www.acuityeducare.com

Acuity Educare

ENTERPRISE JAVA
SEM : V

SEM V: UNIT 5

ACUITY EDUCARRE PVT. LTD.
Caring for Education...always!

607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622

YouTube
Abhay More

Telegram
abhay_more

Q.1 What is Persistence? Explain with an example.

Persistence is one of the fundamental concepts of application development. It allows DATA to outlive the execution of an application that created it. It is one of the most vital pieces of an application without which all the data is simply lost. Majority of applications use persistent data. For example, GUI applications need to store user preferences across program invocations, Web applications track user movements and orders over long periods of time. Hence, it is imperative to choose an appropriate persistence data store. Often when choosing the persistence data store the following fundamental qualifiers are considered:
1. The length of time data must be persisted
2. The volume of data
Example:

```
<%@page import="org.hibernate.*, org.hibernate.cfg.*, mypack.*" %>
<%! SessionFactory sf;

org.hibernate.Session hibSession; %>

<%
sf = new Configuration().configure().buildSessionFactory();
hibSession = sf.openSession();
Transaction tx = null;
GuestBookBean gb = new GuestBookBean();
try{
tx = hibSession.beginTransaction();
String username = request.getParameter("name");
String usermsg = request.getParameter("message");
String nowtime = ""+new java.util.Date();
gb.setVisitorName(username);
gb.setMsg(usermsg);
gb.setMsgDate(nowtime);
hibSession.save(gb);
tx.commit();
out.println("Thank You for your valuable feedback....");
}catch(Exception e){out.println(e);}
hibSession.close();
%>
```

Q. 2 With the help of a diagram explain JPA architecture.

1. Persistence: The javax.persistence.Persistence class contains static helper methods to obtain EntityManagerFactoryinstances in a vendor-neutral fashion.

2. EntityManagerFactory: The EntityManagerFactory is created with the help of a Persistence Unit during the application start up. It serves as a factory for spawning EntityManager objects when required. Typically, it is created once [one EntityManagerFactory object per database] and for later use kept alive for later use.The javax.persistence.EntityManagerFactory class is a factory for EntityManagers.

3. EntityManager: The EntityManager object [javax.persistence.EntityManager] is lightweight and inexpensive to create. It provides the main interface to perform actual database operations. All the POJOs i. e. persistent objects are saved and retrieved with the help of EntityManager object. Typically, EntityManager objects are created as needed and destroyed when not required. Each

YouTube - Abhay More | Telegram - abhay_more
607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622

EntityManager manages a set of persistent objects and has APIs to insert new Objects and delete existing ones. EntityManagers also act as factories for Query instances and CriteriaQuery instances.

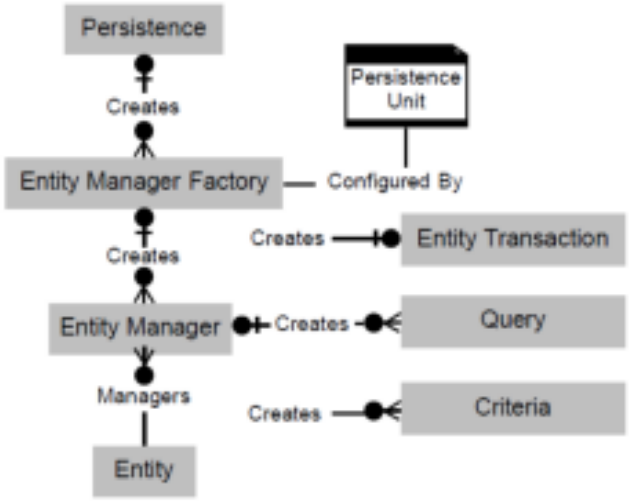
4. Entity: Entites are persistent objects that represent datastore records.

5. Entity Transaction: A Transaction represents a unit of work with the database. Any kind of modifications initiated via the EntityManager object are placed within a transaction. An EntityManager object helps creating an EntityTransaction object. Transaction Objects are typically used for a short time and are closed by either committing or rejecting.

6. Query: Persistent objects are retrieved using a Query object. Query objects [javax.persistence.Query] allows using SQL or Java Persistence Query Language [JPQL] queries to retrieve the actual data from the database and create objects.

