Object Relational Mapping

Introduction:-

1.Objects are the basic building blocks of software systems.

2.Software systems generally need to persist their state.

3.The Relational Data Model is the most widely used model (e.g., Oracle, MySQL, SQL Server, etc.).

4.There exists a gap between Object-Oriented Programming (OOP) and Relational Databases.

5.Hibernate is a framework that bridges this gap.

Why Use Hibernate:-

1.Provides a way to map Java objects to relational database tables.

2.Eliminates the need for complex SQL and JDBC code.

3.Supports automatic generation of SQL queries.

4.Provides data query and retrieval capabilities.

What is Hibernate:-

1.Hibernate is an ORM (Object-Relational Mapping) framework for Java applications.

2.It simplifies database interactions by mapping Java classes to database tables.

3.Uses Hibernate Query Language (HQL) for database operations, which is independent of the database being used.

4.Supports caching, lazy loading, and transaction management.

A First Example:-

1. Persist and Retrieve an Object:-

a. Persistence: Storing an object in the database.

b. Retrieval: Fetching an object from the database.

2. What Do We Need for Persistence and Retrieval?

a. A way to tell Hibernate about the database.

b. A persistent class.

c. A mapping between the persistent class and the database table(s).

d. A main program to manage Hibernate interactions.

3. Hibernate Configuration File (hibernate.cfg.xml):-

a.The file is named hibernate.cfg.xml and is placed in the classpath.

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
"http://www.hibernate.org/dtd/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/employees
        </property>
        <property name="hibernate.connection.username">
            root
        </property>
        <property name="hibernate.connection.password">
        </property>
        <property name="hibernate.connection.pool_size">
            10
        </property>
        <property name="show_sql">
            true
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property name="hibernate.hbm2ddl.auto">
            update
        </property>
        <mapping resource="employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

4. Persistence Class (Employee.java****):-

a. The class is mapped to a database table.

b. Follows the JavaBean standard (no-arg constructor, setters, and getters).

Example Employee Class

```java
package hibernatetutorial.entities;

import java.util.Date;

public class Employee {
    private String firstName;
    private String lastName;
    private Date birthDate;
    private Date hireDate;
    private char gender;

    // Getters and Setters
}
```

5. Mapping File (employee.hbm.xml):-

a. Maps class properties to database table columns.

b. Defines primary key generation strategy.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="hibernatetutorial.entities.Employee" table="employees">
        <id column="emp_no" name="employeeNumber">
            <generator class="assigned"/>
        </id>
        <property name="firstName">
            <column name="first_name"/>
        </property>
        <property name="lastName">
            <column name="last_name"/>
        </property>
        <property name="hireDate">
            <column name="hire_date"/>
        </property>
        <property name="birthDate">
            <column name="birth_date"/>
        </property>
        <property name="gender">
            <column name="gender"/>
        </property>
    </class>
</hibernate-mapping>
```

6. Main Program for Persistence:-

a. Creates a SessionFactory from configuration.

b. Opens a session.

c. Starts a transaction.

d. Creates an Employee object.

e. Saves it to the database.

f. Commits the transaction.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Employee employee = new Employee();
employee.setEmployeeNumber(199999);
employee.setBirthDate(new Date());
employee.setFirstName("Matthew");
employee.setLastName("Gheen");
employee.setHireDate(new Date());
employee.setGender('M');

session.save(employee);
tx.commit();
System.out.println("Done!");
```

Hibernate Architecture Overview:-

a. SessionFactory → Thread-safe factory for Session, maintains second-level cache.

b. Session → Single-threaded, represents DB connection, maintains first-level cache.

c. Persistent Objects → POJOs linked to Session, detached after session closes.

d. Transaction → Manages atomic work units, abstracts JDBC, JTA, CORBA.

e. ConnectionProvider → Manages JDBC connections, abstracts DataSource.

f. TransactionFactory → Creates Transaction instances, customizable.

Mapping One-to-Many in Hibernate:-

1. Steps to Implement One-to-Many:-

a. Create a Title persistent class.

b. Add a Set container in Employee to hold related Title instances.

c. Define mapping files for both Title and Employee.

d. Update the main program to manage the relationship.

2.Title Class (Persistent POJO):-

a.Implements Serializable with an inner class Id as a composite key.

```java
public class Title {
    private Date toDate;
    private Id id;

    public static class Id implements Serializable {
        private long employeeId;
        private String name;
        private Date fromDate;
    }

    // Getters and Setters
}
```

3. Adding a Container in Employee Class

```java
private Set<Title> titles;


public Set<Title> getTitles() { return titles; }
public void setTitles(Set<Title> titles) { this.titles = titles; }
```

4. Modifying Employee Mapping File

```xml
<set name="titles" order-by="column asc" lazy="false">
    <key column="employee_id"/>
    <one-to-many class="hibernatetutorial.entities.Title"/>
</set>
```

Many-to-Many Mapping in Hibernate

1. Without Additional Attributes

```
<set name="departmentsManaged" table="dept_manager">
    <key column="emp_no"/>
    <many-to-many column="dept_no" class="hibernatetutorial.entities.Department"/>
</set>
```

2. With Additional Attributes:-

a. Use an intermediate class (DepartmentManager)

```
public class DepartmentManager {
    private Date fromDate;
    private Date toDate;
    private Department department;
    // Getters and Setters
}
```

```
<set name="departmentsManaged" table="dept_manager">
    <key column="emp_no"/>
    <composite-element class="hibernatetutorial.entities.DepartmentManager">
        <property name="fromDate" column="from_date"/>
        <property name="toDate" column="to_date"/>
        <many-to-one name="department" column="dept_no"/>
    </composite-element>
</set>
```

Mapping Associations in Hibernate:-

1. One-to-One Mapping

Definition: Each entity is associated with at most one other entity.

