Hibernate

==========================================

1) What is use of show\_sql property in Hibernate configuraiton File ?

In Hibernate Applications, when we access persistence methods , Hibernate Software will take

that persistence method call and it will prepare databse dependent native sql query on the basis of

hibername mapping details and configuration details . In this context, to display the generated

database dependent sql query on console we have to use "show\_sql" property with the value

"true" in hibernate configuration file

2)Is it possible to run hibernate applicatins with out using mapping file?

Yes, it is possible to run Hibernate applications without using configuration file.

In Hibernate Applications, we are able to provide configuration details in the following two

approaches.

1) Programmatic Approach

2) Declarative Approach

a) By using properties file

b) By Using XML file

3) What is use of @Entity annotation ?

It is a class level annotation, it can be used to declare a class as an Entity class.

4)What is use of @Table annotation ?

This annotation is a class level annotation, it can be used to configure table name for the entity class. Where "name" member is able to provide the respective table name.

5)what is use of @Id ?

This annotation is Field/Method[getXXX()] level annotation, it can be used to declare a property as ID property.

6)What is use of @Column ?

This annotation is Field/Method[getXXX()] level annotation, it can be used to provide a database table column name inorder to provide mapping .

7)Is it possible to run Hibernate Applications without using configuration file?

Yes, it is possible to run Hibernate applications without using configuration file.

In Hibernate Applications, we are able to provide configuration details in the following two

approaches.

1) Programmatic Approach

2) Declarative Approach

a) By using properties file(hibernate.properties)

b) By Using XML file( hibernate.cfg.xml

8)What is Composite Keys ?

In Database applications, if one column is not sufficient to manage uniqueness as primary key

then it is possible to declare more than one column combnation as primary key, such type of

column combination as primary key is called as "Composite Key".

Note: To declare compisite keys in annotated classes , no need to use any seperate Annot

ation, simply we have to declare @Id annotation at both the columns

9)Explain Hibernate Persistence Object Lifecycle ?

1. Transient State

2. Persistence State

3. Ditached State

4. Removed State

1. Transient State

 In Hibernate Applications, when we create POJO object then automatically POJO object

will come to Transient State.

 If POJO object is available in Transient State then Hibernate Software is not aware about

POJO object

 If POJO object is available in Transient State then POJO object is representing any record

in Database table.

 In this state, Hibernate Software is unable to provide synchronization between POJO object

and record in Database.

 In this state, if we perform modifications in POJO object then that modifications are not

reflected to any record in Database.

 If we keep POJO object in Transient State for long time then POJO object is eligible for

Garbage Collection then Garbage Collector will destroy POJO object.

2. Persistence State:

 If we perform the Operations like save() , update(), saveOrUpdate(),.... over the POJO

object which is available in Transient State then POJO object will come to Persistence

State.

 In Hibernate Applications, if we perform the objects like get() , load(), find(),... automatically

Hibernate Software will create POJO object in Persistence State directly without Transient

State.

 In Persistence State, Hibernate Software is aware about POJO Object and database table

record and its synchronization.

 In Persistence State, if we perform modifications on POJO object then that modifications

are reflected to Database table record.

3. Ditached State:

 If we perform the operations like close(), clear(), evict(),.... over the Session object then

POJO object will come from Persistence State to Ditached State.

 In Ditached State, Hibernate Software is not aware about POJO object and it will not

provide synchroinization between POJO object and database table record.

 In Ditached State, POJO object is representing valid record in database table,but,

modifications on POJO object are not reflected to that record.

 In Ditached State, if we perform operations like save() , update(), saveOrUpdate(),... over

the persistence object after getting Session back then POJO object will transfer from

Ditached State to Persistence State again.

 If we keep POJO object in Ditached State for long time then POJO object is eligible for

Garbage Collection, where Garbage Collector will destroy that object.

4. Removed State:

 If we perform the operations like remove() or delete() over the POJO object in Persistence

state then POJO object will come from Persistence State to Removed State.

 In Removed State, Hibernate Software is not aware about POJO object.

 In Removed State, POJO object is not representing any record in Database table.

 In Hibernate Applications, Once if POJO object is in Removed State then it is not possible

to get back into Persistence State, directly, it must go for Garbage Collection only.

10)What is Schema Generation Tools in hibernate ?

In Hibernate applications, to perform DML operations like insert, update, delete ,... we will use the

methods lile save(), update(), delete(),... from org.hibernate.Session interface provided by

Hibernate Software.

In Hibernate applications, to perform DDL operations like create, alter, drop,... Hibernate has

provided the following implicit tools.

1. SchemaExport

2. SchemaUpdate

3. CodeGeneration

Note: The above Shema generation tools are usefull to create tables when we want to work with

unknown databases

1. Create: This option for "hibernate.hbm2ddl.auto" property will activate SchemaExport tool, it will

create a table irrespective of the table is existed or not.

EX:

1. <hibernate-configuration>

2. ----

3. <property name="hibernate.hbm2ddl.auto">create</property>

4. ----

5. </hibernate-configuration>

2. Update: This option for "hibernate.hbm2ddl.auto" property will activate SchemaUpdate tool, it

will create table if table is not existed, it will alter the table if table is existed as per the mapping

file.

EX:

1. <hibernate-configuration>

2. ----

3. <property name="hibernate.hbm2ddl.auto">update</property>

4. ----

5. </hibernate-configuration>

3. Create-Drop: This option for "hibernate.hbm2ddl.auto" will create table when SessionFactory

object is created and it will drop table when SessionFactory object is closed.

