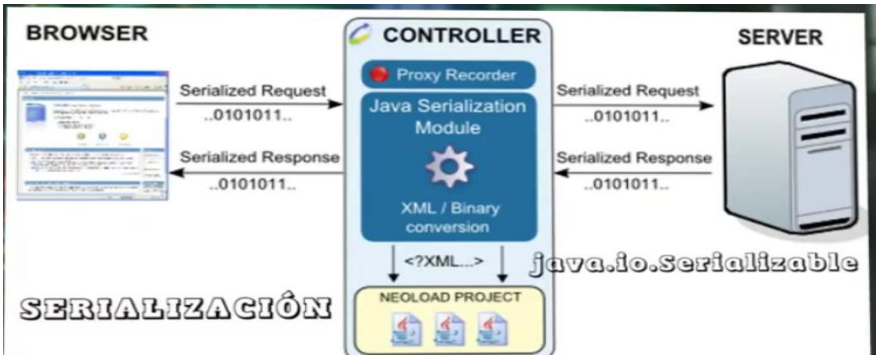
1. PERSISTIR OBJETOS

Java cuenta con varios modos:

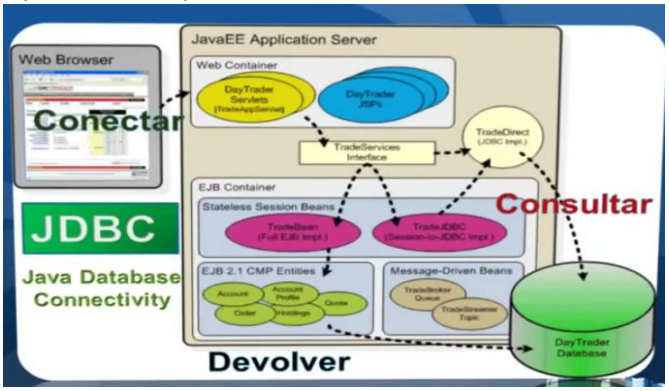
a. **Serialización de objetos**
b. **JDBC**
c. **Herramientas ORM**

a. **Serialización de objetos:** convertir los objetos en una secuencia de bits. Para ello hay que implementar la interface java.io.serializable



b. **Jdbc (java database connectivity)**

Api estándar para acceder a una base de datos relacional



c. **Herramientas ORM o de mapeo.**

El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

El objetivo es el delegar el acceso a bases de datos desde java en herramientas externas, en frameworks, lo que hacen es ofrecernos los datos relacionales como si fueran una vista orientada a objetos y viceversa, los objetos como si fueran una vista de datos relacionales, para conseguirlos se usan las mapping tools.

Existen varios frameswork que lo permiten:

Hibérnate, EclipseLink, JDO, Enterprice JavaBeans, EntityFramework.

Ventajas de las herramientas ORM

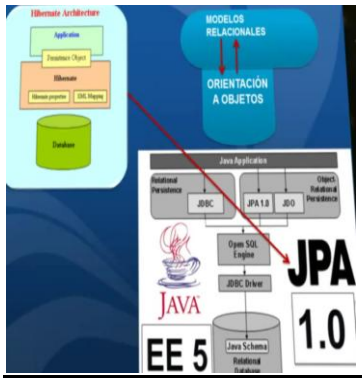
- No trabajar con filas de tablas
- Trabajar con las clases diseñadas en su modelo del dominio
- Permitir elegir la base de datos relacional con la que queremos trabajar
- Generar automáticamente el código SQL usando un mapeo objeto-relacional, el cual se especifica en un documento xml o en anotaciones
- Permitir crear, modificar, recuperar y eliminar objetos persistentes

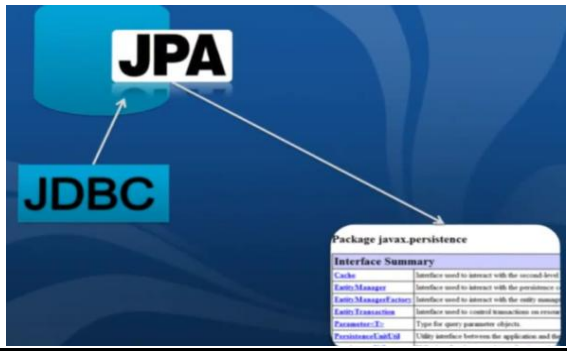
Inconvenientes

- Las aplicaciones son más lentas debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

3. JAVA PERSISTENCE API (JPA)

JPA es la API de persistencia desarrollada para la plataforma Java EE. Es un **framework** del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard (Java SE) y Enterprise (Java EE).