Implementation:-

a. Use @OneToOne annotation in JPA.

b. In XML, use mapping.

```java
@Entity
public class Employee {
    @Id
    private int id;

    @OneToOne
    private Address address;
}
```

2. One-to-Many Mapping:-

Definition: A single entity is related to multiple entities.

Implementation:

a. Use @OneToMany annotation in JPA.

b. In XML, use mapping.

```java
@Entity
public class Employee {
    @OneToMany(mappedBy="employee")
    private Set<Title> titles;
}
```

3. Many-to-Many Mapping:-

Definition: Multiple entities are related to multiple entities.

Implementation:-

a. Use @ManyToMany annotation in JPA.

b. In XML, use mapping.

```java
@Entity
public class Employee {
    @ManyToMany
    private Set<Department> departments;
}
```

Mapping Inheritance in Hibernate:-

1. Single Table Inheritance:

Definition: Uses a single table for all subclasses with a discriminator column.

Implementation:

a. Use @Inheritance(strategy = InheritanceType.SINGLE_TABLE) in JPA.

b. In XML, use mapping.

Example

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="EMP_TYPE", discriminatorType=DiscriminatorType.STRING)
public class Employee { ... }

@Entity
@DiscriminatorValue("MANAGER")
public class Manager extends Employee { ... }
```

2. Table Per Class

Definition: Each class has its table with all inherited fields.

Implementation:

a. Use @Inheritance(strategy=InheritanceType.TABLE_PER_CLASS).

Example

```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Employee { ... }

@Entity
public class Manager extends Employee { ... }
```

---

3. Joined Table Inheritance

Definition: Creates separate tables for each class and links them with a foreign key.

Implementation:

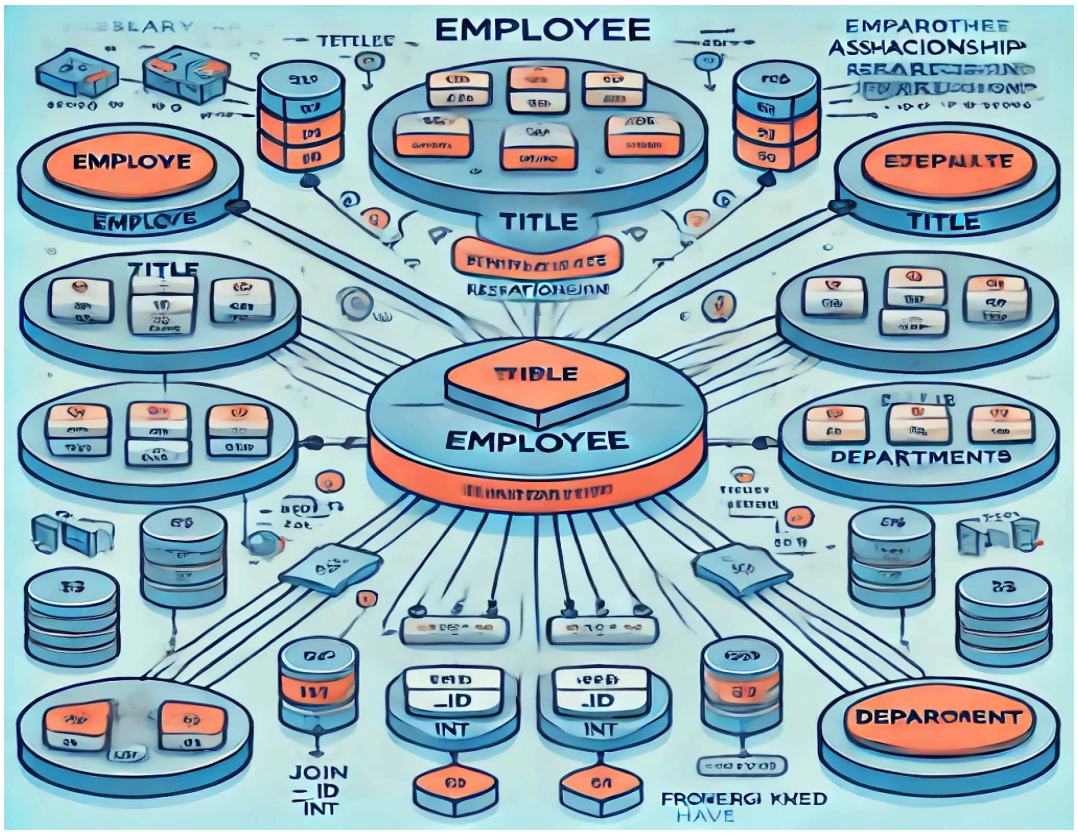
Use @Inheritance(strategy = InheritanceType.JOINED).

Example

```java
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Employee { ... }

@Entity
public class Manager extends Employee { ... }
```

ER Diagram for Mapping Associations and Inheritance



This is your structured Entity-Relationship (ER) diagram for Hibernate Mapping Associations and Inheritance, including One-to-Many, Many-to-Many relationships and Inheritance.