EX:

1. <hibernate-configuration>

2. ----

3. <property name="hibernate.hbm2ddl.auto>create-drop</property>

4. ----

5. </hibernate-configuration>

Note: When mapping contains new colums details when compared with database table then only

SchemaUpdate tool will perform alter operation. If Database table contains more columns when

compared with mapping file configuration details then SchemaUpdate tool will not perform alter

operation

11) Explain Primary Key Generation Algorithms in Hibernate ?

Hibernate Primary Key Generation Strategies – One-Liners

assigned

→ You manually assign the primary key value (no auto-generation).

increment

→ Hibernate uses an in-memory counter and adds +1 to last value.

sequence

→ Uses database sequence object to generate primary key (like in Oracle, PostgreSQL).

identity

→ Relies on database's auto-increment/identity column to generate IDs.

hilo

→ Uses high/low algorithm to generate IDs in memory with fewer DB hits.

native

→ Hibernate chooses the best strategy based on the underlying DB (identity/sequence).

seqhilo

→ Uses database sequence in a HiLo-style (sequence + low range).

select

→ Retrieves primary key from another existing record/table (via SELECT).

uuid

→ Generates 128-bit globally unique ID using UUID (Universally Unique Identifier).

guid

→ Similar to UUID but more specific to certain databases like SQL Server.

foreign

→ Uses primary key of another associated entity (used in one-to-one mapping).

In Hibernate (and JPA), when you annotate a field with `@GeneratedValue`, you're telling the framework to \*\*automatically generate values\*\* for that primary key field. The `strategy` member of `@GeneratedValue(strategy = …)` defines \*\*how\*\* the primary key value will be generated.

It uses the `GenerationType` enum, which has \*\*four strategies\*\*:

---

To provide a particular primary key generation alg in hibernate applications, we have to use

"strategy" member in @GeneratedValue(-) annotation, it will take either of the following constants

from "GenerationType" Enum.

1. IDENTITY

2. SEQUENCE

3. TABLE

4. AUTO

### ✅ 1. `GenerationType.IDENTITY`

\* \*\*Uses\*\*: Database's \*\*auto-increment\*\* feature.

\* \*\*Supported by\*\*: MySQL, SQL Server, PostgreSQL.

\* \*\*How it works\*\*: DB automatically generates the next value when a new row is inserted.

```java

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

```

🔸 No separate sequence or table is used.

🔸 Primary key is generated \*\*after insert\*\*.

---

### ✅ 2. `GenerationType.SEQUENCE`

\* \*\*Uses\*\*: A \*\*database sequence object\*\* to generate unique IDs.

\* \*\*Supported by\*\*: Oracle, PostgreSQL, H2, etc.

```java

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

private Long id;

```

🔸 Sequence must exist in DB or you can define one using `@SequenceGenerator`.

```java

@SequenceGenerator(name="seq", sequenceName="user\_seq", allocationSize=1)

@GeneratedValue(strategy = GenerationType.SEQUENCE, generator="seq")

```

---

### ✅ 3. `GenerationType.TABLE`

\* \*\*Uses\*\*: A separate \*\*table\*\* in the database to simulate sequence behavior.

\* \*\*Database-independent\*\* (used when sequence or identity isn’t supported).

```java

@Id

@GeneratedValue(strategy = GenerationType.TABLE)

private Long id;

```

🔸 Hibernate creates a table (e.g., `hibernate\_sequences`) to track IDs.

🔸 Slower than others but \*\*portable across all databases\*\*.

---

### ✅ 4. `GenerationType.AUTO` (default)

\* \*\*Hibernate chooses the best strategy based on the DB dialect\*\*.

\* MySQL → `IDENTITY`

\* Oracle → `SEQUENCE`

\* HSQLDB → `TABLE`

```java

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

```

🔸 Good for portability; lets Hibernate handle it.

---

### 🧠 Summary Table:

| Strategy | Source | DB Support | Notes |

| -------- | ----------------- | ------------------ | --------------------------------------- |

| IDENTITY | Auto-increment | MySQL, SQL Server | Value generated after insert |

| SEQUENCE | DB Sequence | Oracle, PostgreSQL | Most efficient if DB supports sequences |

| TABLE | Table | All (universal) | Slower, portable |

| AUTO | Hibernate chooses | Depends on DB | Auto-detects the best option |

---

Let me know if you want code examples with custom generators too.

A database sequence object is a database-level object that automatically generates a sequence of unique numeric values, often used to create primary keys for new rows.

🔹 What is a Sequence Object?

It’s like a counter stored in the database.

Every time you request a value from the sequence, it increments and returns the next number.

Commonly used to generate primary keys in systems that don’t support auto-increment (like Oracle or PostgreSQL).

✅ Example in Oracle:

sql

Copy

Edit

CREATE SEQUENCE user\_seq

START WITH 1

INCREMENT BY 1

NOCACHE;

user\_seq starts from 1

Increments by 1 each time you call it

You can use it in an insert like:

sql

Copy

Edit

INSERT INTO users (id, name) VALUES (user\_seq.NEXTVAL, 'Vishnudas');

✅ Why Sequences Are Used in Hibernate (GenerationType.SEQUENCE)

When you write this in a JPA/Hibernate class:

java

Copy

Edit

@Id

@SequenceGenerator(name = "user\_seq", sequenceName = "user\_seq", allocationSize = 1)

@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user\_seq")

private Long id;

Hibernate will internally run:

sql

Copy

Edit

SELECT user\_seq.NEXTVAL FROM DUAL; -- In Oracle

To get the next value for id.

