JPA Queries, Raw SQL, Hibernate Queries, and ORM Methods Using Repositories in Spring Boot

In this article, we will explore various querying techniques in Spring Boot applications using JPA (Java Persistence API) repositories. We will cover JPA queries, raw SQL queries, Hibernate queries, and ORM methods, providing detailed examples for each approach.

JPA Queries Using Repositories

JPA repositories allow you to create custom queries using the `@Query` annotation and JPQL (Java Persistence Query Language).

Example 1: Simple Select Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Query("SELECT u FROM User u WHERE u.name = :name")
    Optional<User> findByName(@Param("name") String name);
}
```

Example 2: Select with Multiple Conditions

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Query("SELECT u FROM User u WHERE u.name = :name AND u.email = :email")
    Optional<User> findByNameAndEmail(@Param("name") String name, @Param("email") St
}
```

Example 3: Update Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Modifying
    @Query("UPDATE User u SET u.email = :email WHERE u.name = :name")
    int updateUserEmail(@Param("name") String name, @Param("email") String email);
}
```

Example 4: Delete Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Modifying
    @Query("DELETE FROM User u WHERE u.name = :name")
    int deleteUserByName(@Param("name") String name);
}
```

Example 5: Aggregation Query

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

Raw SQL Queries Using Repositories

Spring Data JPA allows executing native SQL queries by annotating methods with `@Query` and setting the `nativeQuery` attribute to `true`.

Example 1: Simple Select Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Query(value = "SELECT * FROM users WHERE name = :name", nativeQuery = true)
    Optional<User> findByNameNative(@Param("name") String name);
}
```

Example 2: Select with Multiple Conditions

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Query(value = "SELECT * FROM users WHERE name = :name AND email = :email", nati
    Optional<User> findByNameAndEmailNative(@Param("name") String name, @Param("emai
}
```

Example 3: Update Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Modifying
    @Query(value = "UPDATE users SET email = :email WHERE name = :name", nativeQuery
    int updateUserEmailNative(@Param("name") String name, @Param("email") String ema
}
```

Example 4: Delete Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Modifying
    @Query(value = "DELETE FROM users WHERE name = :name", nativeQuery = true)
    int deleteUserByNameNative(@Param("name") String name);
}
```

Example 5: Aggregation Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Query(value = "SELECT COUNT(*) FROM users WHERE email = :email", nativeQuery = 
    Long countUsersByEmailNative(@Param("email") String email);
}
```

Hibernate Queries Using Repositories

Hibernate queries can be executed using the Hibernate Query Language (HQL). These queries are similar to JPQL but are specific to Hibernate. While typically not used directly in repositories, you can access the Hibernate session from a repository to execute HQL queries.

Example 1: Simple Select Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    default List<User> findByNameUsingHQL(String name) {
        Session session = entityManager.unwrap(Session.class);
        Query query = session.createQuery("FROM User u WHERE u.name = :name");
        query.setParameter("name", name);
        return query.getResultList();
    }
}
```

Example 2: Select with Multiple Conditions

```
public interface UserRepository extends JpaRepository<User, UUID> {
    default List<User> findByNameAndEmailUsingHQL(String name, String email) {
        Session session = entityManager.unwrap(Session.class);
        Query query = session.createQuery("FROM User u WHERE u.name = :name AND u.em
        query.setParameter("name", name);
        query.setParameter("email", email);
        return query.getResultList();
    }
}
```

Example 3: Update Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Transactional
    default int updateUserEmailUsingHQL(String name, String email) {
        Session session = entityManager.unwrap(Session.class);
        Query query = session.createQuery("UPDATE User u SET u.email = :email WHERE
        query.setParameter("name", name);
        query.setParameter("email", email);
        return query.executeUpdate();
    }
}
```

Example 4: Delete Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    @Transactional
    default int deleteUserByNameUsingHQL(String name) {
        Session session = entityManager.unwrap(Session.class);
        Query query = session.createQuery("DELETE FROM User u WHERE u.name = :name")
        query.setParameter("name", name);
        return query.executeUpdate();
    }
}
```

Example 5: Aggregation Query

```
public interface UserRepository extends JpaRepository<User, UUID> {
    default Long countUsersByEmailUsingHQL(String email) {
        Session session = entityManager.unwrap(Session.class);
        Query query = session.createQuery("SELECT COUNT(u) FROM User u WHERE u.email
        query.setParameter("email", email);
        return (Long) query.getSingleResult();
    }
}
```

ORM Methods Using Repositories

ORM (Object-Relational Mapping) methods allow you to perform CRUD operations using the repository interfaces provided by Spring Data JPA.

Example 1: Save Entity

```
public interface UserRepository extends JpaRepository<User, UUID> {
    // No additional methods required for save
}

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public User saveUser(User user) {
        return userRepository.save(user);
    }
}
```

Example 2: Find Entity by ID

```
public interface UserRepository extends JpaRepository<User, UUID> {
    // No additional methods required for findById
}

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public Optional<User> findUserById(UUID id) {
        return userRepository.findById(id);
    }
}
```

Example 3: Find All Entities

```
public interface UserRepository extends JpaRepository<User, UUID> {
    // No additional methods required for findAll
}
```

```java
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public List<User> findAllUsers() {
        return userRepository.findAll();
    }
}
```

Example 4: Delete Entity by ID

```java
public interface UserRepository extends JpaRepository<User, UUID> {
    // No additional methods required for deleteById
}

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public void deleteUserById(UUID id) {
        userRepository.deleteById(id);
    }
}
```

Example 5: Count All Entities

```java
public interface UserRepository extends JpaRepository<User, UUID> {
    // No additional methods required for count
}

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public long countAllUsers() {
        return userRepository.count();
    }
}
```

Conclusion

This article covered various ways to query data in a Spring Boot application using JPA repositories. We explored JPA queries, raw SQL queries, Hibernate queries, and ORM methods, providing examples for each approach. By leveraging these techniques, you can effectively query and manipulate data in your Spring Boot applications.