ORM

✅ ¿Qué es ORM?

ORM (Mapeo Objeto-Relacional) es una técnica que permite trabajar con bases de datos relacionales **usando objetos de un lenguaje de programación** (como Java), evitando escribir directamente consultas SQL.

🧱 Problema que resuelve

- Los **lenguajes orientados a objetos** (como Java) y las **bases de datos relacionales** (como MySQL) **tienen estructuras diferentes**.
- ORM **traduce automáticamente** objetos a registros de base de datos y viceversa.

⚙️ ¿Cómo funciona?

1. Cada **clase Java** representa una **tabla** de la base de datos.
2. Cada **atributo** de la clase representa una **columna**.
3. Cada **instancia de la clase** representa una **fila**.
4. El ORM se encarga de **convertir objetos en sentencias SQL** (y viceversa) automáticamente.

🔧 Herramientas ORM populares en Java

ORM	Descripción
JPA	Estándar de Java para ORM (no es una librería concreta). Necesita una implementación.
Hibernate	Implementación más popular de JPA. Ofrece muchas funcionalidades extra.
EclipseLink	Implementación oficial de JPA (de Oracle). Menos usada que Hibernate.

🛠️ Anotaciones comunes en JPA

Anotación	Uso
`@Entity`	Marca una clase como entidad (tabla).
`@Id`	Indica el atributo clave primaria.
`@GeneratedValue`	Genera el ID automáticamente.
`@Table`	(Opcional) Cambia el nombre de la tabla.

Anotación	Uso
`@Column`	(Opcional) Cambia el nombre o propiedades de la columna.
`@OneToMany`	Relación uno a muchos.
`@ManyToOne`	Relación muchos a uno.
`@ManyToMany`	Relación muchos a muchos.
`@JoinColumn`	Define la clave foránea.

🔁 Operaciones CRUD con ORM

Operación	Descripción	Método típico (con `EntityManager`)
Crear	Insertar objeto	`persist(obj)`
Leer	Buscar por ID	`find(Clase.class, id)`
Actualizar	Modificar objeto	`merge(obj)`
Eliminar	Borrar objeto	`remove(obj)`

📄 `persistence.xml`

Archivo de configuración que indica:

- Nombre de la unidad de persistencia (`persistence-unit`)
- Base de datos y usuario
- Dialecto (ej: `org.hibernate.dialect.MySQL8Dialect`)
- Proveedor (Hibernate, EclipseLink…)
- Entidades usadas

🔍 Consultas

🔸 JPQL (Java Persistence Query Language)

- Lenguaje orientado a objetos (consulta sobre clases, no tablas).

```
Query q = em.createQuery("SELECT p FROM Producto p WHERE p.precio > :min");
q.setParameter("min", 10);
List<Producto> resultados = q.getResultList();
```

🔸 Criteria API

- Forma de construir consultas de forma **dinámica y segura**, con código Java:

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Producto> cq = cb.createQuery(Producto.class);
Root<Producto> root = cq.from(Producto.class);
cq.select(root).where(cb.gt(root.get("precio"), 10));
List<Producto> resultados = em.createQuery(cq).getResultList();
```

🔐 Transacciones

Una transacción agrupa operaciones que deben realizarse **todas juntas o ninguna**.

```
em.getTransaction().begin();
em.persist(obj);
em.getTransaction().commit();
```

📌 Ventajas del ORM

- Código más limpio y mantenible.
- Evita escribir SQL repetitivo.
- Permite trabajar con objetos directamente.
- Portabilidad entre bases de datos.

⚠️ Desventajas

- Puede ocultar lo que realmente pasa (sentencias SQL generadas).
- Menor control sobre optimización de consultas.
- Curva de aprendizaje inicial.

💡 Buenas prácticas

- Usar tipos `Wrapper` (`Integer`, `Long`) en lugar de primitivos.
- Controlar bien las transacciones.
- Usar `fetch` adecuado en relaciones (`LAZY` o `EAGER`).
- Revisar y optimizar las consultas si hay problemas de rendimiento.