1)what is use of Transactional ?

@Transactional annotation is used to manage transactions in Spring applications.

To ensure that a set of operations either complete successfully as a unit (commit) or are rolled back completely if an error occurs — ensuring data integrity.

2)how its work @Transactional

How It Works:

When @Transactional is placed on a method, Spring opens a transaction before the method starts and commits it after the method completes.

If an exception occurs, the transaction is rolled back automatically.

3)Explain Connection Pooling in Hibernate ?

In general, in Database related applications, if we want to perform database operations first

we have to create connection object before database operations then we have to destroy

connection object after the database operations. If we use this approach in database

related applications then application performance will be reduced, because, Creating

connection object and destroying Connection objects are two expensive processes.

In the above context, to improve application performance we have to avoid Connection

object creation and Destruction processes every time, for this, we have to use Connection

Pooling.

In Connection Pooling, first, we will create a pool object with a set of Connection objects at

application loading time , then, we get Connection object from Pool when we want to

perform database operations, after the database operations we will send Connection object

back to Pool object with out destroying Connection object.

In Hibernate applications, there are three ways to implement Connection Pooling.

1. Default Connection Pooling Mech in Hibernate

2. Third Party Vendors provided Connection pooling Mechs

3. Application Servers provided Connection pooling Mechs

4)Explain Bulk Operations ?

In General, in Hibernate Applications, by using org.hibernate.Session interface provided methods

like save (), update (), saveOrUpdate (), delete (), get () ... we are able to perform manipulations

over single record.

In Hibernate applications, if we want to perform manipulations over multiple records then we must

use the following features provided by Hibernate.

1. HQL[Hibernate Query Language]

2. Native SQL

3. Criteria API

5) Explain difference HQL vs SQL ?

+-------------------------------------+-----------------------------------------------+---------------------------------------------+

| Feature | HQL | SQL |

+-------------------------------------+-----------------------------------------------+---------------------------------------------+

| Full Form | Hibernate Query Language | Structured Query Language |

| Used In | Hibernate framework | Direct database interaction (JDBC, etc.) |

| Language Type | Object-oriented query language | Table-based relational query language |

| Focus | Java POJO classes and their properties | Database tables and columns |

| Syntax Basis | Similar to SQL, but object-oriented | SQL syntax |

| Database Dependency | Database-independent | Database-dependent |

| Entity Reference | Class names and property names | Table names and column names |

| Return Type | Collection objects (List, Set, etc.) | ResultSet object |

| Serializable Return? | Yes (Collections are Serializable) | No (ResultSet is not Serializable) |

| Query Conversion | Converted to SQL by Hibernate internally | Executed directly on the DB |

| PL/SQL/DB-specific Support | Not supported | Fully supported |

| CRUD Operations Support | Select (main), Insert/Update/Delete (from 3.x)| All CRUD operations |

+-------------------------------------+-----------------------------------------------+---------------------------------------------+

6) difference between Native SQL Queries and Scalar SQL Queries ?

+---------------------------+-------------------------------+-------------------------------------------+

| Feature | Native SQL Query | Scalar SQL Query |

+---------------------------+-------------------------------+-------------------------------------------+

| Query Type | Raw SQL | Raw SQL |

| Returns | Entities / DTOs | Raw column values (e.g., Object[]) |

| Mapping | Can use Entity class | No entity mapping; direct column output |

| Use Case | Full object fetch, joins, | Lightweight selects, specific column data |

| | procedures | |

| Performance | Slightly heavier | Faster for simple column data |

| | due to entity mapping | |

| Requires Entity Class? | Yes (if mapping to entity) | No |

+---------------------------+-------------------------------+-------------------------------------------+

Use Native SQL Query when you need raw SQL with Hibernate's session support.

Use Scalar SQL Query when you just want column values, not full entity objects.

7) what is Stored Procedures and Functions ?

frequently executed DB logic, define it as Stored Procedures or Functions in the database.

Use a Stored Procedure when you need to perform operations (like insert/update/delete).

Use a Function when you need to compute and return a value (without side effects).

8)What is use of Criterion API ?

By using Session interface methods like save(), persist(), update(), delete(),..... we are able

to perform single record manipulation, but, if we want to perform manipulations over

multiple records then we must go for HQL, Native SQL and Criterion API.

Where HQL is a powerfull, Object Oriented and Database INdependent Query language

provided by Hibernate, but, HQL is not providing environment to perform DDL kind of

operations and it is not supporting database dependent native operations like invoking

stored procedures and functions,.....

To overcome the above problem with HQL we will use "Native SQL", in case of Native SQL

, we have to write database dependent SQL queries directly, but, it is against to Hibernate,

as per the Hibernate view we must not write database dependent sql queries in JAVA

applications.

In Hibernate applications, to avoid totally query langugaes like SQL and HQL,....and to

provide the complete dPersistence logic in the form of JAVA code we must use "Criterion

API".

In the case of Criterion API, we will define persistence logic by using JAVA code only,

where the required predefined library was provided by Hibernate in the form of

"org.hibernate" package and "org.hibernate.criterion" package.

Hibernate

==========================================

1)Which technologies we should use in enterprise applications for Data Persistency ?

1) Serialization and Deserialization

2) JDBC

3) ORM Implementations

a) Hibernate

b) EJBs Entity Beans

c) JPA

d) Open JPA

e) Toplink

2) What is Granularity mismatch ?

---

### ❓ \*\*What is Granularity Mismatch?\*\*

\*\*Granularity Mismatch\*\* is a problem that happens when the \*\*level of detail\*\* (granularity) in your \*\*Java object model (OOP)\*\* is different from the level of detail in your \*\*relational database model (RDBMS)\*\*.

