Spring Data JPA provides various options to define and execute queries. All of them use JPA's query capabilities but make them a lot easier to use. You can:

- reference a named native or JPQL query,
- derive queries from the name of a repository method and
- declare a query using the @Query annotation.

In this tutorial, I want to focus on the first option: the referencing of a named native or JPQL query in a Spring Data JPA repository. It makes executing your query much easier because Spring Data takes care of all the boilerplate code required by JPA.

Defining a Named Query with JPA

Named queries are one of the core concepts in JPA. They enable you to declare a query in your persistence layer and reference it by its name in your business code. That makes it easy to reuse an existing query and enables you to separate the definition of your query from your business code.

You can define a named query using a @NamedQuery annotation on an entity class or using a <named-query /> element in your XML mapping. In this article, I will show you the annotation-based mapping, because it's the by far the most common approach to creating a named query.

When you define a named query, you can provide a JPQL query or a native SQL query in very similar ways. Let's take a look at both options.

Defining a Named JPL Query

The definition of a named JPQL query is pretty simple. You just have to annotate one of your entity classes with @NamedQuery and provide 2 Strings for the name and query attributes.

```
@Entity
@NamedQuery(name = "Author.findByFirstName",
        query = "FROM Author WHERE firstName = ?1")
@NamedQuery(name = "Author.findByFirstNameAndLastName",
        query = "SELECT a FROM Author a WHERE a.firstName = ?1 AND
a.lastName = ?2")
public class Author { ... }
```

If you want to define multiple JPQL queries, you can either annotate your class with multiple @NamedQuery annotations, if you use at least JPA 2.2 or Hibernate 5.2. If you are using an older JPA or Hibernate version, you need to wrap your @NamedQuery annotation within a @NamedQueries annotation.

Defining a Named Native Query

You can define a named native query in almost the same way as you specify a named JPQL query. The 3 main differences are:

1. You need to use a @NamedNativeQuery instead of a @NamedQuery annotation.
2. You need to provide an SQL statement instead of a JPQL statement as the value of the query attribute.
3. You can define an entity class or a reference to an @SqlResultSetMapping that will be used to map the result of your query. Spring Data JPA can provide a set of default mappings so that you often don't need to specify it.

```
@Entity
@NamedNativeQuery(name = "Author.findByFirstName",
        query = "SELECT * FROM author WHERE first_name = ?",
        resultClass = Author.class)
@NamedNativeQuery(name = "Author.findByFirstNameAndLastName",
        query = "SELECT * FROM author WHERE first_name = ? AND last_name
= ?",
        resultClass = Author.class)
public class Author { ... }
```

Executing a Named Query Programmatically with JPA

Using JPA's EntityManager, you can run named native and named JPQL queries in the same way:

1. You call the createNamedQuery method on the EntityManager with the name of the named query you want to execute. That gives you an instance of a Query or TypedQuery interface.
2. You then call the setParameter method on the returned interface for each bind parameter used in your query.
3. As a final step, you call the getSingleResult or getResultSet method on the Query or TypedQuery interface to execute the query and to retrieve 1 or multiple result set records.

Here you can see the required code to execute the Author.findByFirstName query that we defined in the 2 previous examples.

```
Query q = em.createNamedQuery("Author.findByFirstName");
q.setParameter(1, "Thorben");
List a = q.getResultList();
```

Referencing a Named Query in a Spring Data JPA repository

As you have seen in the previous example, executing a named query using JPA's EntityManager isn't complicated, but it requires multiple steps.

Spring Data JPA takes care of that if you reference a named query in your repository definition. Doing that is extremely simple if you follow Spring Data's naming convention. The name of your query has to start with the name of your entity class, followed by "." and the name of your repository method.

In the previous examples, I defined the named queries Author.findByFirstName and Author.findByFirstNameAndLastName as JPQL and native queries. You can reference both versions of these queries by adding the methods findByFirstName and

findByFirstNameAndLastName to the repository definition for the Author entity.

```
public interface AuthorRepository extends JpaRepository<Author, Long> {


    List<Author> findByFirstName(String firstName);


    List<Author> findByFirstNameAndLastName(String firstName, String
lastName);


}
```

You can then inject an AuthorRepository instance in your business code and call the repository methods to execute the named queries.

As you can see in the following code snippet, you can use these repository methods in the same way as a repository method that executes a derived query or a declared query. Spring Data JPA handles the instantiation of the named query, sets the bind parameter values, executes the query, and maps the result.

```
List<Author> a = authorRepository.findByFirstName("Thorben");
```