**Overview**

Spring Data provides many ways to define a query that we can execute. One of these is the *@Query* annotation.

In this tutorial, we'll demonstrate **how to use the *@Query* annotation in Spring Data JPA to execute both JPQL and native SQL queries**.

Also, we'll show how to build a dynamic query when the *@Query* annotation is not enough.

### **JPQL**

By default the query definition uses JPQL.

Let's look at a simple repository method that returns active User entities from the database:

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

### **Native**

We can use also native SQL to define our query. All we have to do is to set the value of the nativeQuery attribute to true and define the native SQL query in the value attribute of the annotation:

|  |  |
| --- | --- |
|  | @Query(    value = "SELECT \* FROM USERS u WHERE u.status = 1",    nativeQuery = true)  Collection<User> findAllActiveUsersNative(); |

## ****Named Parameters****

We can also **pass method parameters to the query using named parameters.** We define these using the @Param annotation inside our repository method declaration.

Each parameter annotated with @Param must have a value string matching the corresponding JPQL or SQL query parameter name. A query with named parameters is easier to read and is less error-prone in case the query needs to be refactored.

### **JPQL**

As mentioned above, we use the @Param annotation in the method declaration to match parameters defined by name in JPQL with parameters from the method declaration:

|  |  |
| --- | --- |
|  | @Query("SELECT u FROM User u WHERE u.status = :status and u.name = :name")  User findUserByStatusAndNameNamedParams(    @Param("status") Integer status,    @Param("name") String name); |

Note that in the above example, we defined our SQL query and method parameters to have the same names, but it's not required, as long as the value strings are the same:

|  |  |
| --- | --- |
|  | @Query("SELECT u FROM User u WHERE u.status = :status and u.name = :name")  User findUserByUserStatusAndUserName(@Param("status") Integer userStatus,    @Param("name") String userName); |

### **Native**

For the native query definition, there is no difference how we pass a parameter via the name to the query in comparison to JPQL — we use the @Param annotation:

|  |  |
| --- | --- |
|  | @Query(value = "SELECT \* FROM Users u WHERE u.status = :status and u.name = :name",    nativeQuery = true)  User findUserByStatusAndNameNamedParamsNative(    @Param("status") Integer status, @Param("name") String name);  @Query("SELECT new com.example.SpringWebJPA.DTO.FeedBackDTO(f.id, f.description)  FROM FeedBack f ")    FeedBackDTO fetchFeedbackData();    @Query("SELECT f FROM FeedBack f")  Collection<FeedBack> findAllFeedbacks();    @Query(value = "SELECT \* FROM feedback",  nativeQuery = **true**)  Collection<FeedBack> findAllFeedbacksNative(); |