---

### ✅ Example:

#### 🔹 In Java (Object-Oriented Model):

You create multiple classes for better structure:

```java

class Address {

String city;

String street;

}

class User {

String name;

int age;

Address address;

}

```

#### 🔹 In Database (Relational Model):

You may have just \*\*one table\*\*:

| name | age | city | street |

|--------|-----|---------|------------|

| Rahul | 25 | Mumbai | MG Road |

Here, all fields (even address) are stored in \*\*one table\*\* without creating a separate table for `Address`.

---

### 💥 This difference causes \*\*Granularity Mismatch\*\*

- Java: fine-grained (User + Address classes)

- DB: coarse-grained (one flat table)

---

### 🔧 What problem does it cause?

1. \*\*Data Mapping Becomes Complex\*\*

You need to write extra logic to convert flat database data into nested Java objects.

2. \*\*Code Becomes Less Reusable\*\*

You cannot reuse components like `Address` easily.

3. \*\*Design Becomes Bad (Not Pure OOP)\*\*

Puts all properties into one class just to match the table.

4. \*\*Database and Code May Get Out of Sync\*\*

Difficult to keep both models updated together.

---

How Hibernate Solves It?

Hibernate gives a solution called \*\*Component Mapping\*\* using:

- `@Embeddable` → for the Address class

- `@Embedded` → in the User class

This way, Hibernate allows you to keep your \*\*Java code clean and modular\*\*, while storing everything in \*\*one table\*\* in the database.

3) Explain Sub types Mismatch ?

### ✅ What is Sub Types Mismatch (Inheritance Mismatch)?

\*\*Sub types mismatch\*\* happens when:

- In \*\*Java\*\*, we use \*\*inheritance\*\* (using `extends`) to share common code.

- But in a \*\*database\*\*, there is \*\*no inheritance\*\*, so we cannot directly represent it.

📌 This creates a \*\*mismatch\*\* between your Java code and database design.

This is called \*\*Inheritance Mismatch\*\* or \*\*Sub Types Mismatch\*\*.

---

### 🧠 Simple Java Example:

```java

class Vehicle {

String brand;

}

class Car extends Vehicle {

int noOfDoors;

}

class Bike extends Vehicle {

boolean hasKickStart;

}

```

👉 This is how we reuse code in Java — using inheritance.

---

### 💽 But in Database (Tables), we don’t have `extends`.

So we must use different tricks to represent this.

Hibernate (ORM tool) gives us 3 ways to solve this mismatch.

---

### 🛠 1. Table Per Class Hierarchy (Single Table for All)

📌 All subclasses (Car, Bike) and parent class (Vehicle) use \*\*one table\*\*.

| id | brand | noOfDoors | hasKickStart | type |

|----|-------|-----------|--------------|------|

| 1 | Honda | 4 | null | Car |

| 2 | Yamaha| null | true | Bike |

✅ Easy

❌ Many `null` values because not all fields apply to all

---

### 🛠 2. Table Per Subclass

📌 Each subclass gets its \*\*own table\*\*, and they are connected by \*\*primary/foreign keys\*\*.

\*\*Car table:\*\*

| id | brand | noOfDoors |

\*\*Bike table:\*\*

| id | brand | hasKickStart |

✅ Cleaner structure

❌ More joins needed when fetching data

---

### 🛠 3. Table Per Concrete Class

📌 Each subclass (Car, Bike) has \*\*its own table\*\* with \*\*all fields\*\* — even from the parent class.

\*\*Car table:\*\*

| id | brand | noOfDoors |

\*\*Bike table:\*\*

| id | brand | hasKickStart |

✅ No joins needed

❌ Duplicate data (brand field repeated in both tables)

---

### 🎯 Problem (Why It’s Called Mismatch):

- Java supports \*\*inheritance\*\* directly

- Relational databases \*\*do not\*\*, so we have to find a way to "simulate" it

- This difference in design is the \*\*sub type mismatch\*\* (or inheritance mismatch)

---

### ✅ Hibernate Fixes This

Hibernate gives us 3 strategies:

| Strategy Name | What It Does |

|--------------------------|----------------------------------|

| @Inheritance(strategy = InheritanceType.SINGLE\_TABLE) | One table for all classes |

| @Inheritance(strategy = InheritanceType.JOINED) | Separate tables, linked with keys|

| @Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS) | Each class has its own full table|

---

### 💡 Summary:

\*\*Sub type mismatch\*\* = Java uses inheritance, but databases don’t support it directly.

Hibernate solves this by offering 3 ways to map Java inheritance to database tables.

---

4) what is Associations Mismatch ?

Sure! Here's a very \*\*simple answer\*\* to:

---

### ✅ \*\*What is Associations Mismatch?\*\*

\*\*Associations Mismatch\*\* happens when:

- In \*\*Java (Object-Oriented Programming)\*\*, we connect classes using \*\*reference variables\*\* or \*\*collections\*\*.

- In a \*\*Database (Relational Model)\*\*, we connect tables using \*\*foreign keys\*\* or \*\*join tables\*\*.

➡️ Since Java and Databases use \*\*different ways\*\* to handle relationships, this creates a \*\*mismatch\*\* — called \*\*Associations Mismatch\*\*.

