Day 21- Facilitation Guide

Index

For (1.5 hrs) ILT

During the previous session, we held a practice interview on SQL.

In this session, we will delve into JPA, the concept of ORM Tool, Hibernate, and its different Annotations.

I. Introduction to JPA

JPA, or Java Persistence API, is a Java-based specification and a set of APIs that provide a standard way to interact with relational databases in Java applications. JPA is part of the Java EE (Enterprise Edition) platform and is used for managing and persisting Java objects into a relational database. It was introduced to simplify database access in Java applications and to promote a more object-oriented approach to data persistence.

Java Persistence API is a class and method collection for storing large volumes of data into a database administered by the Oracle Corporation.

Where to use JPA

A programmer follows the framework JPA Provider, which provides easy interaction with database instances to lessen the strain of writing codes for relational object maintenance. JPA is here to take over the necessary framework.

Advantages of JPA

The advantages of JPA are given below.

- The burden of intercommunication with the database decreases substantially using JPA.
- User programming is made easy by disguising the O/R mapping and processing of database access.
- By employing annotations, the cost of generating a definition file is reduced.
- We can integrate apps with other JPA providers.
- The standard implementation features that can be part of JPA's specification can be added using multiple implementations.

Disadvantages of JPA

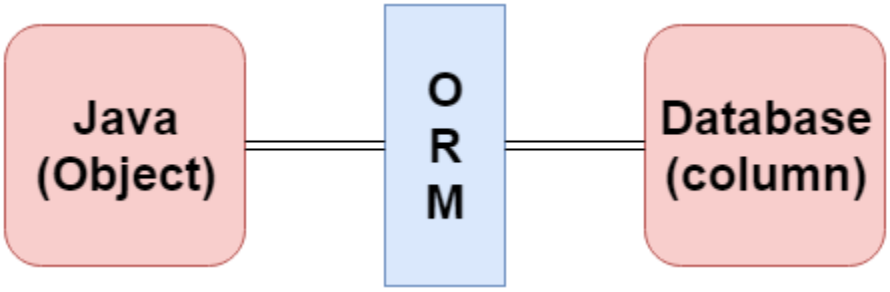
Below are the problems that occur in JPA are:

- Composite Key is the biggest problem of the JPA developers. When we map a composite key, we add huge complexity to the project.
- When we need to use stored procedures or Functions, a project with many business rules in the database can be challenging.
- Slow processing and a lot of RAM is occupied.
- Sometimes, JPA degrades its performance during reporting, including many entities or issues with a transaction.

Now that we have gained insight into JPA, let's delve into the concept of the ORM tool.

II. Introduction to ORM Tool

Object Relational Mapping (ORM) is a functionality which is used to develop and maintain a relationship between an object and relational database by mapping an object state to a database column. It is capable of handling various database operations easily such as inserting, updating, deleting etc.



ORM Frameworks

Following are the various frameworks that function on ORM mechanism: -

- Hibernate
- TopLink
- ORMLite
- iBATIS
- JPOX

Mapping Directions

Mapping Directions are divided into two parts: -

- **Unidirectional relationship -** In this relationship, only one entity can refer the properties to another. It contains only one owing side that specifies how an update can be made in the database.
- **Bidirectional relationship -** This relationship contains an owning side as well as an inverse side. So here every entity has a relationship field or refers the property to another entity.

Types of Mapping

Following are the various ORM mappings: -

- **One-to-one -** This association is represented by @OneToOne annotation. Here, an instance of each entity is related to a single instance of another entity.
- **One-to-many -** This association is represented by @OneToMany annotation. In this relationship, an instance of one entity can be related to more than one instance of another entity.

- **Many-to-one -** This mapping is defined by @ManyToOne annotation. In this relationship, multiple instances of an entity can be related to single instance of another entity.
- **Many-to-many -** This association is represented by @ManyToMany annotation. Here, multiple instances of an entity can be related to multiple instances of another entity. In this mapping, any side can be the owning side.

