

# Quiz Questions

## (1) Derived Queries

← Classroom

Spring Data JPA Queries  
Derived Query

?

V

Problem Submissions Doubts

✓ Derived Queries I

Easy • Score 20/20

Send feedback

Problem statement

Which of the following statements about derived queries in Spring JPA is correct?

Options: Pick one correct answer from below

☒ Derived queries are created by writing methods in a repository interface with a specific naming convention. ✓

☐ Derived queries are created by defining the SQL query string in a @Query annotation.

☐ Derived queries are only useful for simple queries.

☐ Derived queries require a custom implementation of the repository interface.

Solution description

We can create derived queries in Spring JPA by annotating methods in a repository interface with a specific naming convention. If we follow the naming convention, we can use derived queries to generate complex queries and don't need a custom implementation because Spring JPA provides a default implementation.

## (2) Naming Convention for Derived Queries

← Classroom

Spring Data JPA Queries  
DQ Naming Convention

?

V

Problem Submissions Doubts

✓ Naming Convention for Derived Queries

Easy • Score 20/20

Send feedback

Problem statement

Which keywords are not used in Spring JPA's naming convention for derived queries?

Options: Pick one correct answer from below

☐ find

☐ read

☒ query ✓

☐ get

Solution description

The naming convention for derived queries in Spring JPA uses keywords like "find," "read," and "get" to create queries based on method names. The "query" keyword is not used in this convention.

## (3) DQ Range Filter

← Classroom

Spring Data JPA Queries  
Derived Query Range Filter

?

V

Problem Submissions Doubts

✓ DQ Range Filter

Easy • Score 20/20

Send feedback

Problem statement

Which of the following expressions can be used in derived query method names to filter by a property's value based on a range?

Options: One or more answers may be correct

☐ Before

☐ After

☒ LessThan ✓

☒ GreaterThan ✓

Solution description

- Option c) LessThan: The "LessThan" keyword can be used to filter by a property's value that is less than the specified value. For example, findByAgeLessThan(int age).
- Option d) GreaterThan: The "GreaterThan" keyword can be used to filter by a property's value that is greater than the specified value. For example, findByAgeGreaterThan(int age).

#### (4) Invalid JPA-derived Query Method

← Classroom

Spring Data JPA Queries  
JPA-Derived Query Method

?

V

Problem Submissions Doubts

Invalid JPA-derived Query Method

Easy • Score 20/20

Problem statement

Which of the following is an invalid JPA-derived query method?

Send feedback

Options: Pick one correct answer from below

☐ findByFirstNameAndLastName

☐ deleteByAgeGreaterThan

☐ findDistinctByCity

☒ sortByCreationDateAndStatus

Solution description

Option d) sortByCreationDateAndStatus is not a valid JPA-derived query method because JPA-derived queries do not support sorting by multiple properties. To achieve sorting by multiple properties, you would need to define a custom query using JPQL or the Criteria API.

#### (5) Key-Value pair in Spring Data Repository

← Classroom

Spring Data JPA Queries  
Key-Value Pairs

?

V

Problem Submissions Doubts

key-value pair in Spring Data Repository

Easy • Score 20/20

Problem statement

What is the purpose of the key-value query mechanism in the Spring Data repository infrastructure?

Send feedback

Options: Pick one correct answer from below

☐ To construct key-value pairs for entities of the repository.

☒ To transform key-value pairs into query method names.

☐ To apply static ordering to key-value queries.

☐ To define constraints by traversing nested key-value properties.

Solution description

The key-value query mechanism, present in the Spring Data repository infrastructure, allows for constructing queries by using key-value pairs. It transforms these key-value pairs into query method names, enabling easy and intuitive querying based on the provided keys and values.

#### (6) Filtering in JPQL

← Classroom

Spring Data JPA Queries  
Filtering in JPQL

?

V

Problem Submissions Doubts

Filtering in JPQL

Easy • Score 20/20

Problem statement

In JPQL, the keyword \_\_\_\_\_ is used to specify conditional filtering of query results.

Send feedback

Options: Pick one correct answer from below

☐ SELECT

☒ WHERE

☐ FROM

☐ ORDER BY

Solution description

- In JPQL, the "WHERE" keyword is used to specify conditions or filters for retrieving specific data from a database. It allows you to narrow down the results of a query based on specific criteria.
- The "WHERE" clause is typically placed after the "SELECT" statement in a JPQL query. It is followed by one or more conditions that define the filtering criteria. These conditions are based on comparisons between database fields and values or between different fields.

## (7) Query 1

← Classroom

Spring Data JPA Queries  
Query.I

?