---

### 📌 Example:

#### 🔹 In Java:

```java

class Student {

Address address; // one-to-one

List<Course> courses; // one-to-many

}

```

#### 🔹 In Database:

- `Student` table uses \*\*foreign key\*\* to connect to `Address` table

- A \*\*join table\*\* (`student\_courses`) is used to connect `Student` and `Course`

---

### 💥 Problem:

Java and Database \*\*don’t match\*\* in how they represent relationships.

So it becomes difficult to save and fetch data correctly.

---

### 🛠️ \*\*How to Resolve It?\*\*

We use \*\*Hibernate\*\* (an ORM tool), which gives us \*\*Association Mapping\*\*.

It maps Java associations to database relationships using annotations like:

- `@OneToOne`

- `@OneToMany`

- `@ManyToOne`

- `@ManyToMany`

Hibernate handles all the \*\*foreign keys\*\* and \*\*join tables\*\* automatically.

---

### ✅ Final Summary:

| 🔹 Question | 🔸 Answer |

|--------------------------|----------|

| What is Associations Mismatch? | It's the difference in how Java and Database handle relationships between data |

| Why is it a problem? | It makes data saving/fetching harder and inconsistent |

| How to solve it? | Use Hibernate's \*\*Association Mapping\*\* to match Java objects with database tables automatically |

---

5) Explain Identity mismatch ?

### ✅ \*\*What is Identity Mismatch?\*\*

\*\*Identity Mismatch\*\* means:

> Java and Database have \*\*different ways to identify\*\* the same data object.

---

### 🔹 \*\*In Java (Object World)\*\*

- Java uses \*\*memory reference\*\* to identify objects.

- Even if two objects have the same data, they are \*\*not equal\*\* unless they are the \*\*same reference\*\*.

```java

Student s1 = new Student(101, "Amit");

Student s2 = new Student(101, "Amit");

System.out.println(s1 == s2); // false → because they are different objects

```

👉 Even though data is same, Java treats them as \*\*different objects\*\*.

---

### 🔹 \*\*In Database (Relational World)\*\*

- Identity is based on \*\*Primary Key\*\*.

- If two rows have the same primary key → they are considered the \*\*same record\*\*.

| id | name |

|-----|-------|

| 101 | Amit |

👉 Only \*\*one\*\* row with id=101 can exist.

---

### ⚠️ \*\*Problem (Mismatch):\*\*

- In Java → Two objects with same ID can be different

- In DB → Same ID means same record

This mismatch creates confusion when storing, updating, or comparing objects → it can lead to \*\*duplicate records\*\* or \*\*wrong updates\*\*.

---

### 🛠️ \*\*How to Resolve It?\*\*

We use \*\*Hibernate (ORM)\*\* which solves this mismatch using:

#### ✅ 1. `@Id`

- Tells Hibernate which field is the \*\*primary key\*\* (identity).

```java

@Id

private Long id;

```

#### ✅ 2. `@GeneratedValue`

- Auto-generates ID values using DB strategy (like auto-increment).

```java

@GeneratedValue(strategy = GenerationType.IDENTITY)

```

#### ✅ 3. \*\*Session Cache (First-Level Cache)\*\*

- Hibernate gives the \*\*same Java object\*\* for the same DB record (within a session).

- So Java `==` can return true because it reuses the object.

---

### 🎯 \*\*Simple Summary for Notes:\*\*

| Topic | Description |

|---------------------|-------------|

| Identity Mismatch | Java and DB use different methods to identify the same data |

| Java Identity | Based on object reference |

| DB Identity | Based on Primary Key |

| Problem | Same DB record may become multiple Java objects |

| Solution | Hibernate uses @Id, @GeneratedValue, and cache to match Java object with DB record correctly |

6)What is the difference between JPA and Hibernate?

JPA is an abstraction provided by SUN Microsystems and implemented by all Application

Server vendors like Web logic Server vendor, JBOSS Server vendor,..... JPA provides a set

of conventions to implement ORM rules and regulations.

 Hibernate is a product, it has implemented ORM rules and regulations as per JPA

guidelines in order to provide data Persistency in enterprise applications.

7) objective of Hibernate ?

To simplify data persistency in Enterprise

8)Explain Hibernate Features ?

1. Hibernate is Database independent, it can be used for any type of Database.

2. Hibernate is applicable for both standalone applications and Enterprise Applications.

3. Hibernate is providing very good support for Associations and Joins.

4. Hibernate is having very good Annotations in order to reduce XML tech dependency in

enterprise applications.

5. Hibernate is having very good implementations for Primary key generation algorithms in

order to generate and insert a unique primary key value for each and every insertion

operation.

6. Hibernate is providing very good Collections support to manage data.

7. Hibernate is having its own query language, which is database independent, Object

Oriented, that is, HQL in order to perform database operations.

8. Hibernate has provided very good Cache mechanisms in order to reuse the results.

9. Hibernate is having very good support for Connection Pooling mechanisms in order to

improve Connection re usability.

10. Hibernate is supported by almost all IDEs and Application Servers.

11. Hibernate is against for SQL Queries in enterprise applications directly, but, if we want

to write database dependent sql queries then it is possible to provide database

dependent sql queries by using "Native SQL".

12. Hibernate has provided very good transactions support.

9)what is dialect in hibernate application ?

In Hibernate configurations, "dialect" is a property, it able to provide dialect class name, it is

providing the respective database native implementations which we are using in hibernate

applications in order to prepare database dependent sql queries by Hibernate software.

10)What is the difference between save() method and persist() method?

In Hibernate applications, save() method can be used to insert a record into the Database table

and it will retuen Primary Key value of the inserted record.

public Serializable save(Object obj)throws HibernateException

In Hibernate applications, persist() method can be used to insert a record into database table and

it will not return any value.

public void persist(Object obj)throws HibernateException

To update a record ion database table we have to use the following methods.