Note: We will learn Mapping in the upcoming session.

Now that we've acquired an understanding of ORM tools, it's important to note that JPA, being a specification, does not perform operations on its own. It necessitates an implementation. This is where ORM tools like Hibernate, TopLink, and iBatis come into play as they implement JPA specifications for data persistence.

Knowledge check…

Now that we've covered JPA introduction and understanding of ORM tool, it's time to test your understanding. The Trainer will conduct a short poll quiz to assess your knowledge on these topics.

1.What does JPA stand for?

A. Java Persistence Association
B. Java Persistence API
C. Java Persistence Framework
D. Java Persistence Object

2.What is the primary purpose of an ORM tool like Hibernate?

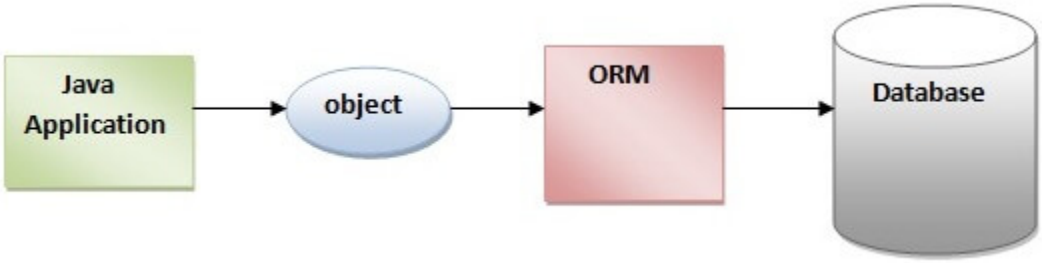
A. To manage HTTP requests
B. To map Java objects to database tables
C. To create user interfaces
D. To secure web applications

At the end of the quiz,the Trainer will provide feedback and explanations to the participants, enhancing the learning experience and understanding of the content.

Now, let's delve into gaining an understanding of Hibernate.

III. Introduction to Hibernate and it's state

Hibernate is a Java framework that simplifies the development of Java applications 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.



Note: The ORM tool internally uses the JDBC API to interact with the database.

Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework: first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if the database is changed for the project, we need to change the SQL query as well, which leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provides statistics about query and database status.

Need of Hibernate Framework

Hibernate is used to overcome the limitations of JDBC like:

1. JDBC code is dependent upon the Database software being used i.e. our persistence logic is dependent, because of using JDBC. Here we are inserting a record into Employee table but our query is Database software-dependent i.e. Here we are using MySQL. But if we change our Database then this query won't work.
2. If working with JDBC, changing the Database in the middle of the project is very costly.
3. JDBC code is not portable code across the multiple database software.
4. In JDBC, Exception handling is mandatory. Here We can see that we are handling lots of Exceptions for connection.
5. While working with JDBC, There is no support for Object-level relationships.
6. In JDBC, there occurs a Boilerplate problem i.e. For each and every project we have to write the below code. That increases the code length and reduces the readability.

To overcome the above problems we use ORM tool i.e. nothing but Hibernate framework.

Question: How does Hibernate simplify database access in Java applications, and what advantages does this provide to developers?