7. Criteria: Criteria API IS a non-string-based API for the dynamic construction of object-based queries [javax.persistence.criteria]. Just like JPQL static and dynamic queries, criteria query objects are passed to the EntityManager'screateQuery() method to create Query objects and then executed using the methods of the Query API. A CriteriaQuery object can be thought of as a set of nodes corresponding to the semantic constructs of the query:

- Domain objects, which correspond to the range variables and other identification variables of the JPQL FROM clause
- WHERE clause predicates, which comprise one or more conditional expression objects
- SELECT clauses, which comprise one or more select item objects
- ORDER-BY and GROUP-BY items
- Subqueries



YouTube - Abhay More | Telegram - abhay_more
607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622

Q. 3 What is Hibernate? Explain the need of using it.

Any project that requires database interaction have started looking at ORM tools than considering the traditional approach i.e. JDBC. Hibernate ease the job of programmer in working with traditional database using concrete SQL queries in Java by using java object mapped with data base and allow programmer to interact with database just like other java class or object. The objective of Hibernate is to free the programmer from tedious database interaction and focus on working with java objects and features of application instead of worrying about how to work with data from database. Hibernate does this by copying data from database table to java class and saving state of an object to database table. Hibernate is a free, open source, high performance persistent ORM and query tool. Gavin King, founded the Hibernate Project in 2001 at JBoss Inc. [now part of RedHat Inc.]
Hibernate provides:

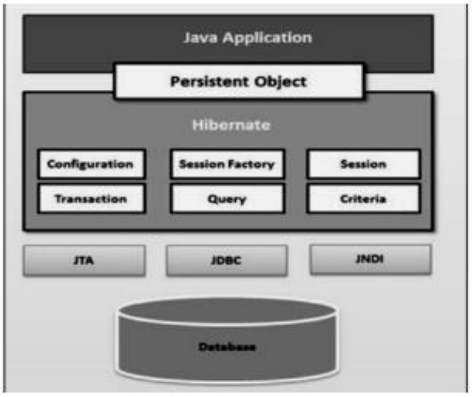
- Mapping of java classes to Database table.
  o e.g.: Student.java class → Student table in Oracle
- Mapping of java data type to SQL data type.
  o e.g.: int → number, String → varchar, java.sql.Date → DateTime, etc.
- Flexibility in Querying and retrieving data from any database.
- Freedom to switch to any data base without changing the application logic/presentation logic.

Q. 4 What are the different components of Hibernate?

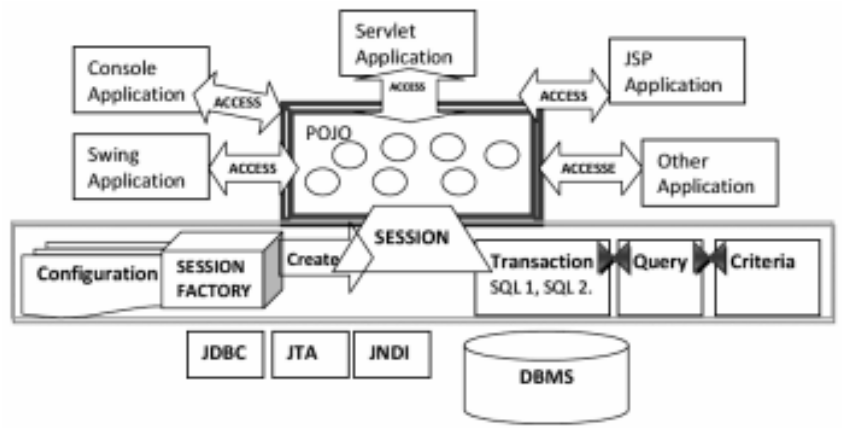
Components of Hibernate:
The main components of Hibernate are:
1. Connection Management: Hibernate solves the problems which arise when a relational database is connected by an application written in object oriented programming language style, due to data type differences, manipulative differences, transactional differences, structural and integrity differences. Connection Management provides efficient connection management and removes the overhead of database interaction from application program.
2. Transaction Management: Transaction in hibernate is managed by JTA and JDBC. It allow to fire more than one SQL query at a time.
3. Object Relation Management: It is used to map java objects to database tables. Hibernate stores the persistent objects in session and reads the state of an object to execute appropriate database query.

