Flyway Database migrations

START HERE

`flyway` **`init -projectName=`**_<name>_ **`-databaseType=`**_<type>_
New project

`flyway` **`baseline`**
Adopt existing DB

`flyway` **`add -description=`**_<text>_
New migration file

`flyway` **`clean`**
Wipe schemas (dev only)

`flyway` **`info`**
What is applied?

`flyway` **`migrate`**
Apply pending

`flyway` **`validate`**
Detect edited scripts

`flyway` **`repair`**
Fix history table

V1__init.sql files in locations → `flyway migrate` → **Target DB** your schema → `records` → **Schema history** flyway_schema_history → `checked by` → **validate** checksums + order

CONCEPTS

Never edit a migration that has been applied: its checksum is stored and validate will fail. Add a new one instead.

Versioned migrations run once in version order; repeatable ones (R__) rerun whenever their checksum changes, after all versioned ones.

Undo migrations, check -dryrun and check -drift/-changes need paid editions; clean, check -code and the rest here are in Community.

MIGRATION FILES

`V`_<n>_**`__`**_<desc>_**`.sql`**
Versioned migration (runs once)
Two underscores between version and description, e.g. V2__add_users.sql

`R__`_<desc>_**`.sql`**
Repeatable migration

`U`_<n>_**`__`**_<desc>_**`.sql`**
Undo migration for a version
Undo needs a paid edition

`B`_<n>_**`__`**_<desc>_**`.sql`**
Baseline migration

MIGRATION FILES (cont.)

`flyway` **`add -description=`**_<text>_
Create next versioned script

`flyway` **`add -type=undo`**
Create undo script

`flyway` **`add -version=`**_<v>_ **`-description=`**_<text>_
Create script with explicit version

`flyway` **`add -timestamp -description=`**_<text>_
Add timestamp to version

`flyway` **`add -location=`**_<path>_
Choose target directory

`flyway` **`add -add.nameOnly=true -q`**
Print the file name only

`${`_<name>_**`}`**
Placeholder inside SQL

`-locations=filesystem:`_<dir>_
Where to scan for scripts

PROJECT AND SETUP

`flyway` **`init -projectName=`**_<name>_ **`-databaseType=`**_<type>_
Generate project and flyway.toml

PROJECT AND SETUP (cont.)

`flyway` **`init -from=flyway.conf -fromType=Conf`**
Upgrade an old .conf file

`flyway` **`list engines`**
List supported databases

`flyway` **`-v`**
Print version

`flyway` **`-environment=`**_<env>_
Pick the target environment

`flyway` **`-configFiles=`**_<file>_
Use a specific config file

`flyway` **`-url="`**_<jdbc-url>_**`" -user=`**_<u>_ **`-password=`**_<p>_ **`migrate`**
Connect without a config file

`-schemas=`_<a>_**`,`**_<b>_
Schemas Flyway manages

MIGRATE AND INSPECT

`flyway` **`migrate`**
Apply all pending migrations

`flyway` **`info`**
Show applied and pending

`flyway` **`validate`**
Compare files with history

`flyway` **`migrate -target=`**_<version>_
Stop at a version

MIGRATE AND INSPECT (cont.)

`flyway` **`migrate -outOfOrder=true`**
Allow lower versions after higher

`flyway` **`migrate -baselineOnMigrate=true`**
Baseline a non-empty DB on first run

`flyway` **`info -infoOfState=pending`**
Filter by state

`flyway` **`undo`**
Undo the last versioned migration

REPAIR AND RECOVER

`flyway` **`repair`**
Remove failed rows, realign checksums
Use the same locations as migrate; clean up objects a failed script left behind by hand

`flyway` **`baseline`**
Mark existing DB as migrated

`flyway` **`baseline -baselineVersion=`**_<v>_
Baseline at a version

`flyway` **`clean -cleanDisabled=false`**
Drop all objects in schemas
Never against production

REPAIR AND RECOVER (cont.)

After a failed migration run repair, fix the script, then migrate again.

Keep cleanDisabled=true outside dev and test environments.

`flyway` **`check -code`**
Review migrations with code analysis

`flyway` **`check -dryrun`**
Preview a deployment script

CONFIGURATION

`flyway.toml`
Project config file

`[environments.`_<name>_**`] url, user, password`**
Define an environment

`[flyway] locations = ["filesystem:sql"]`
Flyway settings in TOML

`FLYWAY_URL / FLYWAY_USER / FLYWAY_PASSWORD`
Env var overrides

`locations`
Folders scanned recursively

`connectRetries`
Retry connecting to the DB

`[flyway.placeholders] name = "value"`
Define placeholders

CONFIGURATION (cont.)

`beforeMigrate.sql / afterMigrate.sql`
Callback scripts in locations

`-X`
Debug output

`-outputType=json`
Machine-readable output

RECIPES

First migration

```
flyway init -projectName=app -
databaseType=PostgreSql
flyway add -
description=create_users
# edit the new file in sql/
flyway info
flyway migrate
```

Adopt an existing database

```
flyway baseline -
baselineVersion=1 -
environment=prod
# add V2__... and later
flyway migrate -
environment=prod
```

Recover from a failed script

```
flyway info
# fix leftovers by hand, fix
the script
flyway repair
flyway migrate
```

CI deployment

```
flyway validate -environment=ci
flyway migrate -environment=ci
-outputType=json
flyway info -
infoOfState=pending
```