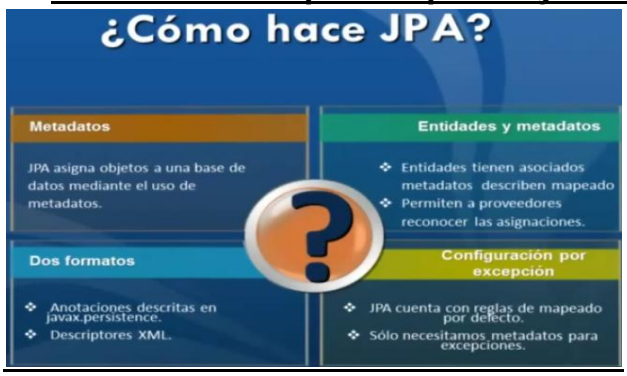


JPA es una abstracción sobre JDBC que nos permite realizar la correlación entre el sistema orientado a objetos de Java y el sistema relacional de la base de datos de forma sencilla, ya que realiza por nosotros toda la conversión entre nuestros objetos y las tablas de una base de datos. Todas las clases de ésta api están en el paquete **javax.persistence.**

Esta conversión se llama ORM (Object Relation Mapping) y se configura a través de metadatos (xml o anotaciones).

3.1 Arquitectura JPA

3.2 ¿Cómo hace JPA para mapear objetos a una BD?



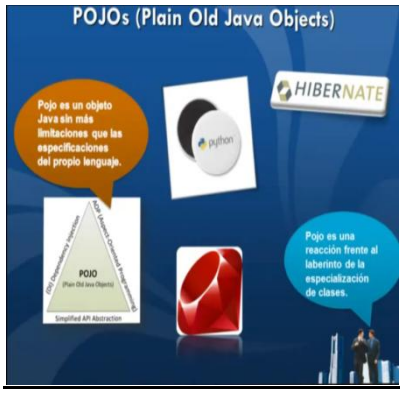
Lo hace a través de metadatos, cada entidad tiene asignados los metadatos que describen el mapeado, la asignación

3.3 COMPONENTES DE LA ARQUITECTURA JPA

a) El **ORM**, mecanismo para asignar objetos a datos almacenados en bases de datos relacionales.
b) Una **Api EntityManager** administradora de la entidad, ejecuta operaciones en la base de datos relacional como son **c**rear, leer (**r**ead), actualizar(**u**pdate) y borrar(**d**elete), se conocen como **CRUD.**
c) En tercer lugar el JPQL, lenguaje persistente de consulta java, que permite recuperar datos con un lenguaje de consulta orientado a objetos.
d) Uso de **listeners** para enlazar la lógica de negocio de un objeto persistente.

3.3.1 Entidades (ENTITY)

Las entities son simples pojos (plain old java objects) los pojo son clases simples que no dependen de un framework especial.



Si las entities son pojos, objetos java simples e independientes de cualquier plataforma, si son manejados por la EntityManager entonces tienen una identidad persistente y su estado se sincroniza con la base de datos, pero cuando no son manejados por una EntityManager, si los desconectamos de la EntityManager, pueden ser usados como cualquier otra clase de java, esto significa que las entities tienen un ciclo de vida

3.3.2 API EntityManager

- Administra la conexión con la base de datos.
- Administra la persistencia y la recuperación de entidades.
- Puede ser administrada por un contenedor JEE o por la aplicación.
- Puede administrar las transacciones o delegar en el contenedor JEE.
- Soporta la ejecución de consultas.

3.3.3 Unidad de Persistencia

Para que JPA pueda persistir esta entidad, necesita un archivo de configuración XML llamado **'persistence.xml'**, el cual debe estar ubicado en el directorio ***META-INF*** de nuestra aplicación.

La agrupación de entidades en tu aplicación se llama *unidad de persistencia.*

El archivo persistence.xml contiene información necesaria para JPA, como el nombre de la unidad de persistencia, el tipo de transacciones que vamos a utilizar (concepto que veremos a continuación), las clases que deseamos que sean manejadas por el proveedor de persistencia, y los parámetros para conectar con nuestra base de datos.

4. ANOTACIONES.

El Api JPA usa anotaciónes para definir clases que serán mapeadas a la base de datos. Las anotaciones comienzan con @.

@Entity

```
@Entity
Public class Empleado implements Serializable{}
```

En este ejemplo se creará una tabla para la entidad Empleado, por defecto el nombre de la tabla corresponde con el nombre de la clase.

@Table

Para definir los datos de una tabla de base de datos usamos la anotación @Table a nivel de declaración de la clase, así podemos definir el nombre de la tabla con la que se está mapeando la clase, el esquema, el catálogo. Si no se usa esta anotación y se usa sólo Entity, el nombre de la tabla será igual al nombre de la clase.

Para definir constraints de unicidad con la anotación @UniqueConstraint

Todas las entidades tienen que poseer una identidad que las diferencie del resto, por lo que deben contener una propiedad marcada con la anotación @Id (es aconsejable que dicha propiedad sea de un tipo que admita valores null, como Integer en lugar de int).

La identidad de una entidad va a ser gestionada por el proveedor de persistencia, así que será dicho proveedor quien le asigne un valor la primera vez que almacene la entidad en la base de datos. Para tal efecto, le añadimos a la propiedad de identidad la anotación @GeneratedValue.

@Column allows you to specify the name of the column in the database to which the attribute is to be persisted

Ejemplo: Entidad

```
import javax.persistence.*;
@Entity // obligatorio
@Table(name = "PERSONAS")
public class Persona {
   @Id // obligatorio
   @Column(name="ID_PERSONA")
   private int id;

   @Column(name="NOMBRE") // opcional
   private String nombre;

   public void setID();
   public String getID();

   public void setNombre();
   public String getNombre();
}
```