SpringData - Consultas

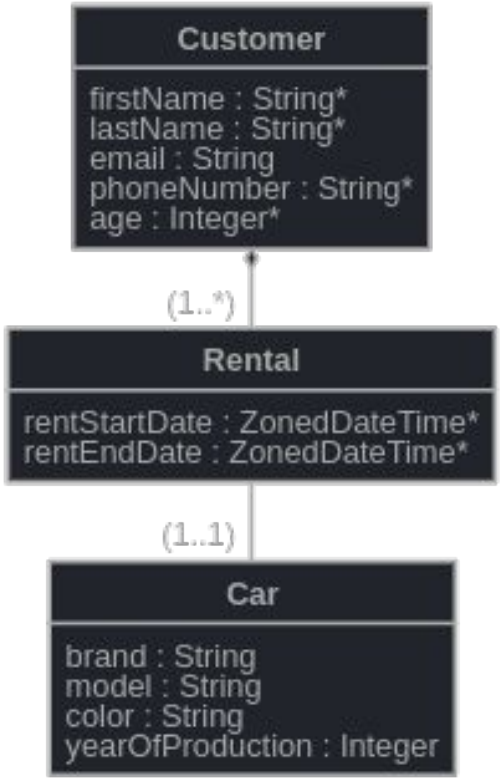
UNRN Universidad Nacional de Río Negro

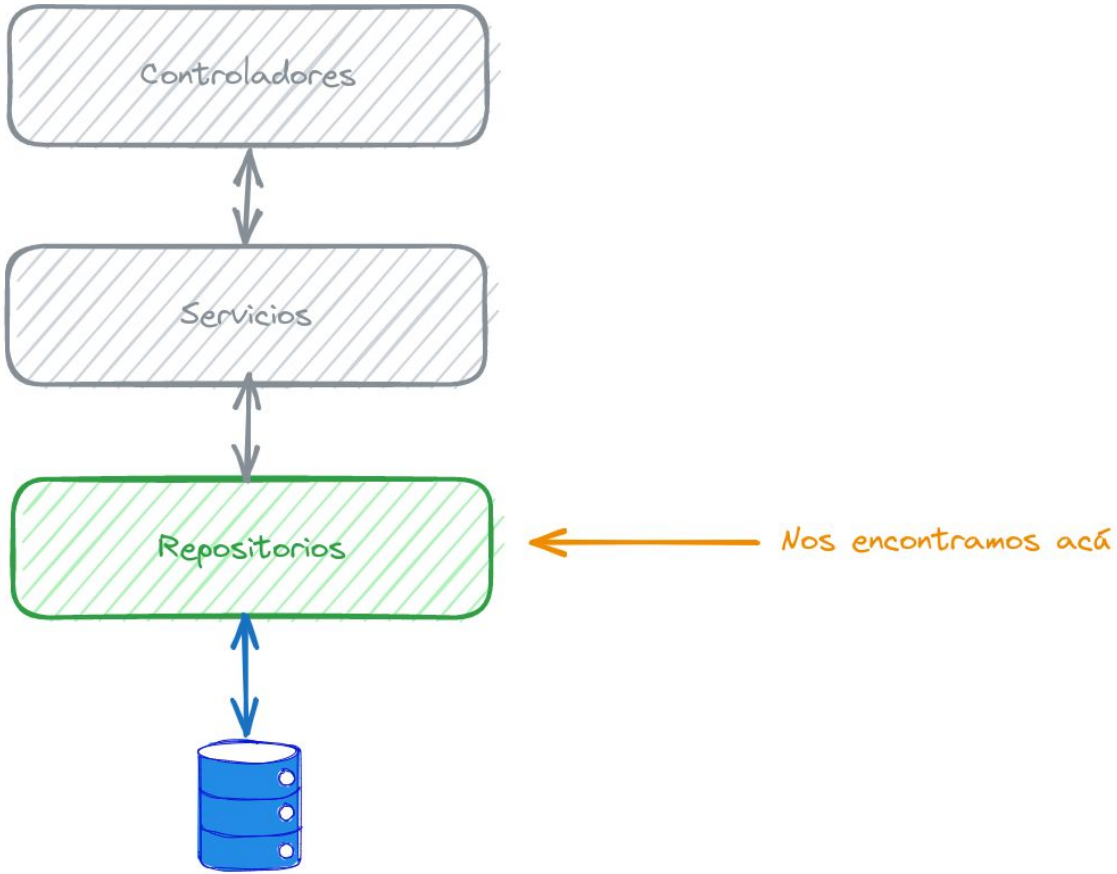
LIA LABORATORIO DE INFORMÁTICA APLICADA

UNRN

Repasando…

LIA LABORATORIO DE INFORMÁTICA APLICADA




Controladores
Servicios
Repositorios
Nos encontramos acá

Operaciones

Cuando creamos una repositorio y extendemos de la clase **JpaRepository**

- ➔ Insertar
- ➔ Eliminar
- ➔ Actualizar
- ➔ **Consultar**

```
public interface UserRepository extends JpaRepository<User, Long> {
}
```

Índice

Consulta

JPQL

JPQL (Java Persistence Query Language) es un lenguaje de consulta orientado a objetos que se utiliza en el contexto de la persistencia de datos en aplicaciones Java utilizando la tecnología Java Persistence API (JPA)

- ➔ Se asemeja a **SQL**
- ➔ Permite a los desarrolladores realizar consultas utilizando **entidades JPA**
- ➔ JPQL es **independiente** del proveedor de persistencia subyacente
- ➔ **Expresa** consultas en términos de objetos y relaciones entre ellos

El proveedor de persistencia JPA más utilizado es **Hibernate**

UNRN

Uso de JPQL en anotación @Query

LIA LABORATORIO DE INFORMÁTICA APLICADA

La anotación @Query permite definir **consultas personalizadas** utilizando JPQL directamente en el código de la aplicación

Ejemplo -> “Seleccionar todos los clientes”

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    @Query("SELECT c FROM Customer c")
    List<Customer> findAllCustomers();

}
```

UNRN

Operadores lógicos

LIA LABORATORIO DE INFORMÁTICA APLICADA

Los operadores lógicos nos permiten construir condiciones de filtrado

- **Or**

```
SELECT c FROM Customer c WHERE c.age > 40 OR c.email = 'example@email.com'
```

- **And**

```
SELECT c FROM Customer c WHERE c.age > 30 AND c.lastName = 'Smith'
```

- **Not**

```
SELECT c FROM Customer c WHERE NOT c.phoneNumber = '123456789'
```

UNRN

Operadores lógicos

LIA LABORATORIO DE INFORMÁTICA APLICADA

- Is Null

```
SELECT c FROM Customer c WHERE c.firstName IS NULL
```

- Is Not Null

```
SELECT c FROM Customer c WHERE c.email IS NOT NULL
```

UNRN

Otros operadores

LIA LABORATORIO DE INFORMÁTICA APLICADA

- OrderBy

```
SELECT c FROM Customer c ORDER BY c.lastName ASC, c.firstName ASC
```

- Between

```
SELECT c FROM Customer c WHERE c.age BETWEEN 30 AND 40
```

Los datos(**age**) de la consulta **Between** están fijos. ¿Cómo los puedo parametrizar?

UNRN

LIA LABORATORIO DE INFORMÁTICA APLICADA

Consulta Parametrizada con @Param

La anotación **@Param** se utiliza en combinación con la anotación **@Query** para enlazar los parámetros de la consulta con los parámetros del método

Ejemplo -> “Consultar todo los Customer entre dos edades”

```
@Query("SELECT c FROM Customer c WHERE c.age BETWEEN :minAge AND :maxAge")
List<Customer> findByAgeRange(@Param("minAge") int minAge, @Param("maxAge") int maxAge);
```

UNRN

Consultas derivadas

LIA LABORATORIO DE INFORMÁTICA APLICADA

Es una característica de JPA que permite definir consultas utilizando **convenciones** de nombres en los métodos de los repositorios

Ejemplos

- ➔ **findBy**LastNameAndFirstName(String lastName, String firstName)
- ➔ **findBy**LastNameStartingWith(String prefix)
- ➔ boolean **existsBy**FirstName(String firstName)
- ➔ long **countBy**FirstName(String firstName)

- ★ No usamos @Query
- ★ Más declarativo

UNRN

Query Nativa

LIA LABORATORIO DE INFORMÁTICA APLICADA

Es una consulta escrita en el lenguaje **SQL nativo** del sistema de gestión de bases de datos (DBMS) subyacente, en nuestro caso **PostgreSQL**

Los podemos utilizar

- ➔ @Query(value= “CONSULTA_SQL”, native = true)
- ➔ @NamedNativeQuery

UNRN

Ordenamiento

LIA LABORATORIO DE INFORMÁTICA APLICADA

Nos permite ordenar el resultado de una consulta

- Consulta derivada
  - `findByOrderByAgeAsc`
- Parámetro
  - `repository.findAll(Sort.by(Sort.Direction.ASC, "age"))`

UNRN

Paginación

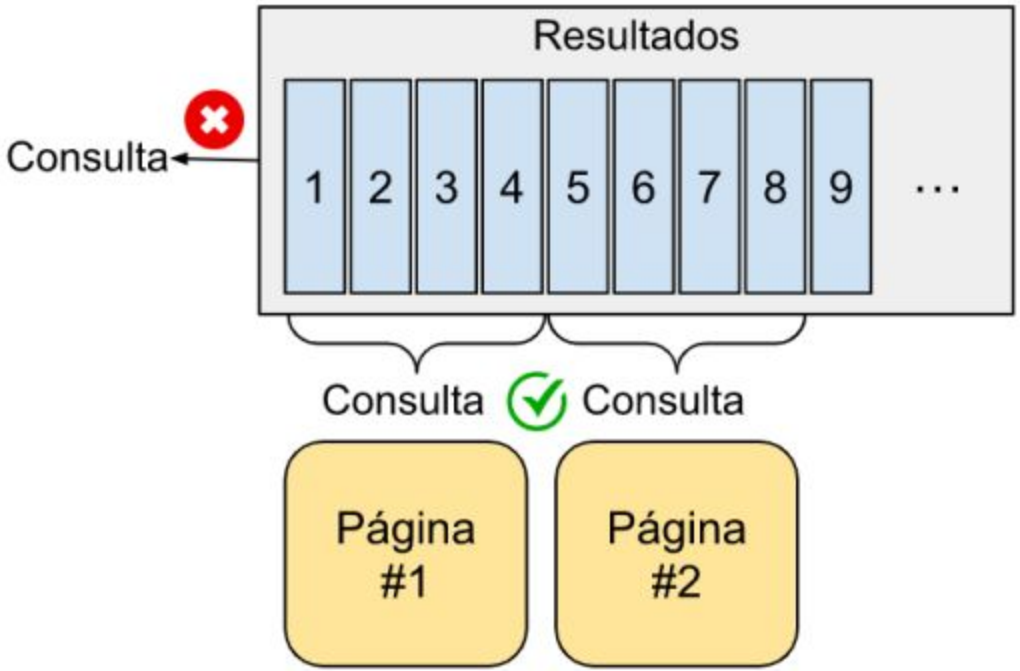
LIA LABORATORIO DE INFORMÁTICA APLICADA

Hasta ahora tenemos **@Query** + **findAll()** para poder realizar consultas, pero con **pocos datos**…

¿Qué pasa si tengo una DB con 100K o 1M de Customer y ejecutó findAll()?

- ➔ Consumo excesivo de **memoria**
- ➔ Tiempo de **respuesta** lento
- ➔ Impacto en el **rendimiento** de la base de datos

Para evitar estos problemas, se recomienda utilizar la **paginación**


Resultados
Consulta
1
2
3
4
5
6
7
8
9
...
Consulta
Consulta
Página #1
Página #2

UNRN

Paginación

LIA LABORATORIO DE INFORMÁTICA APLICADA

La paginación en JPA nos permite recuperar un **conjunto específico** de resultados de una consulta en lugar de todos los datos a la vez

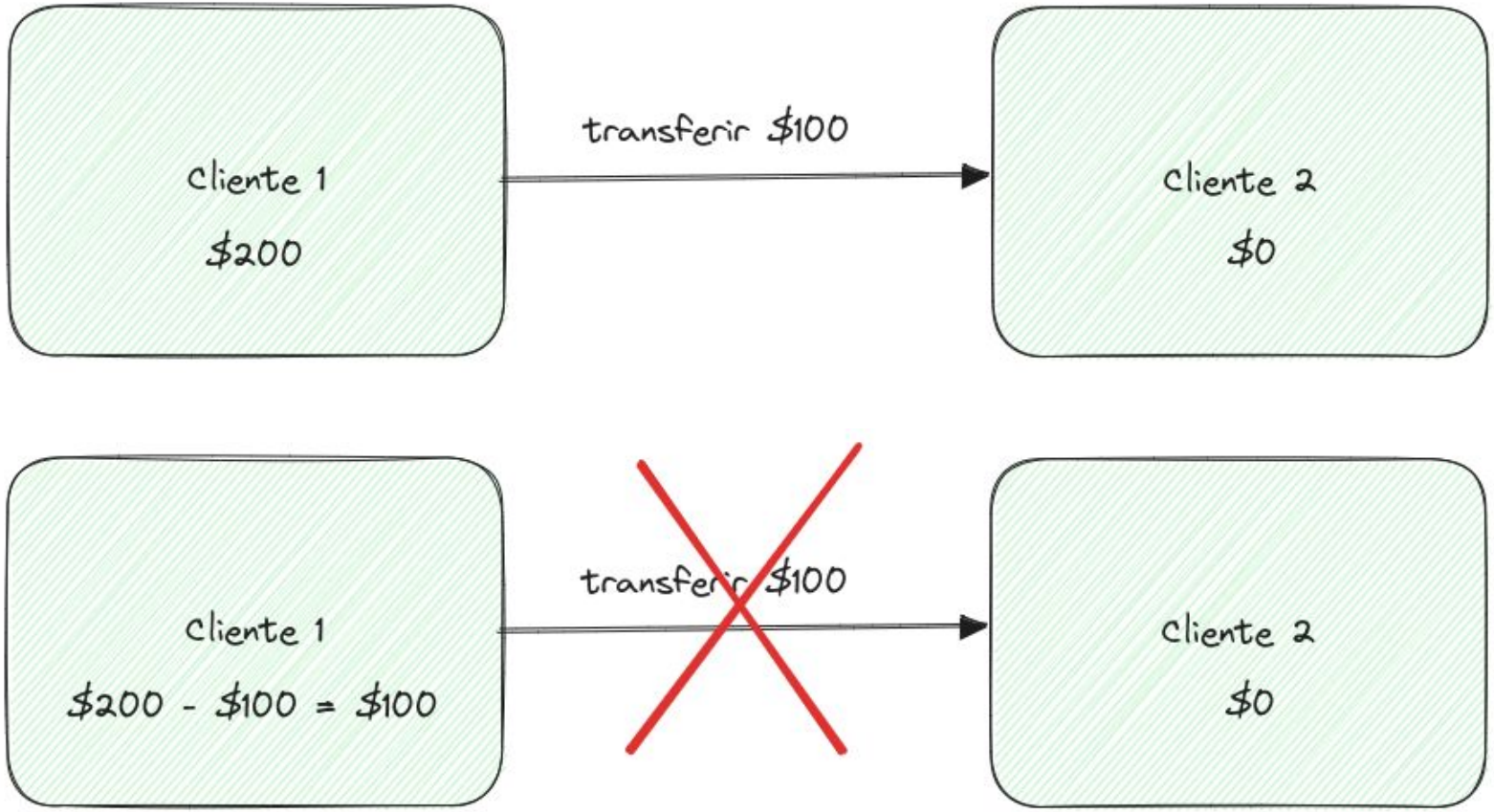
Esto es útil cuando tienes una gran cantidad de registros y deseas mostrarlos o procesarlos por **lotes** más pequeños(páginas)

```
Page<Customer> findByLastname(String lastname, Pageable pageable)
```

```
PageRequest.of(NRO_PAGINA, TAMAÑO_PAGINA)
```

TOTAL_PAGINAS = CANTIDAD_TOTAL / TAMAÑO_PAGINA -> **100** / **20** = **5**

Transacción



¿Qué pasó con los $100?

UNRN

Transacción

LIA LABORATORIO DE INFORMÁTICA APLICADA

“Una transacción en una base de datos es una **secuencia lógica** de operaciones que se ejecutan como una **unidad indivisible**. Estas operaciones pueden ser inserciones, actualizaciones o eliminaciones de datos en la base de datos.”

El objetivo principal de una transacción es asegurar que todas las operaciones se realicen de manera **exitosa** o que se **reviertan** por completo si ocurre algún error.

Nos garantiza…

- ★ **Integridad**
- ★ **Consistencia**

Transacción - ACID

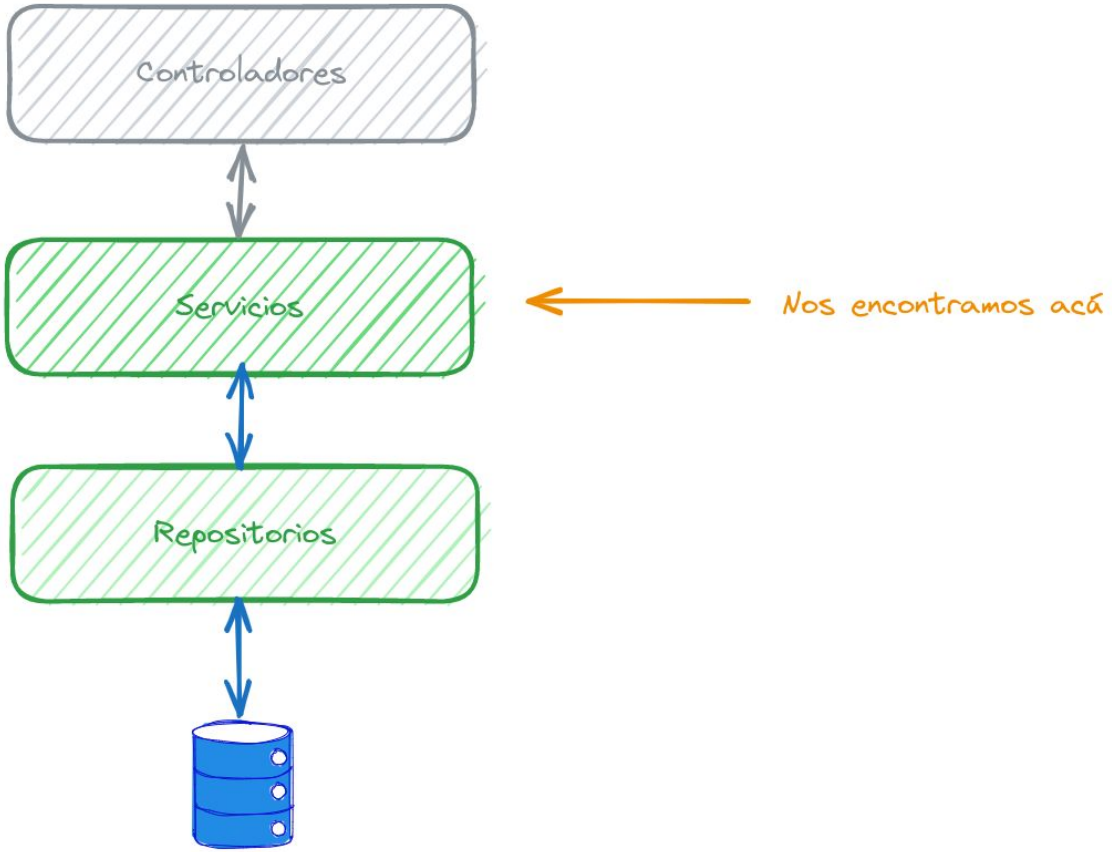
- **Atomicidad**: Una transacción se trata como una unidad atómica de trabajo, lo que significa que todas las operaciones dentro de la transacción se realizan o ninguna se realiza. Si alguna operación falla, se deshacen todas las operaciones previas.

- **Consistencia**: Una transacción lleva la base de datos de un estado consistente a otro estado consistente. Esto significa que todas las restricciones y reglas de integridad definidas en la base de datos deben cumplirse antes y después de la transacción.

- **Aislamiento**: Las transacciones se ejecutan en forma aislada y no deben interferir entre sí. Esto significa que los cambios realizados por una transacción no son visibles para otras transacciones hasta que se complete y se confirme.

- **Durabilidad**: Una vez que una transacción se completa y se confirma, los cambios realizados se mantienen de manera permanente en la base de datos, incluso en caso de fallos del sistema o reinicios.

Servicio


Controladores
Servicios
Repositorios
Nos encontramos acá

UNRN

Servicio

LIA LABORATORIO DE INFORMÁTICA APLICADA

La anotación **@Service** se utiliza comúnmente para marcar las clases que encapsulan la lógica de negocio de una aplicación

Estas clases pueden realizar operaciones como:

- ➔ Cálculos
- ➔ Llamadas a la base de datos
- ➔ Interacciones con servicios externos

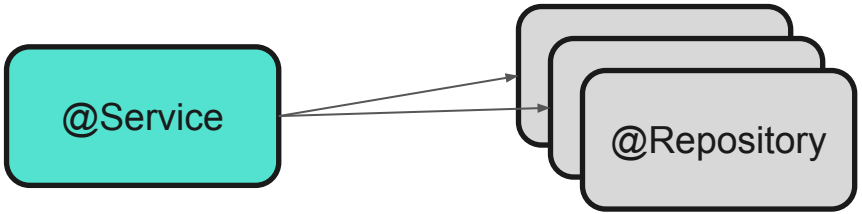
★ Lo podemos inyectar con **@Autowired**

lógica de negocio -> **reglas y procesos que definen el comportamiento**

“Enlazando” los *beans*

```
@Repository
class UsuarioRepository {
}
```


```
@Service
public class UsuarioService {
@Autowired
UsuarioRepository usuarioRepository;
}
```



UNRN

@Service + @Transactional

LIA LABORATORIO DE INFORMÁTICA APLICADA

```
@Service
public class CustomerService {
    @Autowired
    private CustomerRepository customerRepository;

    @Transactional
    public void createCustomer(String firstName, String lastName, int age) {
        Customer customer = new Customer();
        customer.setFirstName(firstName);
        customer.setLastName(lastName);
        customer.setAge(age);
        customerRepository.save(customer);
    }
```

Un ejemplo más complejo…

Queremos registrar una renta donde se tiene que indicar el cliente y el auto a rentar en un rango de fecha valida

Reglas:

- ➔ Que el cliente sea mayor edad
- ➔ Que las fechas de renta seleccionadas no se superpongan

UNRN Universidad Nacional de **Río Negro**

LIA LABORATORIO DE INFORMÁTICA APLICADA

unrn.edu.ar @unrionegro