**Elements of Hibernate**

1. **Session factory** 
   1. Factory of instance of session.
   2. Holds second level cache
   3. Provides factory method to get session instance
2. **Session**
   1. provides an interface between the application and data stored in the database.
   2. It is a short-lived object and wraps the JDBC connection.
   3. It is factory of *Transaction, Query and Criteria*.
   4. It holds a first-level cache (mandatory) of data.
   5. provides methods to ***insert, update and delete the object***.
   6. provides factory methods for Transaction, Query and Criteria.
3. **Transaction**
   1. specifies the atomic unit of work. It is optional.
   2. provides methods for **transaction management**.
4. **Connection provider**
   1. It is a factory of JDBC connections.
   2. It abstracts the application from DriverManager or DataSource. It is optional.
5. **Transaction factory**
   1. It is a factory of Transaction. It is optional

create table emp(

  empno    number(4) primary key,

  ename    varchar2(10),

  job      varchar2(9),

  mgr      number(4,0),

  sal      number(7,2),

  comm     number(7,2),

  deptno   number(2));

insert into emp values(  7839, 'KING', 'PRESIDENT', null, 5000, null, 10);

insert into emp

values(

 7698, 'BLAKE', 'MANAGER', 7839,

 2850, null, 30

);

insert into emp

values(

 7782, 'CLARK', 'MANAGER', 7839,

 2450, null, 10

);

insert into emp

values(

 7566, 'JONES', 'MANAGER', 7839,

2975, null, 20

);

insert into emp

values(

 7788, 'SCOTT', 'ANALYST', 7566,

3000, null, 20

);

insert into emp

values(

 7902, 'FORD', 'ANALYST', 7566,

3000, null, 20

);

insert into emp

values(

 7369, 'SMITH', 'CLERK', 7902,

800, null, 20

);

insert into emp

values(

 7499, 'ALLEN', 'SALESMAN', 7698,

1600, 300, 30

);

insert into emp

values(

 7521, 'WARD', 'SALESMAN', 7698,

1250, 500, 30

);

insert into emp

values(

 7654, 'MARTIN', 'SALESMAN', 7698,

1250, 1400, 30

);

insert into emp

values(

 7844, 'TURNER', 'SALESMAN', 7698,

1500, 0, 30

);

insert into emp

values(

 7876, 'ADAMS', 'CLERK', 7788,

1100, null, 20

);

insert into emp

values(

 7900, 'JAMES', 'CLERK', 7698,

950, null, 30

);

insert into emp

values(

 7934, 'MILLER', 'CLERK', 7782,

1300, null, 10

);

create table empdetails(no number(4) primary key,firstname varchar2(10), lastname varchar2(10),mail varchar2(10));

**Hibernate**

1. Basic Terminologies (persistence, persistence data, persistence store, persistence operation,
2. Limitation of JDBC
3. What is **O-R mapping?**
4. Where to use JDBC and where to use ORM software?
5. Approaches to perform persistence operation?
6. What is programming language, software technology and framework?
   1. Types of framework, advantages of framework
7. Understanding various terminologies regarding classes
   1. Java beans
   2. POJO class
   3. Utility class
   4. BO, DTO, VO classes
   5. Component class
8. **Hibernate features**
9. Hibernate high level architecture diagram
10. Hibernate application and its internal flow

|  |  |
| --- | --- |
| **Save()** | **Persist()** |
| Return generated identity value and is Serializable | Return type is void |
| Uses same generators to generate id | Does not use same generator |
| It is only supported by Hibernate | It is also supported by JPA |
| **Get()** | **Load()** |
| Eager loading | Perform lazy loading |
| Does not generate proxy object | Generates proxy object |
| Involves 1 real object in loading | Involves 1 real and 1 proxy object in loading |
| Use when we have to check whether record is there or not | We should use this method only when we know record is there otherwise throws ObjectNotFoundException |
| Useful for **guaranteed utilization** of the object once it is loaded. | Useful for **delayed for non-guaranteed utilization** of record once it is loaded. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Deleting object**

1. By using session.delete(obj)
2. By loading the object and then use above method.

**Updating the record**

1. **Using session.update()**
2. **Load object then update**
3. **Setting object values in transaction (synchronization process)**

**Approaches**

1. **Using creating object, setting all values and using session.update(object) method** : here we have to create and set all values(values to be updates and not updated also.) We have to provide new values and old values whoch do not require any updation.

Limitations:

Here no provision for checking the record whether it exist or not.

We need to remember the old values and new values in order to update an object. If we are not setting values which don’t require any updation, by default null values will be store.

1. **Loading then updation**

Her we have to load object then call update method. Here no need to set unnecessary values.

1. **Doing updation in transaction (Synchronization process)**

|  |  |  |
| --- | --- | --- |
| **Update** | **saveOrUpdate** | **Merge** |
| Performs object/record update operation | Performs save/insert or update operation on record | Same |
| Does not check availability of object | Check availability of an object | Same |
| Generates update query | Generates select query and insert/update query | Same |
| Do not return any object representing updated record | Same | Returns the object that has been updated |
| Cannot perform record merging | Cannot perform record merging | **Performs record merging** |
|  |  |  |
|  |  |  |

**Cache**

**It is memory to store the data temporary to reduce network trip between client application and server.**

**In hibernate, cache resides at client application by holding domain class object.**

**Types of cache:**

L1 level cache/L1 cache

L2 level cache/L2 cache

L1 cache is associated with session object

L2 cache associated with session factory object and it is available across all sessions.

L1 cache is created with session object and destroyed when session is closed. We no need to do it explicitly.

**Flow:**

When session loads data, it searched the data in L1 cache, if it is not available, it will get it from DB and keep it into L1 and uses L1 cache for multiple requests.

**States of domain class object**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Transient** | not associated with session | * **Doesn’t contain id value** * **Doesn’t represent DB record** * **No associated with session** |
| **Persist** | associated with session | * **Contains id value** * **Associated with session** * **Represents DB records** * **Maintains synchronization** |
| **Detached** | previously associated but currently not associated. | * **Contains id values** * **Doesn’t represents record** * **Does not maintain synchronization** |

**Hibernate Mappings**

**Component Mapping**

If one class has to use another class special type property then we should go for composition.

If one class has to use another class all properties then we should go for inheritance.

If classes are in composition, we cannot take the DB tables in composition. And multiple classes having compositions mapping to single DB table is not possible i. e. more granules (classes) cannot map with single DB table, this is called as **problem of granularity**, that time we should go for **component mapping.**

|  |  |
| --- | --- |
| **Xml** | **Annotation** |
| **Use <component> tag** | **Use**  **@Embeddable**  **@Embedded** |
|  |  |

**Inheritance Mapping**

1. What is **subtype** **problem**?
2. Why we use inheritance mapping?
3. Types of Inheritance mapping?
   1. **Table per class**
   2. **Table per concrete class**
   3. **Table per subclass** (best recommended to use)
   4. **Table per class hierarchy**

* Here all domain classes will be mapped with single DB table. So object/record can be inserted by using any domain class of hierarchy
* One **discriminator value** is maintained which will tell using which domain class record is inserted
* We need to take one discriminator column to store discriminator values to indicate which domain class has inserted record.
* We need to configure discriminator value while configuring each domain class. For each class/ subclass, one separate discriminator value is required.