1. update(--)

2. saveOrUpdate(--)

11)What is the difference between update() method and saveOrUpdate() method?

Where update(-) method will perform updation on a record in database table if the specified record

is existed otherwise it will rise an Exception.

public void update(Object obj)throws HibernateException

Where saveOrUpdate(-) method will insert the specified record in database table if the specified

record is not existed . If the specified record is existed in database table then it will update the

record.

public void saveOrUpdate(Object obj)throws HibernateException

12)What are the differences between get(-) method and load(-) method?

1. get() method can be used to retrive a record from database table if the record is existed. If the

required record is not existed then get() method will return null value.

public Object get(String class\_Name, Serializable pk\_Val)

public Object get(Class class\_Type, Serializable pk\_Val)

load() method can be used to retgrive a record from database table if the record is existed. If the

required record is not existed then load() method will rise an Exception like HibernateException.

public Object load(String class\_Name, Serializable pk\_Val)

public Object load(Class class\_Type, Serializable pk\_val)

2. get() method is able to perform eager or early loading, that is, it will interact with database

directly and it will retrive data and return to Hibernate application in the form of Object on the

method call.

load() method will perform Lazy or late Loading , that is, when we access load() method then a

duplicate object will be created with the primary key value without interacting with database. When

we use other properties of the Object then only it will fetch data from database table and return

that data to Java application.  
  
  
  
  
  
Question on Hibernate

=================================

1)How do you create an immutable class in hibernate?

Immutable class in hibernate creation could be in the following way. If we are using

the XML form of configuration, then a class can be made immutable by

markingmutable=false. The default value is true there which indicating that the class

was not created by default.

In the case of using annotations, immutable classes in hibernate can also be created

by using @Immutable annotation.

2)Can you explain the concept behind Hibernate Inheritance Mapping?

+--------------------------+---------------------------------------------------------+--------------------------------------------------------------+

| Strategy Name | Annotation | Description |

+--------------------------+---------------------------------------------------------+--------------------------------------------------------------+

| Single Table | @Inheritance(strategy = InheritanceType.SINGLE\_TABLE) | All classes share one table |

| Table per Class | @Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS)| Each class gets its own table |

| Joined (Normalized, Recommended) | @Inheritance(strategy = InheritanceType.JOINED) | Parent and child have separate tables, joined by a foreign key |

+--------------------------+---------------------------------------------------------+--------------------------------------------------------------+

3)what is SQL injection attack?

If the \*\*interviewer asks about SQL Injection Attack\*\*, here's how you can confidently answer in \*\*interview style\*\* with clarity:

---

### ✅ Best Answer (English + Hindi Mixed – Practical and Clear):

> \*\*SQL Injection\*\* ek \*\*security vulnerability\*\* hai jisme attacker malicious SQL code inject karta hai input fields ke through, taaki woh database ke upar \*\*unauthorized actions\*\* perform kar sake.

>

> Example ke liye, agar hum user se input le rahe hain bina validate kiye, jaise login form me:

>

> ```sql

> SELECT \* FROM users WHERE username = 'admin' AND password = ' ' OR '1'='1'

> ```

>

> To `'1'='1'` hamesha true hota hai, aur isse attacker \*\*bina valid password ke login\*\* ho sakta hai.

>

> Is attack se \*\*data leak\*\*, \*\*unauthorized access\*\*, ya even \*\*data delete\*\* bhi ho sakta hai.

>

> Isse prevent karne ke liye hum:

>

> \* \*\*Prepared Statements / Parameterized Queries\*\* ka use karte hain

> \* \*\*ORM tools\*\* jaise Hibernate, JPA ka use karte hain (ye internally safe hote hain)

> \* Input validation aur stored procedures bhi madad karte hain

---

### ✅ Short & Direct Answer (if time-limited):

> SQL Injection ek attack hai jisme malicious SQL queries input ke through inject ki jati hain taaki unauthorized database access mil sake.

> Prevention ke liye hum \*\*Prepared Statements\*\*, \*\*ORM tools\*\*, aur \*\*input validation\*\* use karte hain.

---

### ✅ Bonus Tip (If Asked for Code Example):

```java

// Unsafe

String query = "SELECT \* FROM users WHERE username = '" + user + "' AND password = '" + pass + "'";

// Safe

PreparedStatement stmt = conn.prepareStatement("SELECT \* FROM users WHERE username = ? AND password = ?");

stmt.setString(1, user);

stmt.setString(2, pass);

```

---

Let me know if you want a Hindi-only version or a Java mini project demo for SQL Injection safe login.

3) @Access Annotation in JPA (Java Persistence API)

### 🔍 `@Access` Annotation in JPA (Java Persistence API)

The `@Access` annotation (from `javax.persistence.Access`) is used to define \*\*how JPA accesses the entity data\*\* — through:

\* \*\*fields (instance variables)\*\* → `@Access(AccessType.FIELD)`

\* \*\*properties (getter methods)\*\* → `@Access(AccessType.PROPERTY)`

---

### ✅ Why Use `@Access`?

By default, JPA decides whether to use \*\*fields\*\* or \*\*getters/setters\*\* based on where you put your \*\*annotations\*\* like `@Id`.

But if you want to \*\*explicitly control\*\* how JPA reads/writes data, you use `@Access`.