Q. 5 Write a short note on hibernate architecture.

Hibernate is a layered architecture. The main components are Configuration, Session Factory, Session, Transaction, Query and Criteria. Hibernate uses existing Java APIs, like JDBC for database connectivity, Java Transaction API(JTA) for transaction and Java Naming and Directory Interface (JNDI) for easy integration with other enterprise application



YouTube - Abhay More | Telegram - abhay_more
607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622



Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction offunctionality common to relational databases, allowing almost any database with a JDBC driver to supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.
Configuration: It represents properties/configuration of a hibernate application. It the first object created in a hibernate application and created once at the time of application execution. This object reads the configuration file to establish database connection and mapping. This object helps in creating session factory. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components:

1. Database Connection: This is handled through one or more configuration files supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.
2. Class Mapping Setup This component creates the connection between the Java classes and database tables.

Session Factory: It is created using configuration object at the time of application startup to serve as a base for creating light weight sessions conveniently during client's request. One session factory is created for one database for multiple database multiple session factory objects are created. **Session Object:** Sessions are single threaded, lightweight objects to communicate with database represented by session class from org.hinernate package. Persistent object are created, saved and retrieved using session object during client interaction. It wraps the Connection class from java.sql package and serves as factory for Transaction. The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.
Transaction: Transaction is a single threaded object used by application to represent group of SQL queries to form a unit of work called transaction. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA). All changes during a session are placed within transaction. Transactions are either completed using commit or cancelled using rollback. The org.hibernate.Transaction interface provides methods for transaction management.
Query: It uses either conventional SQL or Hibernate Query Language (HQL) to communicate with database. It associates the query parameters, restricts the results coming from database and executes queries. Persistent objects are retrieved using query object.
Criteria: Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

YouTube - Abhay More | Telegram - abhay_more
607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622

Q. 6 Write a JSP code to add visitor's feedback using Hibernate in FeedBack table in database.

```
<%@page import="org.hibernate.*, org.hibernate.cfg.*, mypack.*" %>
<%! SessionFactory sf;
 org.hibernate.Session hibSession; %>

 <%
 sf = new Configuration().configure().buildSessionFactory();
 hibSession = sf.openSession();
 Transaction tx = null;
 GuestBookBean gb = new GuestBookBean();
 try{
 tx = hibSession.beginTransaction();
 String username = request.getParameter("name");
 String usermsg = request.getParameter("message");
 String nowtime = ""+new java.util.Date();
 gb.setVisitorName(username);
 gb.setMsg(usermsg);
 gb.setMsgDate(nowtime);
 hibSession.save(gb);
 tx.commit();
 out.println("Thank You for your valuable feedback....");
 }catch(Exception e){out.println(e);}
 hibSession.close();
 %>
```

Q. 7 Write a JSP code to add guest feedback using JPA in GuestBook table in database.

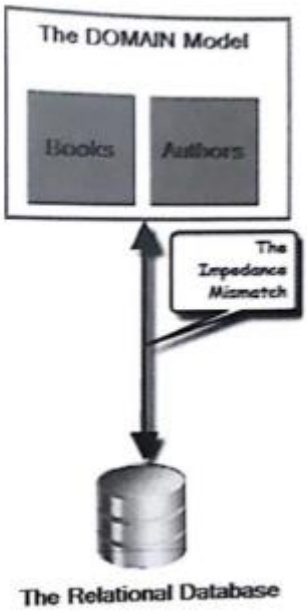
```
<%@page import="java.util.*,javax.persistence.*,mypack.GuestBook" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%!
private EntityManagerFactory entityManagerFactory;
private EntityManager entityManager;
private EntityTransaction entityTransaction;
%>
<%
entityManagerFactory = Persistence.createEntityManagerFactory("JPAApplication1PU");
entityManager = entityManagerFactory.createEntityManager();
String submit = request.getParameter("btnSubmit");
try {
String guest = request.getParameter("guest");
String message = request.getParameter("message");
String messageDate = new java.util.Date().toString();
GuestBook gb = new GuestBook();
gb.setVisitorName(guest);
gb.setMessage(message);
gb.setMessageDate(messageDate);
entityTransaction = entityManager.getTransaction();
entityTransaction.begin();
entityManager.persist(gb);
entityTransaction.commit();
} catch (RuntimeException e) {
if(entityTransaction != null) entityTransaction.rollback();
```


YouTube - Abhay More | Telegram - abhay_more
607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622

```
throw e; }
try {
guestbook = entityManager.createQuery("SELECT g from GuestBook
g").getResultList();
} catch (RuntimeException e) { }
entityManager.close();%>
```

Q. 8 What is Impedance Mismatch? How it can be solved?

The object oriented [domain] model use classes, whereas the relational databases use tables. This creates a gap [The Impedance Mismatch]. Getting the data and associations from objects into relational table structure and vice versa requires a lot of tedious programming due to the difference between the two. This difference is called The Impedance Mismatch.

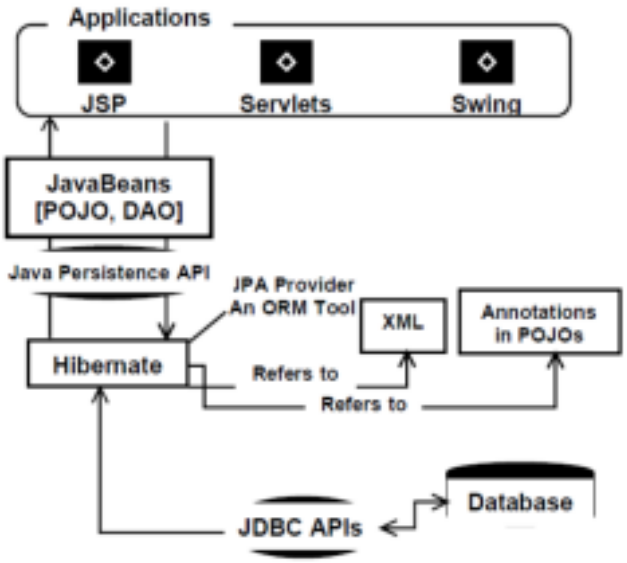


The Impedance Mismatch:
This issue arises as the design of relational data and object-oriented instances share a very different relationship structure within their respective environments. Relational databases are structured in a tabular manner and the object-oriented instances are structured in a hierarchical manner. This means that in this object-oriented world, data is represented as OBJECTS [often Called DOMAIN model]. However, the storage medium is based on a RELATIONAL paradigm. Hence, there exists an inevitable mismatch, the so-called Object/Relational Impedance Mismatch which creates a vacuum between the Object-Oriented Model of a well-designed application [the DOMAIN model] and the relational model in a database schema. This vacuum is surprisingly wide.

YouTube - Abhay More | **Telegram** - abhay_more
607A, 6th floor, Ecstasy business park, city of joy, JSD road, mulund (W) | 8591065589/022-25600622

Q. 9 What is the relationship between JPA, ORM, Database and the application?

JPA is made up of a few classes and interfaces. The application communicates with the configured IPA provider [in this case EclipseLink] to access the underlying data. Typically, applications invoke the appropriate methods of the Java Persistence API. These methods are passed the persistence objects and instructed to operate upon them. The information about the mapping [metadata] between the instance variables of classes and the columns of the tables in the database is available either in XML and/ or POJOs with the help of Annotations. POJOs are Java classes that represent the tables in the database. Data Access Object [DAO] is the design pattern that can be used [if required] to deal with database operations.'
EclipseLink uses the database [using JDBC API internally] and refers to the metadata to provide persistence services [and Persistent Objects] to the application. The application talks to EclipseLink via the JPA to perform the SELECT, INSERT, UPDATE and DELETE operations on the database tables. The ORM tool automatically creates the required SQL queries and fires them using the JDBC APIs.



Q. 10 Write short note on Functions in JPQL and Downcasting in JPQL.

Functions in JPQL:
JPA supports a set of database functions which you can use to perform small operations and transformations within a query. This is often easier and faster than doing it in the Java code. SQL supports a lot more database functions than JPQL, Specific database vendors also provide their own set of functions. Users and libraries can also define their own database functions. JPA 2.0 JPQL provided no mechanism to call database specific functions.
JPA 21 introduced function() to call database functions which are not directly supported by the standard. The syntax is very easy. You need to provide the name of the function as the first parameter and then all parameters of the custom function. In following example, the name of the function is "calculate" and I provide the numbers 1 and 2 as parameters.
Example:
Author a = em.createQuery("SELECT a FROM Author a WHERE a.id = function('calculate', 1, 2)", Author.class).getSingleResult();

Downcasting in JPQL:
JPQL will be extended to cast in the FROM and WHERE clause. The format of this will use the keyword "TREAT" and be part of the join clause. The operator TREAT can be used to cast an entity to its subclass value, i.e. downcast related entities with inheritance. This is typically used in JOIN queries

