintain the order in which the .sql files has to be migrated and versioned accordingly.

* DB team has to take care of versioning the newly created DDL files (\*.sql files)

**To migrate between versions (Eg. 8.2.1 to 8.2.3)**

* Here we have to identify the minimum version which is released in any environment, and then set that as a baseline i.e. if 821 already been released, set the baseline to 821 and any DB files which are committed to previously released version, should be renamed as <V\_8.2.1\_\_description\_of\_file>.
* Identify the DDL files in 822.
* Rename them in SVN as V\_8.2.2.1\_\_<description\_of\_the\_file>, V\_8.2.2.2\_\_ <description\_of\_the\_file, etc.,
* Ordering with which it should reflect in DB, should be taken care.
* Repeat the above to migrate from 822 to 823.

**Handling DB procedures**

In case of DBprocs, we will be applying Repeatable migrations to the available files as <R\_\_description\_of\_the\_file> and the detailed report on handling repeatable version is in

**To maintain and connect to multiple DB2 databases**

We can connect to any database to flywayDB by setting the *flyway.url* value in conf file. Below is the detail to connect DB2 database to flyway DB:

# DB2\* : jdbc:db2://<host>:<port>/<database>

# \* = JDBC driver must be downloaded and installed in /drivers manually

We can maintain different DB instances in multiple conf files and then we can point these multiple conf files and is detailed below.

It is also possible to point Flyway at one or more additional config files. This is achieved by supplying the command line parameter -configFiles= as follows:



To pass in multiple files, separate their names with commas:

