

# UA05. ORM

---

## 5.5. CRUD

---

Ya hemos llegado al punto en que tenemos todo preparado para poder trabajar con Hibernate en las operaciones fundamentales de una base de datos, las operaciones CRUD.

- **Create:** Guardar un nuevo objeto en la base de datos.
- **Read:** Leer los datos de un objeto de la base de datos.
- **Update:** Actualizar los datos de un objeto de la base de datos.
- **Delete:** Borrar los datos de un objeto de la base de datos.

Estas 4 operaciones será tan sencillas de usar desde hibernate como llamar a un único método para cada uno de ellos.

```
private static SessionFactory sessionFactory = null;  
private static void guardar() {  
    PersonasEntity persona = new PersonasEntity("Pocholo", "H");  
    Session session = sessionFactory.openSession();
```

```
Transaction transaction = session.beginTransaction();
int id = (int) session.save(persona);
transaction.commit();
System.out.println(id);
sessionFactory.close();
}
```

Para la lectura existen varios métodos de la clase **Session**:

- **get(Class, Serializable)**, al que le deberemos pasar la clase que queremos leer y su clave primaria.
- **load(Class, Serializable)** acepta un objeto Class, y cargará el estado de una nueva instancia de esa clase, inicializada en estado persistente.

```
private static void leer(int id) throws Exception {
    session = instancia.abrir();
    Transaction transaction=session.beginTransaction();
    PersonasEntity persona = session.load(PersonasEntity.class, id);//
    System.out.println(persona.getNombre());
    transaction.commit();
    instancia.cerrar();
}
```

Para sentencias SQL, también se pueden usar varios métodos:

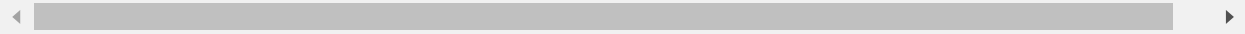
- **createQuery**

- createNamedQuery
- createNativeQuery
- createSQLQuery

## createQuery

Método usado para crear consultas dinámicas, las cuales están definidas directamente mediante un String. Por ejemplo:

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .setMaxResults(10)  
        .getResultList();  
}
```



Este método permite usar lenguaje HQL. Por ejemplo:

```
public List findWithName(String name) {  
    return em.createQuery(  
        "FROM Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .setMaxResults(10)  
        .getResultList();  
}
```

## createNamedQuery

Método usado para crear consultas estáticas o consultas que usan anotación `javax.persistence.NamedQuery`. El elemento `name` de `@NamedQuery` especifica el nombre de la consulta que será usada mediante el método `createNamedQuery`. La consulta en sí de `@NamedQuery` estará especificada en `query`:

```
@NamedQuery(  
    name="findAllCustomersWithName",  
    query="SELECT c FROM Customer c WHERE c.name LIKE :custName"  
)
```

Para acceder a este método, se realiza una llamada al nombre de la consulta:

```
@PersistenceContext  
public EntityManager em;  
...  
customers = em.createNamedQuery("findAllCustomersWithName")  
    .setParameter("custName", "Smith")  
    .getResultList();
```

## createNativeQuery y createSQLQuery

Permite ejecutar consultas con código SQL nativo.

```
sesion.createSQLQuery("SELECT * FROM PersonasEntity WHERE  
idPersona=6").iterate();
```

```
sesion.createNativeQuery("SELECT * FROM PersonasEntity WHERE  
idPersona=6").iterate();
```

Los tipos de consulta SQL más básicas para obtener una lista de escalares (valores) son los siguientes:

```
sess.createNativeQuery("SELECT * FROM Personas").list();  
entityManager.createNativeQuery("SELECT ID,NOMBRE, EDAD FROM PERSONAS
```

Estas retornarán una lista de objetos arrays (Object[]) con valores escalares para cada columna en la tabla PERSONAS. Hibernate utilizará ResultSetMetadata para deducir el orden real y los tipos de los valores escalares retornados.

Otro tipo de consulta más compleja, es la consulta de entidades. Para obtener los objetos entidades desde una consulta sql nativa, se utiliza por medio de addEntity().

```
sess.createNativeQuery("SELECT * FROM PERSONAS").addEntity(Persona.class);  
entityManager.createNativeQuery("SELECT * FROM PERSONAS", Persona.class);
```

```
private static void actualizar(int id,String nombre, String tipo)  
    session = instancia.abrir();  
    Transaction transaction = session.beginTransaction();  
    PersonasEntity persona = session.get(PersonasEntity.class,id)
```

```
    persona.setNombre(nombre);  
    persona.setTipo(tipo);  
    // session.saveOrUpdate(persona);        // session.merge(persc  
    session.update(persona);  
    transaction.commit();  
    instancia.cerrar();  
}
```

Para actualizar podemos usar:

- **update**, actualiza un objeto usando su **id** y seteando los campos que se deseen

◀ modificar. ▶

- **saveOrUpdate**, muchas veces resulta cómodo al programar no tener que estar pendiente de si un objeto va a insertarse o actualizarse. Para ello, este método, inserta o actualiza en la base de datos en función de si ya existe o no dicha fila.
- **merge**, copia el estado del objeto dado en el objeto persistente con el mismo identificador.

La mejor forma de eliminar un objeto es cargarlo mediante **get** o **load** según algún criterio para luego eliminarlo.

```
session.get(Entidad.class, 8);  
session.delete(profesor);
```