**Hands on 4 : Difference between JPA, Hibernate and Spring Data JPA**

**What Is Java Persistence API?**

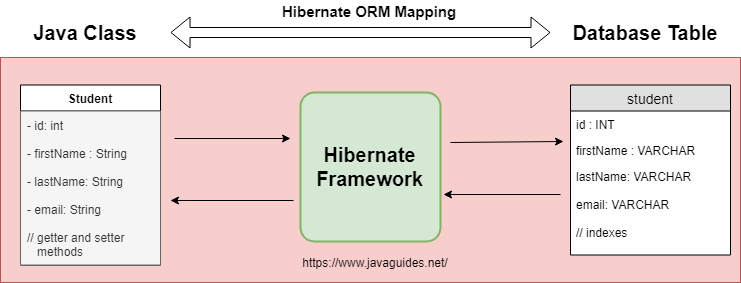
The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.

**What Is Hibernate Framework?**

Hibernate is an object-relational mapping solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.

**Example**: Below diagram shows an *Object Relational Mapping* between the **Student** Java class and the **students**tablein the database.

****

**What Is Spring Data JPA?**

Spring Data is a part of Spring Framework. The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).

**What Is the Difference Between Hibernate and Spring Data JPA?**

Hibernate is a JPA implementation, while Spring Data JPA is a JPA Data Access Abstraction.

Spring Data offers a solution to GenericDao custom implementations. It can also generate JPA queries on your behalf through method name conventions.

With Spring Data, you may use Hibernate, Eclipse Link, or any other JPA provider. A very interesting benefit is that  you can control transaction boundaries declaratively using the @Transactional annotation.

Spring Data JPA is not an implementation or JPA provider, it's just an abstraction used to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.

Remember, Spring Data JPA always requires the JPA provider such as Hibernate or Eclipse Link.

**Spring Data JPA :-**

**EmployeeRepository.java:-**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java :-**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

 No need to manage session/transactions explicitly.

 Just use save(), findById(), delete()—all built-in.

 Clean and readable.

**Hibernate (Manual way) :-**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

You manually manage Session, Transaction, Commit, Rollback.

More control, more boilerplate.

**Java Persistence API (JPA) :-**

JPA (Java Persistence API) is a **specification** provided by Java that defines a standard approach for **object-relational mapping (ORM)**—that is, how Java objects are mapped to database tables. It includes a set of **annotations** and **interfaces** to handle CRUD operations and query data using JPQL (Java Persistence Query Language). However, JPA itself **does not contain any actual implementation**; it only provides the rules and structure. To use JPA in a real-world application, you need a **JPA provider**, such as **Hibernate**, **EclipseLink**, or **OpenJPA**, which gives life to the specification by providing the underlying functionality.

**Hibernate :-**

Hibernate is a **popular open-source ORM tool** that serves as one of the main **implementations of JPA**. It allows Java developers to interact with relational databases using Java objects without writing complex SQL queries. Hibernate **implements all features of JPA** and also provides **additional advanced features** like caching, lazy loading, and better performance tuning, which go beyond the standard JPA capabilities. You can use Hibernate **with or without JPA annotations**, meaning it can act both as a JPA provider and as a standalone ORM framework. However, using Hibernate directly requires more **manual handling of sessions and transactions**, which can lead to **boilerplate code**.

**Spring Data JPA :-**

Spring Data JPA is a part of the **Spring Framework ecosystem** and provides a **higher-level abstraction over JPA**. It does not implement JPA itself but rather **works on top of an existing JPA provider** like Hibernate. The goal of Spring Data JPA is to **simplify data access layers** in Spring applications by eliminating boilerplate code such as DAO (Data Access Object) implementations. With Spring Data JPA, you can define **repository interfaces** like JpaRepository, and Spring will automatically provide the implementation at runtime. It also **manages transactions automatically** with annotations like @Transactional, and it integrates smoothly with Spring’s dependency injection, making your code cleaner, shorter, and easier to maintain.