---

### 🧑‍💻 Syntax:

```java

@Access(AccessType.FIELD) // or AccessType.PROPERTY

```

---

### 📌 Two Types of Access

| Access Type | Description |

| --------------------- | -------------------------------------------- |

| `AccessType.FIELD` | JPA reads/writes data using class fields |

| `AccessType.PROPERTY` | JPA reads/writes using getter/setter methods |

---

### 🧑‍💻 Example:

#### 👉 Field Access (Default)

```java

@Entity

@Access(AccessType.FIELD) // Not needed unless overriding default

public class User {

@Id

private Long id;

private String name;

// Getter and Setter

}

```

#### 👉 Property Access

```java

@Entity

@Access(AccessType.PROPERTY)

public class User {

private Long id;

private String name;

@Id

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

// name getter/setter...

}

```

---

### 🔁 Use Case: Mixed Access

You can even define different access types \*\*on a per-field basis\*\*:

```java

@Access(AccessType.FIELD)

@Entity

public class Product {

@Id

private Long id;

private String name;

@Access(AccessType.PROPERTY)

public String getName() {

// logic before returning

return name;

}

public void setName(String name) {

// validation logic

this.name = name;

}

}

```

---

### 🧠 Summary

| Keyword | Meaning |

| --------- | -------------------------------------------- |

| `@Access` | Tells JPA how to access entity data |

| Default | Field-level (if annotations are on fields) |

| Override | Use `@Access(AccessType.PROPERTY)` to change |

---

Let me know if you want a working demo project or how `@Access` works in Spring Data JPA specifically.

4) No worries! Let’s explain `@Cascade` in \*\*simple and clear Hindi + Example\*\*.

---

## 🔍 `@Cascade` क्या है?

`@Cascade` Hibernate की एक special annotation है, जो बताती है कि:

> जब आप एक \*\*parent entity\*\* (जैसे `Customer`) पर कोई operation (save, delete) करते हैं,

> तो वही operation \*\*related child entity\*\* (जैसे `Order`) पर \*\*automatically apply\*\* हो।

---

### 📦 Example: Customer और Orders

Imagine you have a `Customer` और उसके कई `Order` हैं:

```java

@Entity

public class Customer {

@Id

private Long id;

@OneToMany

@Cascade(org.hibernate.annotations.CascadeType.ALL)

private List<Order> orders;

}

```

इसका मतलब:

➡ जब आप `customer` को \*\*save\*\* करोगे, तो उसके साथ जुड़े हुए सारे `orders` भी \*\*automatically save\*\* हो जाएंगे।

➡ जब आप `customer` को \*\*delete\*\* करोगे, तो उसके orders भी \*\*delete\*\* हो जाएंगे।

---

### 🔁 Without Cascade क्या होता?

अगर आप cascade नहीं लगाओगे, तो:

\* आपको manually पहले child entities (orders) को save करना पड़ेगा

\* फिर parent entity (customer) को

---

### ✅ Hibernate-specific Cascade Types:

| Type | Use Case |

| ------------------ | ---------------------------------------- |

| `SAVE\_UPDATE` | Save ya update child entity |

| `DELETE` | Delete child automatically |

| `ALL` | All operations (save, delete, etc) apply |

| `MERGE`, `REFRESH` | Merge or reload child entity |

---

### 🚫 Warning:

`@Cascade` सिर्फ \*\*Hibernate\*\* में काम करता है.

अगर आप \*\*JPA standard\*\* use कर रहे हो, तो आपको ये लिखना चाहिए:

```java

@OneToMany(cascade = CascadeType.ALL)

private List<Order> orders;

```

---

### 🔁 Short Summary in Hindi:

| Term | Meaning in Hindi |

| ----------------- | ------------------------------------------------------------- |

| `@Cascade` | Hibernate की annotation – parent ke action child pe lagaye |

| `CascadeType.ALL` | Save, update, delete – सब कुछ child entity पर भी होगा |

| कब Use करें? | जब aapko automatically child records bhi save/delete करने हैं |

---

### ❓ Example in Use:

```java

Customer c = new Customer();

Order o1 = new Order();

Order o2 = new Order();

c.setOrders(Arrays.asList(o1, o2));

session.save(c); // orders bhi save ho jaayenge automatically if @Cascade is used

```

---

Aap batao — kya aapko Hibernate + JPA dono ke examples chahiye? Main ek simple working project structure bhi de sakta hoon.

4) Composite Keys:

In Database applications, if one column is not sufficient to manage uniqueness as primary key

then it is possible to declare more than one column combnation as primary key, such type of

column combination as primary key is called as "Composite Key".

sql>create table emp1(ENO number, ENAME varchar2(10), ESAL float, EADDR varchar2(10),

primary key(ENO,ENAME));

sql>commit

To rerpesent Composite keys in Hibernate applications we have to use the following xml tags in

mapping file.

1. <hibernate-mapping>

2. <class name="--" table="--">

3. <composite-id>

4. <key-property name="--" column="--"/>

5. -----

6. </compisite-id>

7. ----

8. </class>

9. </hibernate-mapping>

 Where <composite-id> tag is representing composite key configuration in mapping file.

 Where <key-property> tag is able to represent single ID or column configuration as part of

composite key.

 Where "name" attribute will take id property name and column attribute will take "database

table column name".

EX:Employee.hbm.xml

1. <hibernate-mapping>

2. <class name="com.durgasoft.hbn.pojo.Employee" table="emp1">

3. <composite-id>

4. <key-property name="eno" column="ENO"/>

5. <key-property name="ename" column="ENAME"/>

6. </composite-id>

7. <property name="esal" column="ESAL"/>

8. <property name="eaddr" column="EADDR"/>

9. </class>

10. </hibernate-mapping>

11.

