CONSULTAS EN BASE AL NOMBRE DEL MÉTODO

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

Obviado por Spring Data

Procesado por Spring Data

Los parámetros son necesarios para poder realizar la consulta

CONSULTAS EN BASE AL NOMBRE DEL MÉTODO

Palabra	Ejemplo	JPQL
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1

El resto de la tabla lo podemos encontrar en la documentación oficial de Spring Data

CONSULTAS JPQL

- Similares a SQL
- Nos permiten *bucear* por los objetos

Dependiendo de su definición:

- *@NamedQuery*
- *@Query*

CONSULTAS JPQL: @NamedQuery

- Permiten definir una consulta en la definición del *bean* entidad.
- Podemos referirnos a ella implementando un método en el repositorio.

```
@Entity
@NamedQuery(name = "Student.findById", query = "select s from Student s where s.id = ?1")
@Table(name="STUDENT")
public class Student {

    //resto del código de la clase

}
```

CONSULTAS JPQL: @Query

- Permiten definir una consulta junto con el método en el controlador.
- Mejor cuando el número de consultas

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long>{

    //resto del código

    /*
     * CONSULTAS AVANZADAS
     */

    /* Realizar la búsqueda en base al id */
    @Query("select s from Student s where s.id = ?1")
    public Student findById(Long id);

}
```

CONSULTAS JPQL: @Query con *LIKE*

- Podemos usar el operador LIKE y el carácter comodín %

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long>{

    //resto del código

    /*
     * CONSULTAS AVANZADAS
     */

    //resto del código

    /* Realizar la búsqueda si el apellido comienza con una cadena */
    @Query("select s from Student s where s.lastName = ?1%")
    public List<Student> findByLastNameStartsWith(String lastName);

}
```

CONSULTAS JPQL: @Query con *consultas nativas*

- Podemos crear consultas SQL
- Es menos recomendable

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long>{

    //resto del código

    /*
     * CONSULTAS AVANZADAS
     */

    //resto del código

    /* Realizar la búsqueda si coincide el nombre */
    @Query("SELECT * FROM STUDENTS WHERE FIRST_NAME = ?1", nativeQuery = true)
    public List<Student> findByFirstName(String firstName);

}
```

CONSULTAS JPQL: @Query con parámetros con nombre

- Definimos los parámetros con un nombre, y no un índice.

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long>{

    //resto del código

    /*
     * CONSULTAS AVANZADAS
     */

    //resto del código

    /* Realizar la búsqueda si coincide el nombre y el apellido */
    @Query("select s from Student s where s.firstName = :firstname or u.lastname = :lastname")
    public List<Student> findByFirstNameOrLastName(
            @Param("firstname") String firstName,
            @Param("firstname") String lastName);

}
```

CONSULTAS JPQL: @Query con modificación (@Modifying)

- Tenemos que ejecutar la consulta en un marco transaccional.
- Anotamos con *@Transactional* el método del controlador que use la consulta.

```
@RequestMapping(value = "/edit", method = RequestMethod.POST)
@Transactional
public String submitEdit(@ModelAttribute("studentForm") Student student,  Model model) {

    studentRepository.setFirstNameAndLastNameFor(student.getFirstName(), student.getLastName(), student.getId());
    return "redirect:/list";
}
```