Hibernate
   Don't write SQL Query(DB specific)

Hibernate
      a. SRO(Single Row Operation)
      b. Bulk Operation
                a. HQL(Hibernate Query Language)=> Write query using Entityname and properties.
                b. By writing NativeSQL query   => Write query using Tablename and column names.
                c. Using Criterian API(Pure java code).

NativeSQL

=> This is another bulk operation technique to perform both select and non select operation.
=> With this native sql, we can execute sql commands on database for bulk operations.
=> The two reasons to give NativeSQL in hibernate is
      a. For programmer convinence
      b. To migrate jdbc application as hibernate application easily.
=> The only problem with nativesql is it makes hibernate application "Database dependent".
=> To run sql command we need to first create SQLQuery object, we can create this by using createSQLQuery().
=> For select operation we call list()/getResultList() and for non-select operation we call executeUpdate().
=> These queries performance is good as they go to database directly for execution.
=> These queries will be written specifically by using db tablename and db column names.

select query without using Entity object

```
NativeQuery<Object[]> query = session
                                    .createSQLQuery("select * from employee where eid>=:id1 and eid<=:id2");
query.setParameter("id1", 1);
query.setParameter("id2", 7);
List<Object[]> emps = query.list();
emps.forEach(row -> {
                        for (Object obj : row) {
                                    System.out.print(obj + "\t");
                              }
                        System.out.println();
               });
```

select query using Entity object

```
NativeQuery<Employee> query = session
                                    .createSQLQuery("select * from employee where eid>=:id1 and eid<=:id2");
query.setParameter("id1", 1);
query.setParameter("id2", 7);
query.addEntity(Employee.class);
List<Employee> emps = query.list();
emps.forEach(System.out::println);
```

Select particular column from a table

```
-------------------------------------
NativeQuery<Object[]> query = session
                                    .createSQLQuery("select ename,esalary from
employee where eid>=:id1 and eid<=:id2");
query.setParameter("id1", 1);
query.setParameter("id2", 7);
query.addScalar("ename", StandardBasicTypes.STRING);
query.addScalar("esalary", StandardBasicTypes.INTEGER);
List<Object[]> emps = query.list();
      emps.forEach(row -> {
                        for (Object obj : row) {
                                    System.out.print(obj + "\t");
                              }
                        System.out.println();
                  });

Note: It is a good practise to bind EntityQuery result with Entity class and
Scalary query results with Hibernate data types.
            Query<T>(I)
               |
               | extends
               |
         NativeQuery<T>(I)



Performing insert query(need transaction object)
-----------------------------------------------
NativeQuery query = session.createSQLQuery(
                                    "insert into
employee(`ename`,`eage`,`eaddress`,`esalary`)values(:name,:age,:addr,:sal)");

query.setParameter("name", "nitin");
query.setParameter("age", 30);
query.setParameter("addr", "RCB");
query.setParameter("sal", 1500);
rowCount = query.executeUpdate();


NamedNativeQuery
---------------
@Entity
@NamedNativeQuery(name = "SQL_INSERT_QUERY",
                  query = "insert into
employee(`ename`,`eage`,`eaddress`,`esalary`)values(:name,:age,:addr,:sal)")
public class Employee implements Serializable {
                  ;;;;;
                  ;;;;;
                  ;;;;;
}

NativeQuery query = session.getNamedNativeQuery("SQL_INSERT_QUERY");
query.setParameter("name", "hyder");
query.setParameter("age", 28);
query.setParameter("addr", "RCB");
query.setParameter("sal", 2000);
rowCount = query.executeUpdate();

Working with StoredProcedure
```

```
----------------------------
=> To make persistence logic/buisness logic reusable across multiple modules of the
project by keeping logic in db software, then we need to
   go for StoredProcedure.
=> The syntax of stored procedure varies from database to database
            a. Oracle(PL/SQL)
            b. MySQL
      When we work with StoredProcedure we use 3 types of params like
            a. IN(default mode)
            b. OUT
            c. INOUT

Getting the complete resultList
==============================
CREATE DEFINER=`root`@`localhost` PROCEDURE `P_GET_PRODUCT_BY_NAME`(IN name1
VARCHAR(20), IN name2 VARCHAR(20))
BEGIN
            SELECT pid,pname,price,qty FROM products WHERE pname IN (name1,name2);
      END$$

DELIMITER ;

To call a procedure we use
      CALL `P_GET_PRODUCT_BY_NAME`("fossil","tissot")

code
----
ProcedureCall procedureCall =
session.createStoredProcedureCall("P_GET_PRODUCT_BY_NAME", Product.class);
      String name1 = "fossil";
      String name2 = "tissot";
      procedureCall.registerParameter(1, String.class,
ParameterMode.IN).bindValue(name1);
      procedureCall.registerParameter(2, String.class,
ParameterMode.IN).bindValue(name2);

      List<Product> products = procedureCall.getResultList();
      products.forEach(System.out::println);


Getting the record on the basis of id
====================================
CREATE DEFINER=`root`@`localhost` PROCEDURE `P_GET_PRODUCT_DETAILS_BY_ID`(IN id
INT,OUT NAME VARCHAR(20),
                  OUT rate INT, OUT qnt INT)
BEGIN
            SELECT pname,price,qty INTO NAME,rate,qnt FROM products WHERE pid = id;
      END$$

DELIMITER ;

To call a procedure we use
      CALL `P_GET_PRODUCT_DETAILS_BY_ID`(1,@name,@rate,@qnt)
      SELECT @name,@rate,@qnt


Code
----
ProcedureCall procedureCall =
```

```
session.createStoredProcedureCall("P_GET_PRODUCT_DETAILS_BY_ID");
Integer id = 1;

procedureCall.registerParameter(1, Integer.class, ParameterMode.IN).bindValue(id);
procedureCall.registerParameter(2, String.class, ParameterMode.OUT);
procedureCall.registerParameter(3, Integer.class, ParameterMode.OUT);
procedureCall.registerParameter(4, Integer.class, ParameterMode.OUT);

String pname = (String) procedureCall.getOutputParameterValue(2);
Integer price = (Integer) procedureCall.getOutputParameterValue(3);
Integer qty = (Integer) procedureCall.getOutputParameterValue(4);

System.out.println("PID\tPNAME\tPRICE\tQTY");
System.out.println(id+"\t"+pname+"\t"+price+"\t"+qty);
```