12. Note: To declare compisite keys in annotated classes , no need to use any seperate Annot

ation, simply we have to declare @Id annotation at both the columns.

13.

14. EX: Employee.java

15. -----------------

16. @Entity

17. @Table(name="emp1")

18. public class Employee implements Serializable {

19. @Id

20. @Column(name="ENO")

21. private int eno;

22. @Id

23. @Column(name="ENAME")

24. private String ename;

25. @Column(name="ESAL")

26. private float esal;

27. @Column(name="EADDR")

28. private String eaddr;

29. setXXX() and getXXX()

30. }

5) Prepare Client Application:

The main intention of Hibernate Client application is to activate Hibernate Software, creating

persistence objects and performing Persistence operations.

To prepare Client Application in hibernate applications we have to use the following steps as per

Hiberante3.x version.

1. Create Configuration class object

2. Create Session Factory object

3. Create Session Object

4. Create Transaction object if it is required.

5. Perform Persistence operations

6. Close Session Factory and Session objects.

1. Create Configuration class object

In Hibernate, the main intention of Configuration object is to store all the configuration details

which we provided in hibernate configuration file.

To represent Configuration object Hibernate has provided a predefined class in the form of

"org.hiberante.cfg.Configuration".

To create Configuration class object we have to use the following constructor from Configuration

class.

public Configuration()

EX: Configuration cfg = new Configuration();

If we use the above instruction in Hibernate applications then we are able to get an empty

Configuration object in heap memory, it will not include any Configuration details.

If we want to store Configuration details from Configuration file we have to use either of the

following methods.

1. public Configuration configure()

This method will get configuration details from the configuration file with the name

hibernate.cfg.xml

2. public Configuration configure(String config\_file\_Name)

This method can be used to get configuration details from hibernate configuration file with any

name, it will be used when we change configuration file name from hibernate.cfg.xml file to some

other name .

3. public Configuration configure(File file)

This method can be used to get configuration details from a file which is represented in the form of

java.io.File class object.

public Configuration configure(URL url)

This method can be used to get Configuration details from a file which is available in network

represented in the form of java.net.URL .

EX: Configuration cfg = new Configuration();

cfg.configure();

When we use configure() method then Hibernate Software will search for hibernate.cfg.xml file, if it

is available then Hibernate software will load the content of hibernate.cfg.xml file, parse it and

read content from configuration file to Configuration object.

2. Create Session Factory object:

In Hibernate, the main intention of Session Factory object is to manage Connections, Statements,

Cache levels, .... and it able to provide no of Hibernate Session objects.

To represent Session Factory object Hibernate has provided a predefined interface in the form of

"org.hibernate.Session Factory".

To get Session Factory object we have to use the following method from Configuration class.

public Session Factory buildSessionFactory()

EX: SessionFactory sf = cfg.buildSessionFactory();

Note: The above approach to get Session Factory object is available upto Hibernate3.x version,

buildSessionFactory() was deprecated in Hibernate4.x version.

In Hibernate applications, if we use multiple Databases then we have to prepare multiple

Configuration files, multiple Configuration Object, w.r.t this, we have to prepare multiple Session

Factory objects.

Session Factory object is heavy weight and it is thread safe upto a particular Database, because,

it able to allow more than one thread at a time.

3. Create Session Object:

In Hibernate, for each and every database interaction a separate Session will be created.

In Hibernate, Session is able to provide no of persistence methods in order to perform persistence

operations.

To represent Session object, Hibernate has provided a predefined interface in the form of

"org.hibernate.Session".

To get Session object, we have to use the following method from Session Factory.

public Session openSession()

EX: Session s =sf.openSession();

In Hibernate, Session object is light weight and it is not thread safe, because, for each and every

thread a separate Session object will be created.

4. Create Transaction Object:

Transaction is a unit of work performed by Front End applications on Back end systems.

To represent Transactions, Hibernate has provided a predefined interface in the form of

"org.hibernate.Transaction".

To get Transaction object we will use either of the following methods.

1. public Transaction getTransaction()

It will return Transaction object with out begin, where to begin Transaction we have to use the

following method.

public void begin()

2. public Transaction beginTransaction()

It will return Transaction and begin Transaction.

In Hibernate applications, after performing persistence operations we must perform either commit

or rollback operations inorder to complete Transactions, for this, we have to use the following

methods from Transaction.

public void commit()

public void rollback()

Note: In Hibernate applications, Transaction is required for only Non Select operations, not

required for Select operations.

5. Perform Persistence Operations:

In Hibernate applications, to perform persistence operations Session has provided the following

methods.

To insert an object or record into Database table we have to use the following methods.

1. save(--)

2. persist(--)

6)Is it possible to interact with more than one Database from a single Hibernate Application?

Ans:

Yes, in single hibernate application we are able to interact with more than one Database, but, we

must use the following conventions.

1) We must provide separate configuration file for each and every Database.

2) Use either single mapping file if we have same table name and same columns in both the

databases tables and use different mapping file for each and every database if we have

difference in table names and column names.

3) In Client application.

a) Prepare Seperate Configuration object for each and every DB.

b) Prepare seperate SessionFactory object for each and every DB.

c) Prepare seperate Session object for each and every DB.

d) Prepare Seperate Transaction object for each and every DB as per the requirement.

e) Perform persistance operations on DB respective Session objects.

7)Is it possible to run hibernate applicatins with out using mapping file?

ANS:

Yes, it is possible to run hibernate applications with out using mapping file, but, we have to use

Annotations.