V

Problem Submissions Doubts

Send feedback

**Problem statement**

Suppose you are developing a project management application that tracks projects, tasks, and team assignments. One of the required features is to calculate the total duration of a project by summing up the durations of its tasks. To achieve this, you need to retrieve all tasks associated with a specific project ID.

```
// Task Java Class
@Entity
@Table(name = "tasks")
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Integer duration;
    private Integer projectId;
    // Getters and setters
}
```

Which of the following JPQL queries can retrieve all the tasks associated with a given project ID?

**Options:** Pick one correct answer from below

☒ SELECT t FROM Task t WHERE t.projectId = :projectId ✓

☐ SELECT task FROM tasks WHERE tasks.projectId = :projectId

☐ SELECT Task FROM tasks WHERE Task.projectId = :projectId

☐ SELECT Task FROM Task WHERE Task.projectId = :projectId

**Solution description**

This JPQL query retrieves all the tasks associated with a given project ID by selecting the Task entity (t) from the Task table and filtering the results based on the projectId attribute. The :projectId placeholder represents the parameter value that should be provided when executing the query.

## (8) Query 2

← Classroom

Spring Data JPA Queries  
Query.II

?

V

Problem Submissions Doubts

Send feedback

**Find the Query II**

Easy • Score 20/20

**Problem statement**

In continuation to the Project Management Application. Which of the following queries would you use to calculate the average duration of all tasks in a project using JPQL?

**Options:** Pick one correct answer from below

☒ SELECT AVG(t.duration) FROM Task t WHERE t.projectId = :projectId ✓

☐ SELECT AVG(duration) FROM tasks WHERE projectId = :projectId

☐ SELECT AVG(Task.duration) FROM tasks WHERE Task.projectId = :projectId

☐ SELECT AVG(Task.duration) FROM Task WHERE Task.projectId = :projectId

**Solution description**

To calculate the average duration of all tasks in a project using JPQL, we need to use the AVG function to calculate the average duration attribute. The correct syntax is to specify the attribute (t.duration) and the entity alias t in the AVG function. The WHERE clause filters the tasks based on the project ID.

## (9) Query 3

← Classroom

Spring Data JPA Queries  
Query.III

?

V

Problem Submissions Doubts

Send feedback

**Problem statement**

Suppose you are developing a Spring Boot application for an online bookstore. The application allows users to browse and search for books, add them to their shopping cart, and place orders. As part of the application, you must implement a feature that retrieves a list of books based on specific criteria using JPQL queries.

```
@Entity
@Table(name = "books")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    private double price;
    // Getters and setters
}
```

Which of the following JPQL queries can retrieve a list of books with a price greater than a specified value and order them by price in ascending order?

**Options:** Pick one correct answer from below

☐ SELECT b FROM books WHERE price > :price ORDER BY price ASC

☒ SELECT b FROM Book b WHERE b.price > :price ORDER BY b.price ASC ✓

☐ SELECT Book FROM books WHERE Book.price > :price ORDER BY Book.price ASC

☐ SELECT Book FROM Book WHERE book.price > :price ORDER BY book.price ASC

**Solution description**

To retrieve a list of books with a price greater than a specified value and order them by price in ascending order, we need to use the SELECT statement to specify the entity (Book) and its alias b. The WHERE clause filters the books based on the price condition, and the ORDER BY clause is used to sort the results by the price attribute in ascending order (ASC).

## (10) Debugging JPQL Error

← Classroom

Spring Data JPA Queries  
Debugging JPQL Error

?