**Tags used**

**<class> tag -**

**<subclass> tag -**

**<discriminator-column> tag -**

**discriminator-value –**

**<id> tag –**

**[each class and subclass must have discriminator-value attribute].**

<**class** name=*"com.nt.domain.Person"* table=*"PERSON\_JOB\_DETAILS\_INH\_1"* **discriminator-value**=*"per"*>

<id name=*"id"* column=*"id"* />

<**discriminator column**=*"per\_type"* type=*"string"* />

<property name=*"name"* column=*"NAME"* type=*"string"* length=*"10"* />

<property name=*"company"* column=*"CMPNY"* type=*"string"* length=*"10"* />

<**subclass** name=*"com.nt.domain.Employee"* **discriminator-value**=*"emp"*>

<property name=*"dept"* column=*"DEPT"* type=*"string"* length=*"10"* />

<property name=*"salary"* column=*"SAL"* type=*"int"* length=*"4"* />

</subclass>

<**subclass** name=*"com.nt.domain.Customer"* **discriminator-value**=*"cus"*>

<property name=*"address"* column=*"addr"* type=*"string"* length=*"10"* />

</subclass>

</class>

**Limitation:**

1. **we cannot apply not-null constraints on subclass properties.**
2. **Single table having huge no. Of columns is a bad practice**
3. **To manipulate discriminator value, we need to use native sql.**
   1. **Table per subclass hierarchy**

* Here each domain class of the inheritance hierarchy will be mapped with a separate table. Table mapped with the super class is called as *parent table* and table mapped with subclasses is called as *child table. And both parent table and child table will have a relation.*
* Here each child table will have a **foreign key column** mapped with parent table primary column.

**Tags used**

**<class> tag -**

**<joined-subclass> tag -**

**<key-column> tag -**

**<id> tag –**

**<class name=*"com.nt.domain.Person"* table=*"PERSON\_TAB2"*>**

**<id name=*"id"* column=*"id"* />**

**<property name=*"name"* column=*"NAME"* />**

**<property name=*"company"* column=*"CMPNY"* />**

**<joined-subclass name=*"com.nt.domain.Employee"* table=*"EMPLOYEE\_TAB2"*>**

**<key column=*"emp\_id"*/>**

**<property name=*"dept"* column=*"DEPT"* />**

**<property name=*"salary"* column=*"SAL"* />**

**</joined-subclass>**

**<joined-subclass name=*"com.nt.domain.Customer"* table=*"CUSTOMER\_TAB2"*>**

**<key column=*"cus\_id"*/>**

**<property name=*"address"* column=*"ADDR"* />**

**</joined-subclass>**

**</class>**

* 1. **Table per concrete class hierarchy**
* Here each domain class of the inheritance hierarchy will be mapped with a separate table. Table mapped with the super class is called as *parent table* and table mapped with subclasses is called as *child table. And both parent table and child table will not have a relation.*
* *Child table also contain parent table column.*
* No **foreign key column,** no **discriminator column** will be there.
* There are 2 approaches to configure child table.
  + Use **<class>** tag to configure super class and subclass without inheriting parent table column.
  + Use **<class>** tag to configure super class and use **<union-subclass>** to configure child class.

**Tags used**

**<class> tag -**

**<union-subclass> tag -**

**<id> tag –**

**BULK PERSISTENT OPERATIONS IN HIBERNATE**

**To perform one or more record manipulation by using our choice valuesas criteria alue, use following technique:**

* + - 1. **HQL**
      2. **NATIVE HQL**
      3. **CRITERIA**

**HQL Queries**

1. **Introduction**
2. **Executing query with positional parameters**
3. **Executing query with named parameters**
4. **Retrieving specific multiple columns from DB table**
5. **Difference between list() and iterate()**
6. **Retrieving specific single column value from DB table**
7. **HQL Non Select queries**
8. **Introduction**

* Use to manipulate one or more records
* Object based query
* Database software independent
* For executing select query, use list() and iterate()
* For executing non-select query, use executeUpdate()
* Following thing are not possible
* Inserting record (direct values) in to DB, but we can insert values from one table to another table
* DDL operations : ***CREATE, DROP, ALTER, RENAME, TRUNCATE***
* PL/SQL programming

Ex:

**Without condition**

Sql> select \* from employee;

HQL> from Employee;

HQL> from Employee emp;

HQL> from Employee as emp;

HQL> select emp from Employee emp;

**Condition based**

Sql> select \* from employee where empId=12;

