# JPA(JAVA PERSISTANCE API)

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

# Hibernate Framework

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

# ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.

# Advantages of Hibernate Framework

* Open Source and Lightweight
* Fast Performance
* Database Independent Query
* Automatic Table Creation
* Simplifies Complex Join
* Provides Query Statistics and Database Status

# JPA Creating an Entity

A Java class can be easily transformed into an entity. For transformation the basic requirements are: -

* No argument constructor
* Annotation

To transform the java class into an entity add @Entity and @Id annotation in it.

* @Entity - This is a marker annotation which indicates that this class is an entity. This annotation must be placed on the class name.
* @Id - This annotation is placed on a specific field that holds the persistent identifying properties. This field is treated as a primary key in database.

# JPA Entity Manager

Important roles of an entity manager: -

* The entity manager implements the API and encapsulates all of them within a single interface.
* Entity manager is used to read, delete and write an entity.
* An object referenced by an entity is managed by entity manager.

# Steps to persist an entity object.

1. Creating an entity manager factory object

EntityManagerFactory emf=Persistence.createEntityManagerFactory("Student\_details");

The EntityManagerFactory interface present in java.persistence package is used to provide an entity manager.

1. Obtaining an entity manager from factory.

EntityManager em=emf.createEntityManager();

1. Intializing an entity manager.

em.getTransaction().begin();

1. Persisting a data into relational database.

em.persist(s1);

1. Closing the transaction

em.getTransaction().commit();

1. Releasing the factory resources.

emf.close();

    em.close();

# Entity Operations

* Inserting an Entity
* Finding an Entity
* Updating an Entity
* Deleting an Entity

# Relationships with JPA and hibernate

* One to one mapping
* Many to one mapping
* Many to many mapping

## One to one mapping:

Annotation: @OneToOne(targetEntity=Address.class,cascade=CascadeType.ALL)

Here, we are going to perform one to one mapping by one-to-one element using annotation. In such case, no foreign key is created in the primary table.

For example, one employee can have one address and one address belongs to one employee only. Here, we are using bidirectional association.

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

## Many to one mapping:

Annotation: @ManyToOne(cascade=CascadeType.ALL)

In many to one mapping, various attributes can be referred to one attribute only.

for example, every employee has one company address only and one address belongs to many employees. Here, we are going to perform many to one mapping using annotation.

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

## Many to many mapping:

Annotation: @ManyToMany(targetEntity = Answer.class, cascade = { CascadeType.ALL })

We can map many to many relation either using list, set, bag, map etc. Here, we are going to use list for many-to-many mapping. In such case, three tables will be created.

For example, we will generate a many to many relation between questions and answers by list.so here we can have one question we can have different answers and same answer with different questions.

# Inheritance Hierarchy

* Single table inheritance:

In this strategy, all the classes in a hierarchy are mapped to a single table.

The annotation @Inheritance is used on the root entity class

* Table per class:

In this strategy, the superclass and subclasses in a hierarchy are mapped to different individual tables.

The annotation @Inheritance is used on the root entity class

* Joined:

Using the joined strategy will create a separate table for parent class and a separate table for each of its child classes

# Java Persistence Query language

JPQL is Java Persistence Query Language defined in JPA specification. It is used to create queries against entities to store in a relational database. JPQL is developed based on SQL syntax. But it won’t affect the database directly.

JPQL can retrieve information or data using SELECT clause, can do bulk updates using UPDATE clause and DELETE clause. EntityManager.createQuery() API will support for querying language.

SQL works directly against relational database tables, records and fields, whereas JPQL works with Java classes and instances

# Transaction management

Transaction Interface in Hibernate

In hibernate framework, we have Transaction interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).

A transaction is associated with Session and instantiated by calling session.beginTransaction().

The methods of Transaction interface are as follows:

**void begin()**

**void commit()**

**void rollback()**

**void setTimeout(int seconds)**

**boolean isAlive()**

**void registerSynchronization(Synchronization s)**

**boolean wasCommited()**

boolean wasRolledBack()