Criterian api

It is another technique to perform bulk operations.
Using Criterian api we can perform only select operations.
Even by using HQL we can perform select operation by writing select query.
If we directly write HQL select query, then we need to tune the query for better performance.
we can read the same data from database by constructing query in multiple ways, but performance is important.
Performance: reading the data with small amount of time.
In Hibernate applications, to avoid totally query langugaes like SQL and HQL,....and to provide the complete Persistence logic in the form of JAVA code we must use "Criterion API".
Criterian api does not supports StoredProcedure.
Criterian api does not supports NamedQuery,NamedNativeQuery.

There are 2 modes of Writing Criterian API
- a. HB QBC/Criteria API(specific to hibernate only and we can use to work only for select operation)
- b. JPA QBC/Criteria API(common to all ORM frameworks,we can perform both select and non-select operation)

1. Create Criteria Object:
Criteria object is a central object in Critera API, it able to manage HQl query repersentation internally and it has provided predefined methods to defined query logic.
To create Criteria object we have to use the following method from Session.

```
public Criteria createCriteria(Class cls);
```

EX: `Criteria c = session.createCriteria(Employee.class);`
Note: It is equalent to the HQL query internally "from Employee".

2. Prepare Criterion objects and add that Criterion objects to Criteria object:
Criterion is an object , it able to manage a single Conditional expression in database logic.
To create Criterion object we have to use the following methods from "org.hibernate.Restrictions" class.

```
public static Criterion isEmpty(String property)
public static Criterion isNotEmpty(String property)
public static Criterion isNull(String property)
public static Criterion isNotNull(String property)
```

```
      public static Criterion in(String property, Object[] obj)
      public static Criterion in(String property, Collection c)
      public static Criterion between(String property, Object min_Val, Object
max_Val)
      public static Criterion between(String property, Object[] obj)
      public static Criterion eq(String property, Object val)
      public static Criterion ne(String property, Object val)
      public static Criterion lt(String property, Object val)
      public static Criterion le(String property, Object val)
      public static Criterion gt(String property, Object val )
      public static Criterion ge(String property, Object val)
      -----
      -----
```

To add a particular Criterion object to Criteria object we have to use the following method.

```
                  public void add(Criterion c)
```

EX:

```
Criterion c1 = Restrictions.ge("esal", 60000);
Criterion c2 = Restrictions.le("esal", 90000);
            c.add(c1);
            c.add(c2);
```

Note: With the above steps, Critera object is able to prepare the query like "from Employee where esal>=6000 and esal<=9000".

