COURSEWARE

Custom Queries

Contents

Overview

Spring Data provides a lot of very useful methods for querying the database by default, just by extending `JPARepository` and adding more specific methods can be done easily and intuitively.

Derived queries

Spring Data is capable of deriving queries from method names; using this method for example:

```
@Repository
public interface PersonRepo extends JpaRepository<Person, Long> {

    List<Person> findPersonByName(String name);

}
```

Will generate a query functionally identical to `SELECT * FROM Person WHERE name = '[name]'`.
There are a massive amount of words and phrases that Spring Data is capable of understanding and a full list of them is available from the Spring docs here.

The easiest way to think about this process is to follow these steps:

1. Are you trying to modify one record or multiple? If only one then make the return type a *single* object (like `Person`) otherwise make it a *collection* of objects (like `List<Person>`).
2. What action are you trying to perform; Find? Delete? Update? Put the action at the start of the method name.
3. Are there any parameters/extra criteria for the query? If yes, add them to the end.

Some example methods and the corresponding SQL queries:

```
@Repository
public interface PersonRepo extends JpaRepository<Person, Long> {

    // SELECT * from Person where name = '[name]';
    List<Person> findPersonByName(String name);

    // SELECT * from Person where name = '[name]' and age = [age];
    List<Person> findPersonByNameAndAge(String name, int age);

    // SELECT * from Person where name = '[name]' and age = [age] LIMIT 1;
    Person findPersonByNameAndAge(String name, int age);

    // SELECT * from Person where age > [age];
    List<Person> findPersonByAgeGreaterThan(int age);

    // SELECT * from Person where age < [age];
    List<Person> findPersonByAgeLessThan(int age);

    // SELECT * from Person where name = '[name]' IS NOT NULL;
    List<Person> findPersonByNameIsNotNull();

    // SELECT * from Person where name = '[name]' IS NULL;
    List<Person> findPersonByNameIsNull();

}
```

@Query

Sometimes a query is required that Spring Data can't derive by itself; in those cases it is possible to write the query manually and apply it to a method using the `@Query` annotation.
When using SQL with `@Query` it is necessary to specify `nativeQuery = true`, otherwise Spring will expect a **JPQL** query.

Find all with SQL:

```
    @Query(value = "SELECT * from Person", nativeQuery = true)
    List<Person> findAllSQL();
```

Parameters can be passed from the method to the query using `?[parameter number]`, for example:

```
    @Query(value = "SELECT * from Person WHERE name = ?1", nativeQuery = true)
    List<Person> findPersonByNameSQL(String name); // name is parameter 1

    @Query(value = "SELECT * from Person WHERE name = ?1 and age = ?2",
nativeQuery = true)
    Person findPersonByNameAndAgeSQL(String name, int age); //name is parameter
1 and age is 2.
```

JPQL

Java has its own query language, known as **Java Persistence Query Language**, this is preferable to SQL queries as it makes the query completely database independent.
Using JPQL means the query doesn't have to comply with the SQL syntax of the specific database being used and queries can be written simply using the entity variables rather than the actual field names in the database.

Essentially, JPQL works by assigning each entity in the table to a temporary variable, similar to an **enhanced for loop**; for example - find all with JPQL:

```
    @Query("SELECT p from Person p")
    //value is the default variable and nativeQuery defaults to false so JPQL
queries can be written directly after @Query
    List<Person> findAllJPQL();
```

In this example each person in the `Person` table is assigned to the temporary variable `p` - this allows for queries to check the `Person` classes fields *instead* of the table's.
Parameters can be passed in the same way as with native queries:

```
@Query("SELECT p from Person p WHERE p.name = ?1")
List<Person> findPersonByNameJPQL(String name);

@Query("SELECT p from Person p WHERE p.name = ?1 and p.age = ?2")
Person findPersonByNameAndAgeJPQL(String name, int age);
```

Tutorial

There is no tutorial for this module.

Exercises

First, clone down this repo.

In the `PersonRepo` class there are a selection of methods, the methods using derived queries are already implemented but for each of those there is a SQL and JPQL method with a dummy query (`@Query("SELECT p from Person p")`).
For each of the methods with this default query implement a working SQL or JPQL query to replicate the derived query method and replace the dummy query with it.
Test these new queries using the `PersonRepoTest` class - *you should not need to modify this class!*