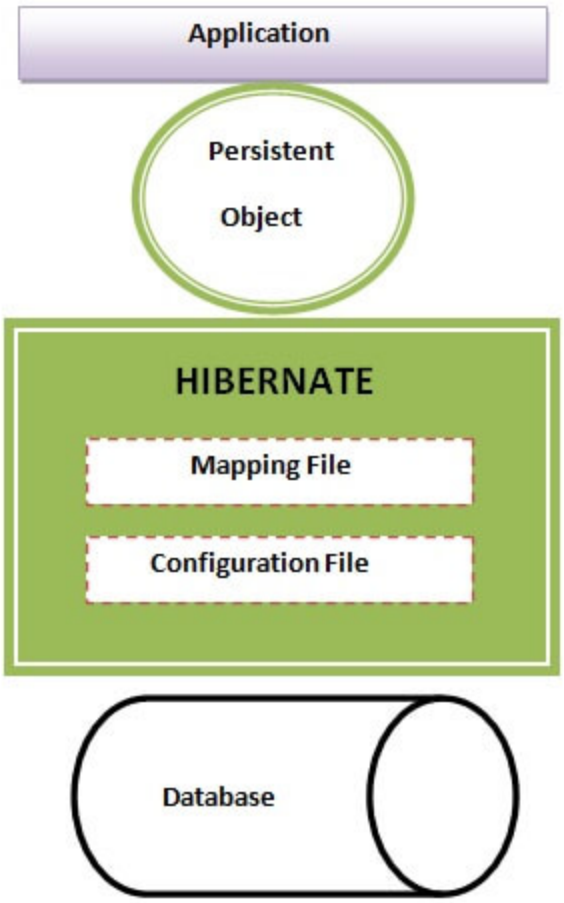
IV. Hibernate Architecture

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

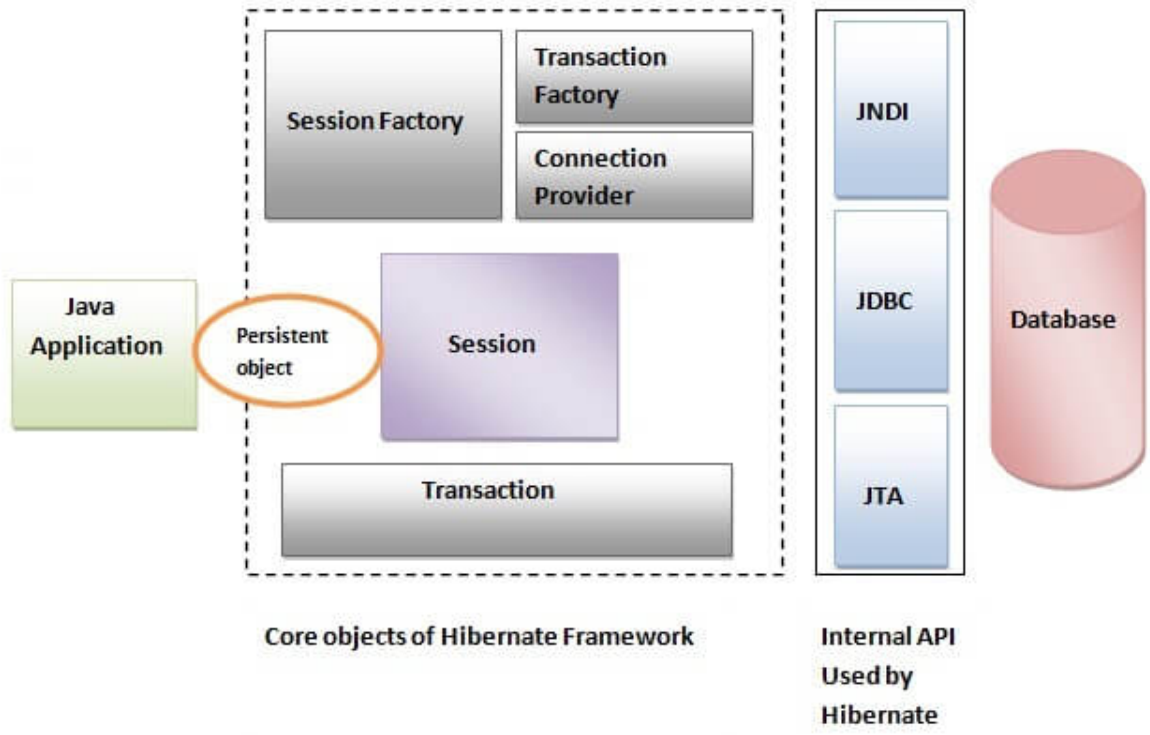
The Hibernate architecture is categorized in four layers.

- Java application layer
- Hibernate framework layer
- Backhand api layer
- Database layer

Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects such as session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds a second level cache (optional) of data. The org.hibernate.SessionFactory interface provides a factory method to get the object of Session.

Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is a factory of

Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transactions. It is optional.

Note: We will learn these Elements in the upcoming session.

V. Important Annotation

Annotations	Use of annotations
@Entity	Used for declaring any POJO class as an entity for a database
@Table	Used to change table details, some of the attributes are- • name – override the table name • schema • catalog • enforce unique constraints

@Id	Used for declaring a primary key inside our POJO class
@GeneratedValue	Hibernate automatically generate the values with reference to the internal sequence and we don’t need to set the values manually.
@Column	It is used to specify column mappings. It means if in case we don’t need the name of the column that we declare in POJO but we need to refer to that entity you can change the name for the database table. Some attributes are- ● Name – We can change the name of the entity for the database ● length – the size of the column mostly used in strings ● unique – the column is marked for containing only unique values ● nullable – The column values should not be null. It’s marked as NOT
@Transient	Tells the hibernate, not to add this particular column
@Temporal	This annotation is used to format the date for storing in the database
@Lob	Used to tell hibernate that it’s a large object and is not a simple object

@OrderBy	This annotation will tell hibernate to OrderBy as we do in SQL. For example – we need to order by student firstname in ascending order @OrderBy(“firstname asc”)

Food for thought..

The trainer can ask the students the following question to engage them in a discussion:

Why are annotations used in Hibernate?

VI. Difference between JPA and Hibernate

JPA (Java Persistence API) and Hibernate are related, but they serve different roles in the context of Java-based data persistence and object-relational mapping (ORM). Here are the key differences between JPA and Hibernate:

Definition and Role:

JPA: JPA is a specification that defines a set of standard interfaces and annotations for Java-based ORM. It is part of the Java EE (Enterprise Edition) platform and provides a standard way to interact with relational databases.

Hibernate: Hibernate is an ORM framework that implements the JPA specification along with its own proprietary features. It is a specific implementation of JPA and provides additional functionality beyond what JPA mandates.

Standard vs. Implementation:

JPA: JPA is a standard, and multiple ORM frameworks can implement it. It provides a common API that allows developers to write portable, vendor-independent code that can work with different JPA-compliant implementations.

Hibernate: Hibernate is a concrete implementation of JPA. While it follows the JPA standard, it also provides Hibernate-specific features and extensions. This means that Hibernate offers features that go beyond the standard JPA specification.

Portability:

JPA: JPA aims for portability, allowing you to switch between different JPA implementations (such as Hibernate, EclipseLink, and OpenJPA) without changing your application code. This makes it a good choice for applications that may need to work with different databases.

Hibernate: While Hibernate is JPA-compliant and offers portability, it also has Hibernate-specific features and configurations that can make your code less portable if you heavily rely on those features.

Configuration and Flexibility:

JPA: JPA typically requires less configuration and is well-suited for developers who want to adhere strictly to the JPA standard without delving into implementation-specific details.

Hibernate: Hibernate provides more configuration options and features beyond the JPA standard. This gives you greater flexibility and control but can be more complex to configure and may require more knowledge of Hibernate-specific concepts.

Learning Curve:

JPA: JPA is generally easier to learn for beginners due to its standardized approach. It provides a good starting point for those new to ORM.

Hibernate: Hibernate has a steeper learning curve, especially if you want to make use of its advanced features. It can be more suitable for experienced developers who require fine-grained control over the ORM process.

In summary, JPA is a standard API for Java-based ORM, designed to promote portability and provide a common interface for working with databases. Hibernate, on the other hand, is a popular and feature-rich ORM framework that implements JPA but also offers its own features and extensions.

Question: What is the main role of JPA in the world of Java-based data persistence, and how does it relate to Hibernate?

Exercise:

Use ChatGPT to explore Hibernate:

Put the below problem statement in the message box and see what ChatGPT says.

I was learning hibernate and I was wondering how can I write database independant query using hibernate.