3. Create Projection objects , add Projection objects to ProjectionList and add ProjectionList to Criteria object:

The main intention of Projection object is to represent a single POJO class property.

To get Projection object with a particular Property name we have to use the following method from "Projections" class.

```
                  public static Projection projection(String pro_Name)
```

To create ProjectionList object we have to use the following method from Projections class.

```
                  public static ProjectionList projectionList()
```

To add Projection object to ProjectionList we have to use the following method from ProjectionList class.

```
                  public void add(Projection p)
```

To set ProjectionList to Criteria object we have to use the following method.

```
                  public void setProjection(ProjectionList pl)
```

Ex:

```
ProjectionList pl = Projections.projectionList();
pl.add(Projections.property("eno"));
pl.add(Projections.property("ename"));
pl.add(Projections.property("esal"));
pl.add(Projections.property("eaddr"));
c.setProjection(pl);
```

EX: With the above , Criteria object is able to prepare HQL query internally like below.

```
                        "select eno, ename, esal, eaddr from Employee where esal
>=6000 and esal<=90000".
```

4) Provide a particular Order to the query:

To represent a particular Order over the results, we have to use either asc(-) or desc(-) methods from "Order" class.

```
            public static Order asc(String prop_Name)
            public static Order desc(String prop_Name)
To add Order object to Criteria object we have to use the following method.
            public void addOrder(Order o)
EX:
      Order o = Order.desc("ename");
      c.addOrder(o);

Note: With this, Criteria object is able to create HQl query like
       "select eno, ename, esal, eaddr from Employee where esal >=6000 and
esal<=90000 order by desc(ename)"


Scalar Projection
-----------------
Criteria criteria = session.createCriteria(Employee.class);

Criterion c1 = Restrictions.le("salary", 65000);
Criterion c2 = Restrictions.gt("salary", 6000);
criteria.add(c1);
criteria.add(c2);

ProjectionList projectionList = Projections.projectionList();
      projectionList.add(Projections.property("eid"));
      projectionList.add(Projections.property("ename"));
      projectionList.add(Projections.property("eage"));

      criteria.setProjection(projectionList);

      Order order = Order.desc("ename");
      criteria.addOrder(order);

      List<Object[]>emps = criteria.list();
      emps.forEach(row->{
                        for (Object obj : row) {
                              System.out.print(obj+"\t");
                        }
                        System.out.println();
                  });



Adding multiple condition with or,AND
-------------------------------------
Criteria criteria = session.createCriteria(Employee.class);

Criterion cond1 = Restrictions.between("salary",6000,65000);
Criterion cond2 = Restrictions.in("ename", "sachin","kohli");
Criterion cond3 = Restrictions.ilike("eaddress","R%");
Criterion finalCond = Restrictions.or(Restrictions.and(cond1,cond2),cond3);
      criteria.add(finalCond);

List<Employee> list = criteria.list();
list.forEach(System.out::println);


Adding only one column
----------------------
Criteria criteria = session.createCriteria(Employee.class);
```

```
Criterion c1 = Restrictions.eq("eaddress", "RCB");
criteria.add(c1);

PropertyProjection property = Projections.property("ename");
criteria.setProjection(property);

List<String> emps = criteria.list();
emps.forEach(System.out::println);


HQL/JPQL
--------
1. Persistence logic is DB independent
2. Query will be written using Entity classname and property name.
3. Supports only named params.
4. Supports both select and non-select queries.
5. It is not suitable for calling storedprocedure.

NativeSQL
--------
1. It is dependent
2. Query will be written using DBTablenames and column names.
3. Supports only named params.
4. Supports both select and non-select queries.
5. It is  suitable for calling storedprocedure.


CriteriaAPI
----------
1. It is database independent
2. Query will be written using Entity classname and property name.
3. It won't support params
4. Only select operation using HB-QBC
5. It is not suitable for calling storedprocedures.


Remaining topics of hibernate
-----------------------------
Filters,SoftDeletion
Pagination
ORMapping(uni-directional,BiDirectional using joins)
      a. OnetoOne
      b. OnetoMany
      c. ManytoOne
      d. ManytoMany
```