Let Spring Data JPA Derive Your Query

Spring Data JPA can generate a query based on the name of a repository method. Your method name has to start with one of the keywords listed in this cheat sheet and follow a specific pattern to use this feature.

Derived queries are great for all statements that aren't too complex. As a rule of thumb, derived queries work great if you don't require more than 1 join clause or more than 2-3 bind parameters. If it gets more complex, you should explicitly define your query.

Define a Derived Query

To let Spring Data JPA generate a query that selects entity objects or other complex data structures, you need to start your method name with one of the following keywords:

- find...By
- get...By
- read...By
- query...By
- search...By
- stream...By

To get a query that counts the number of records that fulfill the defined WHERE clause, you need to use the keyword:

- count...By

If you only need to check if there is at least one record that fulfills the defined WHERE clause, you need to use the keyword:

- exists...By

To get distinct query results, you need to add the following keyword in between previously listed query keywords:

- ...Distinct ...

You can limit the size of your query result to the first *<number>* records by adding the following keyword in between previously listed query keywords:

- ...First<number>...
- ...Top<number>...

Filter the Query Results

The part of the method name after the *By* keyword defines the WHERE clause. It has to follow a specific pattern so that Spring Data JPA can generate the query statement.

Equals Comparison

You can reference one or more entity attributes by their name that you want to check for equality with a provided method parameter with the same name. If you're referencing more than one attribute name, you need to connect them using *And* or *Or*.

This query selects all *ChessPlayer* entities with a provided first name and last name:

```
List<ChessPlayer> findByLastNameAndFirstName(String lastName, String firstName);
```

Filter Predicates

Spring Data JPA supports several filter predicates. You can specify these comparison operations by using one of the keywords listed below together with the name of your entity attribute.

- After / IsAfter
- Before / IsBefore
- Containing / IsContaining / Contains
- Between / IsBetween
- Exists
- EndingWith / IsEndingWith / EndsWith
- False / IsFalse

- GreaterThan / IsGreaterThan
- GreaterThanEqual / IsGreaterThanEqual
- In / IsIn
- Is / Equals / (no keyword)
- Empty / IsEmpty
- NotEmpty / IsNotEmpty
- NotNull / IsNotNull
- Null / IsNull
- LessThan / IsLessThan
- LessThanEqual / IsLessThanEqual
- Like / IsLike
- Near / IsNear
- Not / IsNot
- NotIn / IsNotIn
- NotLike / IsNotLike
- Regex / MatchesReges / Matches
- StartingWith / IsStartingWith / StartsWith
- True / IsTrue
- Within / IsWithin
- IgnoreCase / IgnoringCase
- AllIgnoreCase / AllIgnoringCase

This query selects all *ChessPlayer* entities with a last name that's like the provided String:

> *List<ChessPlayer> findByLastNameLike(String lastName);*

Order the Query Results

You can define an ORDER BY clause by adding the following keywords together with an entity attribute's name to the end of your method name:

- OrderBy...Asc / OrderBy...Desc

This query selects all *ChessPlayer* entities with a provided last name in the ascending order of their first name:

> *List<ChessPlayer> findByLastNameOrderByFirstNameAsc(String lastName);*