HQL> from Employee emp where emp.empId = 12;

HQL> from Employee where empId = 12;

*If you are writing HQL query to get all proporties of object I. e. all colom values of DB,* ***select*** *keyword is optional.*

**Query Object creation**

Query query=session.createQuery(“query”);

**Execution**

We can execute above method as

Query.list()/query.iterate()

We have two parameters passing ways:

1. **Positional parameters**
   1. *Drawback: if we are having number of positional parameters, then we may get confused with indexes. That’s why deprecated.* Alternate way- **named parameters or JPA possible parameters**
   2. Syntax: **=?**

Query.setXXX(int index, object value) ; => way to set parameters

1. **Named parameters**

Syntax: **:<name>**

Query.setXXX(String paramName, Object value) ; => way to set parameters

1. **JPA style parameter**

Syntax: **=?index (starting from 0)**

1. **Executing query with positional parameters ( 0 based index)**
2. **Executing query with named parameters**

* We cannot place positional parameter after using named parameter

From Employee where empName =? and empId = :EMPID(valid)

From Employee where empId = :EMPID and empName =? (invalid)

* We cannot place parameters in HQL representing HQL keywords, domain class name, domain class properties.

From Employee where ? = :empId;

1. **Retrieving specific multiple columns from DB table**

* When we use HQL query to retrieve **all column** values, we will get **List collection having domain class objects** as an element.
* When we use HQL query to retrieve **specific column** values, we will get **List collection having Object Array** as an element i. e. Each element of List will and java.lang.Object class array.

As Object[] is return type, it can be collected in List<> but not in Iterator<>.

1. **Difference between list() and iterate()**

|  |  |
| --- | --- |
| **List()** | **Iterate()** |
| Performs eager loading | Performs lazy loading |
| Doesn’t generate proxy object | Generates proxy object |
| Generates 1 select query to generate ‘n’ records | Generates 1 select query to generate ‘1+n’ records |
| Creates ‘**n**’ objects for domain class to retrieve ‘n’ records from DB | Creates ‘**2\*n**’ objects for domain class to retrieve ‘n’ records from DB  For example: dynamically created proxy class objects |
| Returns **List** collection having columns values in form of domain class object | Returns **Iterator** object that points to list collection having **proxy**  class object |
| Use when there is guarantee that fetched data is going to process or not | Use when there is no guarantee that fetched data is going to process or not |

1. **Retrieving specific single column value from DB table**

* When HQL query retrieves specific single column value then that column property type element’s List collection is returned.

1. **HQL Non Select queries**

* We need to call executeUpdate() to execute non select operations which will return a nueric value representing no. Of records affected.
* It must be executed in ***transactional*** *environment*

**Named HQL Query**

Drawback of HQL:

* + - 1. One query is associated with one session object, we can’t use one query in multiple sessions.
      2. We cannot perform some operations using HQL.

Syntax:

On top Entity class write below annotation

@NamedQuery(name=”query\_name”,query=””)

@NamedQueries(

@NamedQuery(name=”query\_name1”,query=””),

@NamedQuery(name=”query\_name2”,query=””),

)

Creating query object

Query query=session.getamedQuery(query\_name1);

Setting parameters

Query.setXxx();

**Criteria (Query by criteria)**

* It is persistent logic written in java based statements without using queries. It generates optimized SQL query as required for DATABASE software. It is developed by domain class and its properties so it is database software independent logic.
* It can perform only **single row or bulk row select** **operations** like select all columns, select specific columns, aggregation functions**.**
* Non select operations, PL/SQL procedures and functions, DDL operations are not supported by it.

Objects used in QBC are:

**Criteria** – used to develop QBC logic

**Criterion** – used to put condition in query

**Criteria** object represents whole QBC logic & it is obtained by session object by passing .class property of domain class as an argument.

Only **list()** method is used to execute its logic and iterate() is not used that mean only eager loading is possible.

**Criterion** object is used to create a condition & we need to add that condition to criteria object.