Flyway Database migrations

CONCEPTS

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

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

> 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)

R__*<desc>*.sql
Repeatable migration

U*<n>*__*<desc>*.sql
Undo migration for a version

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

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

${*<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` **-v**
Print version

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

PROJECT AND SETUP (cont.)

`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

`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

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

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

`flyway` **clean -cleanDisabled=false**
Drop all objects in schemas

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

REPAIR AND RECOVER (cont.)

> 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

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
```