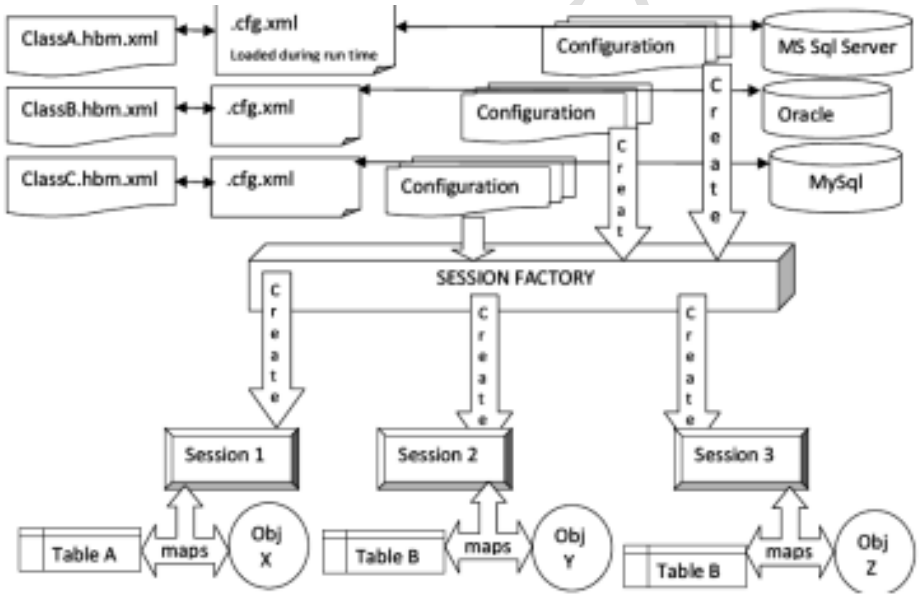
where we want to impose a restriction in WHERE clause involving a subclass field. The following is an example:

Example:
select e from Employee e join TREAT(e.projects AS LargeProject) lp where lp.attribute = value

Q. 11 How hibernate works? / Explain the modus operandi behind hibernate application.

• All configuration files hibernate.cfg.xml are created to describe about the java classes and there mapping with database tables.
• At the time of application startup these files are compiled to provide hibernate framework with necessary information.
• This dynamically builds java class objects by mapping them to appropriate database table.
• A session factory object is created from compiled collections of mapping documents.
• Session Factory spawns a lightweight session to provide interface between java objects and applications.

• Database communication is performed by this session using hibernate API used to map the changes from java object to database table and vice versa.



Q. 12 Explain the structure of hibernate.cfg.xml.

Hibernate requires to know in advance that, where to find the mapping information that defines how your Java classes relate to the database tables. Hibernate also requires a set of configuration settings related to database and other related parameters. All such information is usually supplied as a standard Java properties file called hibernate.properties, or as an XML file named hibernate.cfg.xml.

We will consider XMI. formatted file hibernate.cfg.xml to specify required Hibernate properties in following example. Most of the properties take their default values and it is not required to specify them in the property file unless it is really required. This file is kept in the root directory of your application's classpath.

We will be required to configure for a databases in a standalone situation following properties:

- hibernate.dialect: This property makes Hibernate generate the appropriate SQL for the chosen database.
- hibernate.connection.driver class: The JDBC driver class.

- hibernate.connection.url: The JDBC URL to the database instance hibernate.connection.username: The database username.

- hibernate.connection.password: The database password.

- hibernate.connection.pool size: Limits the number of connections waiting in the Hibernate database connection pool.

- hibernate.connection.autocommit: Allows autocommit mode to be used for the JDBC connection

If you are using a database along with an application server and JNDI, then you would have to configure the following properties:

- hibernate.connection.datasource: The JNDI name defined in the application server context, which you are using for the application.
- hibernate.jndi.class: The InitialContext class for JNDL
- hibernate.jndi.<JNDIpropertyname>: Passes any JNDI property you like to the JNDI InitialContext.
- hibernate.jndi.url: Provides the URL for JNDI.
- hibernate.connection.username: The database username.
- hibernate.connection.password: The database password.

The following sample file gives some insight of hibernate,cfg.xml file where MySQL is used as database server for the application.

```
<?xml version="1.0" encoding="utf-8"?>
 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
 <hibernate-configuration>
    <session-factory>
      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/java2novice</property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password">root</property>
      <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
      <property name="show_sql">false</property>
    </session-factory>
 </hibernate-configuration>
```