JPA is simpler, cleaner, and less labor-intensive than JDBC, SQL, and hand-written mapping. JPA is suitable for non-performance oriented complex applications. The main advantage of JPA over JDBC is that, in JPA, data is represented by objects and classes while in JDBC data is represented by tables and records. It uses POJO to represent persistent data that simplifies database programming. There are some other advantages of JPA:

JPA avoids writing DDL in a database-specific dialect of SQL. Instead of this, it allows mapping in XML or using Java annotations.

JPA allows us to avoid writing DML in the database-specific dialect of SQL.

JPA allows us to save and load Java objects and graphs without any DML language at all.

When we need to perform queries JPQL, it allows us to express the queries in terms of Java entities rather than the (native) SQL table and columns.

Spring Boot JPA is a Java specification for managing relational data in Java applications. It allows us to access and persist data between Java object/ class and relational database. JPA follows Object-Relation Mapping (ORM). It is a set of interfaces. It also provides a runtime EntityManager API for processing queries and transactions on the objects against the database. It uses a platform-independent object-oriented query language JPQL (Java Persistent Query Language).

In the context of persistence, it covers three areas:

The Java Persistence API

Object-Relational metadata

The API itself, defined in the persistence package

**JPA is not a framework. It defines a concept that can be implemented by any framework.**

**JPA Architecture Classes:**

JPA is a source to store business entities as relational entities. It shows how to define a POJO as an entity and how to manage entities with relation.

The following figure describes the class-level architecture of JPA that describes the core classes and interfaces of JPA that is defined in the javax persistence package. The JPA architecture contains the following units:

**Persistence**: It is a class that contains static methods to obtain an EntityManagerFactory instance.

**EntityManagerFactory**: It is a factory class of EntityManager. It creates and manages multiple instances of EntityManager.

**EntityManager**: It is an interface. It controls the persistence operations on objects. It works for the Query instance.

**Entity**: The entities are the persistence objects stores as a record in the database.

**Persistence Unit**: It defines a set of all entity classes. In an application, EntityManager instances manage it. The set of entity classes represents the data contained within a single data store.

**EntityTransaction**: It has a one-to-one relationship with the EntityManager class. For each EntityManager, operations are maintained by EntityTransaction class.

**Query**: It is an interface that is implemented by each JPA vendor to obtain relation objects that meet the criteria.

**JPA Class Relationship:**

The relationship between EntityManager and EntiyTransaction is one-to-one. There is an EntityTransaction instance for each EntityManager operation.

The relationship between EntityManageFactory and EntiyManager is one-to-many. It is a factory class to EntityManager instance.

The relationship between EntityManager and Query is one-to-many. We can execute any number of queries by using an instance of EntityManager class.

The relationship between EntityManager and Entity is one-to-many. An EntityManager instance can manage multiple Entities.