V

Problem Submissions Doubts

Debugging JPQL Error

Easy • Score 20/20

Send feedback

Problem statement

Suppose you are working on a Spring Boot application that utilizes JPQL for data retrieval. During testing, an error occurs when executing a JPQL query. Analyze the code snippet below and identify the potential cause for the error.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("SELECT u FROM User u WHERE u.username = :username")
    User findByUsername(@Param("username") String username);
}
```

The error message received is: "No property found for type User! Did you mean 'username'?"

Based on the given code snippet and the error message, choose the most likely cause of the error.

Options: One or more answers may be correct

☒ The User entity does not have a property named username.

☐ The UserRepository interface is not annotated with @Repository.

☐ The findByUsername method is missing the @Param annotation.

☐ The JPQL query syntax is incorrect.

☐ None of the above

Solution description

The error message states, "No property found for type User! Did you mean 'username'?" This suggests that the issue lies in the property access within the User entity. The JPQL query in the UserRepository attempts to access the User entity's username property. However, the User entity does not seem to have a property with the name username. To resolve this error, one should check the User entity class and ensure that it has a username property defined. Additionally, ensure that the property name matches exactly the one used in the JPQL query.

## (11) Native Query Features

← Classroom

Spring Data JPA Queries  
Native Query Features

?

V

Problem Submissions Doubts

Native Query Features

Easy • Score 20/20

Send feedback

Problem statement

Which of the following are valid features of native queries in JPA?

Options: One or more answers may be correct

☒ Support for executing SQL queries directly against the database.

☒ Support for mapping query results to entity objects.

☐ Support for using named parameters in queries.

☐ Support for automatic query generation based on entity mappings.

Solution description

- Native queries in JPA provide support for executing SQL queries directly against the database, allowing developers to leverage the power and flexibility of SQL when needed.
- Additionally, they offer the ability to map query results to entity objects, enabling seamless integration between the database and the object-oriented model.
- However, named parameters are not supported in native queries, and automatic query generation based on entity mappings is a feature provided by JPQL and criteria queries, not native queries.

## (12) Find The Query I

← Classroom

Spring Data JPA Queries  
Find The Query I

?

V

Problem Submissions Doubts

@Query Annotation with Native query in JPA

Easy • Score 20/20

Send feedback

Problem statement

Suppose you are developing a Spring Boot application that manages a blog platform. The blog post entity has the following attributes: 'id' (Long), 'title' (String), 'content' (String), 'author' (String), 'publishedDate' (Date), 'rating' (Double) and 'published' (Boolean). Below is the BlogPostRepository interface with some comments:

```
***
public interface BlogPostRepository extends JpaRepository<BlogPost, Long> {
    // TODO:
    // 1. Implement a native SQL query using @Query to fetch the titles of all published blog
    // posts written by the author.
    // 2. The SQL query should return a List<String> containing the titles of the matching
    // blog posts.
    // 3. Use the native query approach in Spring Data JPA.
    // Method signature (Write the code below)
}
```

Based on the comments, choose the query that retrieves the titles of all published blog posts written by a specific author.

Options: Pick one correct answer from below

☒

```
***
@Query(value = "SELECT title FROM BlogPost WHERE author = ?1 AND published = true",
nativeQuery = true)
List<String> findPublishedBlogPostTitlesByAuthor(String author);
```

☐

```
***
@Query(value = "SELECT title FROM BlogPost WHERE author = ?1 AND published = 1", nativeQuery =
true)
List<String> findPublishedBlogPostTitlesByAuthor(String author);
```

☐

```
***
@Query(value = "SELECT title FROM BlogPost WHERE author = ?1 AND published = 'true'",
nativeQuery = true)
List<String> findPublishedBlogPostTitlesByAuthor(String author);
```

☐

```
***
@Query(value = " SELECT title FROM BlogPost WHERE author = ?1 AND published = '1'",
nativeQuery = true)
List<String> findPublishedBlogPostTitlesByAuthor(String author);
```

### (13) Find the Query II

← Classroom

Spring Data JPA Queries  
Find The Query II

90%

Problem Submissions Doubts

Find the Query II

Easy • Score 20/20

Problem statement

Continuing with the previous problem, you now need to implement a functionality to fetch the titles of all published blog posts, where the title contains a specific keyword and has a rating greater than or equal to a given value using a native SQL query. Choose the correct SQL query that retrieves the title list containing the specified keyword and has a rating greater than or equal to the specified value.

Send feedback

Options: Pick one correct answer from below

☐

```
***
@Query(value = "SELECT title FROM BlogPost WHERE published = true AND title LIKE '%?%' AND
rating >= ?2", nativeQuery = true)
List<String> findPublishedBlogPostTitlesByKeywordAndRating(String keyword, double rating);
```

☒

```
***
@Query(value = "SELECT title FROM BlogPost WHERE published = true AND title LIKE
CONCAT('%', ?1, '%') AND rating >= ?2", nativeQuery = true)
List<String> findPublishedBlogPostTitlesByKeywordAndRating(String keyword,
double rating);
```

☐

```
***
@Query(value = "SELECT title FROM BlogPost WHERE published = true AND title LIKE ?1 AND
rating >= ?2", nativeQuery = true)
List<String> findPublishedBlogPostTitlesByKeywordAndRating(String keyword, double
rating);
```

☐

```
***
@Query(value = "SELECT title FROM BlogPost WHERE published = true AND title LIKE ?1 AND
rating >= ?2", nativeQuery = true)
List<String> findPublishedBlogPostTitlesByKeywordAndRating(String keyword, double
rating);
```

☐

```
***
@Query(value = "SELECT title FROM BlogPost WHERE published = true AND title LIKE
CONCAT('%', ?1, '%') AND rating > ?2", nativeQuery = true)
List<String> findPublishedBlogPostTitlesByKeywordAndRating(String keyword,
double rating);
```

### (14) @Named Query

← Classroom

Spring Data JPA Queries  
@NamedQuery

?

V

Problem Submissions Doubts

@NamedQuery

Easy • Score 20/20

Problem statement

You are working on a Java web application that uses Hibernate for database operations. Your team has implemented a named query in one of the entity classes to fetch a list of products based on their category. The team wants you to verify and correct if any problem exists in the code.

In the given scenario, assume there is a Product entity class with the following named query:

```
***
@NamedQuery(name = "Product.findByCategory", query = "SELECT p FROM
Product p WHERE p.category = ?1")
```

Identify the error in the named query implementation and suggest a fix if required.

Send feedback

Options: Pick one correct answer from below

☐ The named query should use a native SQL rather than JPQL.

☐ The named query is missing the @NamedQuery annotation.

☐ The named query is missing the @Query annotation.

☒ No Error. The named query implementation is correct.

Solution description

The named query implementation is correct as it follows the syntax for a named query in Hibernate. It selects all the Product objects from the database where the category attribute matches the provided category parameter. Therefore, option A is the correct answer.

## (15) Choose the Query I

← Classroom

Spring Data JPA Queries  
[Choose the Query I](#)

Problem Submissions Doubts

✓ Chose the Query I

Easy • Score 20/20

Problem statement

Send feedback

Suppose you are developing an application for an **Animal Care Center**. The application uses Spring Data JPA for database operations. The **Animals** entity has attributes such as 'id' (Long), 'name' (String), 'species' (String), 'age' (Integer), 'gender' (String), and 'vaccinated' (Boolean). You need to implement a native SQL query using **@NamedNativeQuery** to fetch animals of a specific species that are vaccinated.

Which option demonstrates the correct usage of **@NamedNativeQuery** to implement the desired functionality?

Options: Pick one correct answer from below

☒

```
***
@Entity
@NamedNativeQuery(name = "Animals.findVaccinatedAnimalsBySpecies",
    query = "SELECT * FROM animals WHERE species = ?1 AND vaccinated = true",
    resultClass = Animals.class)
public class Animals {
    // Entity attributes and methods
}
```

☐

```
***
@Entity
@NamedNativeQuery(name = Animals.findVaccinatedAnimalsBySpecies,
    query = "SELECT * FROM animals WHERE species = ?1 AND vaccinated = true",
    resultClass = Animals.class)
public class Animals {
    // Entity attributes and methods
}
```

☐

```
***
@Entity
@NamedNativeQuery(name = "Animals.findVaccinatedAnimalsBySpecies",
    query = "SELECT * FROM Animals WHERE species = ?1 AND vaccinated = true",
    resultClass = Animals.class)
public class Animals {
    // Entity attributes and methods
}
```

☐

```
***
@Entity
@NamedNativeQuery(name = "findVaccinatedAnimalsBySpecies",
    query = "SELECT * FROM Animals WHERE species = ?1 AND vaccinated = true",
    resultClass = Animals.class)
public class Animals {
    // Entity attributes and methods
}
```

## (16) Choose the Query II

← Classroom

Spring Data JPA Queries  
[Choose the Query II](#)

Problem Submissions Doubts

✓ Chose the Query II

Easy • Score 20/20

Problem statement

Send feedback

Continuing the previous problem, you must implement a native SQL query using the **@NamedNativeQuery** annotation to count the number of animals whose names contain a specific substring and belong to a particular species. Which of the following options shows the correct usage of **@NamedNativeQuery** to implement the desired functionality?

Options: Pick one correct answer from below

☐

```
***
@Entity
@NamedNativeQuery(name = "Animals.countAnimalsBySpeciesAndSubstring",
    query = "SELECT COUNT(*) FROM Animals WHERE species = ?1 AND name LIKE CONCAT('%', ?2, '%')",
    resultClass = Long.class)
public class Animals {
    // Entity attributes and methods
}
```

☐

```
***
@Entity
@NamedNativeQuery(name = "countAnimalsBySpeciesAndSubstring",
    query = "SELECT COUNT(*) FROM animals WHERE species = ?1 AND name LIKE CONCAT('%', ?2, '%')",
    resultClass = Long.class)
public class Animals {
    // Entity attributes and methods
}
```

☒

```
***
@Entity
@NamedNativeQuery(name = "Animals.countAnimalsBySpeciesAndSubstring",
    query = "SELECT COUNT(*) FROM animals WHERE species = ?1 AND name LIKE CONCAT('%', ?2, '%')",
    resultClass = Long.class)
public class Animals {
    // Entity attributes and methods
}
```

☐

```
***
@Entity
@NamedNativeQuery(name = "countAnimalsBySpeciesAndSubstring",
    query = "SELECT COUNT(*) FROM Animals WHERE species = ?1 AND name LIKE CONCAT('%', ?2, '%')",
    resultClass = Long.class)
public class Animals {
    // Entity attributes and methods
}
```