Flyway Database migrations

START HERE

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

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

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

`flyway` **migrate**
Apply pending

`flyway` **validate**
Detect edited scripts

`flyway` **repair**
Fix history table

`flyway` **baseline**
Adopt existing DB

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

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

`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

`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

-workingDirectory=<*dir*>
Run as if in another directory

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

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

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

`flyway` **migrate -validateOnMigrate=false**
Skip validation (avoid)

`flyway` **migrate -mixed=true**
Mix transactional and non-transactional

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

MIGRATE AND INSPECT (cont.)

`flyway` **info -infoSinceVersion=**<*v*>
From a version onwards

`flyway` **info -infoSinceDate="dd/MM/yyyy HH:mm"**
Applied after a date

`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

> 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

`flyway` **check -drift**
Detect out-of-band changes

`flyway` **check -changes**
Report pending changes

CONFIGURATION

flyway.toml
Project config file

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

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

CONFIGURATION (cont.)

FLYWAY_URL / FLYWAY_USER / FLYWAY_PASSWORD
Env var overrides

locations
Folders scanned recursively

table
Name of the history table

defaultSchema
Schema holding the history table

baselineVersion
Version used by baseline

sqlMigrationPrefix / sqlMigrationSeparator
Change naming defaults

connectRetries
Retry connecting to the DB

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

beforeMigrate.sql / afterMigrate.sql
Callback scripts in locations

-X
Debug output

-outputType=json
Machine-readable output

-outputFile=<*file*>
Also write output to a file

-q
Quiet 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
```