Custom Queries and Migrations

Hibernate and Flyway DB

Content

Currencies

1. Add Currency Entity => Columns [id, name]
2. Add Transaction <-> Currency Relationship
3. Currency Seeding
4. Add Currency Information To Transaction Endpoints

CrudRepository

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
        <S extends T> S save(S entity);
        <S extends T> Iterable<S> saveAll(Iterable<S> entities);
        Optional<T> findById(ID id);
        boolean existsById(ID id);
        Iterable<T> findAll();
        Iterable<T> findAllById(Iterable<ID> ids);
        long count();
        void deleteById(ID id);
        void delete(T entity);
        void deleteAll(Iterable<? extends T> entities);
        void deleteAll();
}
```

Custom Repository Methods

a. We can start our query method names with find...By, count...By. Before By we can add expression such as Distinct . After By we need to add property names of our entity.

b. To get data on the basis of more than one property we can concatenate property names using And and Or while creating method names.

c. If we want to use completely custom name for our method, we can use @Query annotation to write query.

Custom Repository Methods

```
public interface UserRepository extends CrudRepository<User, Long> {

  // Enables the distinct flag for the query
  List<User> findDistinctPeopleByLastNameOrFirstName(String lastName, String firstName);
  List<User> findPeopleDistinctByLastNameOrFirstName(String lastName, String firstName);

  // Enabling static ORDER BY for a query
  List<User> findByLastNameOrderByFirstNameAsc(String lastName);
  List<User> findByLastNameOrderByFirstNameDesc(String lastName);
}
```

Spring Data JPA @Query

```
public interface UserRepository extends CrudRepository<User, Long> {

  @Query("SELECT u FROM User u WHERE u.status = 1")
  Collection<User> findAllActiveUsers();
}
```

Spring Data JPA @Query Native

```
public interface UserRepository extends CrudRepository<User, Long> {
  @Query(
        value = "SELECT * FROM USERS u WHERE u.status = 1",
        nativeQuery = true)
  Collection<User> findAllActiveUsersNative();
}
```

Database Migrations with Flyway

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**.

Flyway Maven Plugin

```
<plugin>
        <groupId>org.flywaydb</groupId>
        <artifactId>flyway-maven-plugin</artifactId>
        <version>6.0.4</version>
        <configuration>
                <user>root</user>
                <url>jdbc:mysql://localhost:3307/money-transfer-app</url>
                <schemas>
                  <schema>money-transfer-app</schema>
                </schemas>
                <baselineOnMigrate>false</baselineOnMigrate>
       </configuration>
</plugin>
```

Plugin Configuration

```
<configuration>
   <user>root</user>
   <url>jdbc:mysql://localhost:3307/money-transfer-app</url>
   <schemas>
      <schema>money-transfer-app</schema>
   </schemas>
   <baselineOnMigrate>false</baselineOnMigrate>
</configuration>
```

Migration Naming Convention

Format : `<Prefix><Version>__<Description>.sql`

Location: `src/main/resources/db/migration`

Example: `V1_1_0__my_first_migration.sql`

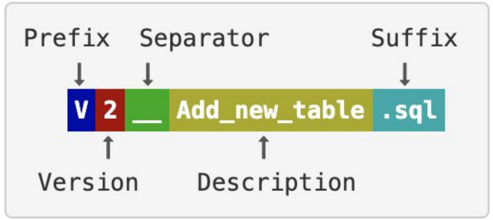
- <*Prefix*> – Default prefix is *V*, which may be configured in configuration file using the ***flyway.sqlMigrationPrefix*** property.
- <*Version*> – Migration version number. Major and minor versions may be separated by an *underscore*. Migration version should always start with 1.
- <*Description*> – Textual description of the migration. The description needs to be separated from the version numbers with a double underscore.

Naming Conventions

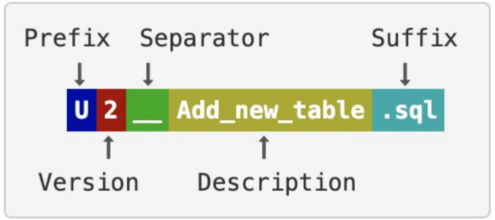
Naming

In order to be picked up by Flyway, SQL migrations must comply with the following naming pattern:

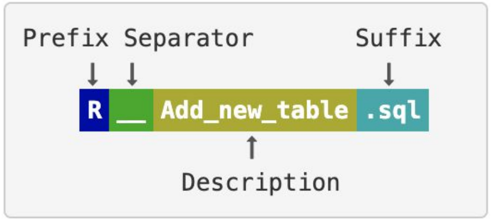
Versioned Migrations



Undo Migrations



Repeatable Migrations



The file name consists of the following parts:

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

How Flyway Works

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

Commands

Flyway supports the following basic commands to manage database migrations.

- ***Info***: Prints current status/version of a database schema. It prints which migrations are pending, which migrations have been applied, what is the status of applied migrations and when they were applied.
- ***Migrate***: Migrates a database schema to the current version. It scans the classpath for available migrations and applies pending migrations.
- ***Baseline***: Baselines an existing database, excluding all migrations, including *baselineVersion*. Baseline helps to start with Flyway in an existing database. Newer migrations can then be applied normally.
- ***Validate***: Validates current database schema against available migrations.
- ***Repair***: Repairs metadata table.
- ***Clean***: Drops all objects in a configured schema. All database objects are dropped. Of course, you should never use clean on any production database.

Exercises

1. Add birthdate to user using Flyway DB
2. Add add user filtering endpoints
   a. Find by firstname or lastname or email
   b. Fetch top ten matching
   c. Make parameters optional
   d. Ignore case
3. Add transaction filtering logic by transaction id
4. Add transaction monitoring columns. They should be automatically saved
   a. Created date
   b. Modified date
5. Add active column to users table
6. Filter transactions by currency

References

- https://www.baeldung.com/database-migrations-with-flyway
- https://www.iban.com/currency-codes
- https://www.concretepage.com/spring-5/spring-data-crudrepository-example
- https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html
- https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html
- https://www.baeldung.com/spring-data-jpa-query
- https://docs.spring.io/spring-data/commons/docs/current/reference/html/#repositories.query-methods.query-creation