Note:- to see the ORM tool(HB) generated sql queries on the console add the following property inside the persistence.xml

```
<property name="hibernate.show_sql"  value="true"/>
```

to create or update the table according to the entity class mapping information:-

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

create :- drop the existing table then create a fresh new table and insert the record.

update :- if table is not there then create a new table, and if table is already there it will perform insert operation only in the existing table.

some of the annotations of JPA:-

@Entity :- to make a java bean class as entity , i.e to map with a table

@Id :- to make a field as the ID field (to map with PK of a table)

@Table(name="mystudents") :- if the table name and the class name is different

@Column(name="sname") :- if the column name of table and corresponding variable of the class is diff.

@Transient : it will ignore the filed value.

Generators in JPA:-

--Generators are used to generate the ID filed value automatically.

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int roll;
```

--here roll will be generated automatically for each row.

**Note:- if we use this @GeneratedValue annoation then we are not allowed to give the roll explicitly while inserting a record.

--so we should create a object by using zero argument constrcutor and set the each value by calling setter method.or we can use overloaded constructor which ignore the Id field.

AUTO :- internally underlaying ORM s/w creates a table called "hibernate_sequence" to maintain the Id value.

IDENTITY :- it is used for auto_increatement feature to auto generate the id value

SEQUENCE :- it is used for sequence feature to auto generate the id value

DAO pattern example with JPA:-
==========

EMUtil.java:-

```
public class EMUtil {


	private static EntityManagerFactory emf;

	static{
		emf=Persistence.createEntityManagerFactory("account-unit");
	}
```

```
	public static EntityManager provideEntityManager(){

		//EntityManager em= emf.createEntityManager();
		//return em;

		return emf.createEntityManager();
	}
}
```

Account.java:- (Entity class)

```
@Entity
public class Account  {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int accno;
	private String name;
	private int balance;


	public Account() {
		// TODO Auto-generated constructor stub
	}


	public int getAccno() {
		return accno;
	}


	public void setAccno(int accno) {
		this.accno = accno;
	}


	public String getName() {
		return name;
	}
```

```
	public void setName(String name) {
		this.name = name;
	}


	public int getBalance() {
		return balance;
	}


	public void setBalance(int balance) {
		this.balance = balance;
	}


	public Account(int accno, String name, int balance) {
		super();
		this.accno = accno;
		this.name = name;
		this.balance = balance;
	}


	@Override
	public String toString() {
		return "Account [accno=" + accno + ", name=" + name + ", balance="
				+ balance + "]";
	}

}
```

AccountDao.java:-(interface)

```
public interface AccountDao {

	public boolean createAccount(Account account);

	public boolean deleteAccount(int accno);

	public boolean updateAccount(Account account);
```

```
	public Account findAccount(int accno);

}
```

AccountDaoImpl.java:-

```
public class AccountDaoImpl implements AccountDao{

	@Override
	public boolean createAccount(Account account) {

		boolean flag= false;

		EntityManager em= EMUtil.provideEntityManager();

		em.getTransaction().begin();

		em.persist(account);
		flag=true;

		em.getTransaction().commit();


		em.close();

		return flag;
	}

	@Override
	public boolean deleteAccount(int accno) {
		boolean flag=false;


		EntityManager em= EMUtil.provideEntityManager();

		Account acc=em.find(Account.class, accno);

		if(acc != null){

			em.getTransaction().begin();
```

```
                em.remove(acc);
                flag=true;

                em.getTransaction().commit();
        }

        em.close();



        return flag;
}

@Override
public boolean updateAccount(Account account) {

        boolean flag=false;


        EntityManager em= EMUtil.provideEntityManager();

        em.getTransaction().begin();

        em.merge(account);
        flag=true;

        em.getTransaction().commit();


        em.close();

        return flag;

}

@Override
public Account findAccount(int accno) {
        /*Account account=null;

 EntityManager em=EMUtil.provideEntityManager();


        account = em.find(Account.class, accno);
```

```
                return account;*/

                return EMUtil.provideEntityManager().find(Account.class, accno);

        }
}
```

persistence.xml:-

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
          version="2.0">


   <persistence-unit name="account-unit" >


<properties>

          <property name="hibernate.connection.driver_class"
value="com.mysql.cj.jdbc.Driver"/>
          <property name="hibernate.connection.username" value="root"/>
          <property name="hibernate.connection.password"  value="root"/>
          <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/ratandb"/>

          <property name="hibernate.show_sql"  value="true"/>
          <property name="hibernate.hbm2ddl.auto"  value="update"/>



      </properties>
```

```
   </persistence-unit>
</persistence>
```

DepositUseCase.java:-

```
public class DepositUseCase {

	public static void main(String[] args) {

		AccountDao dao=new AccountDaoImpl();

		/*Account acc1=new Account();
		acc1.setName("Ramesh");
		acc1.setBalance(880);



		boolean f= dao.createAccount(acc1);

		if(f)
			System.out.println("Account created..");
		else
			System.out.println("Not created...");*/


		Scanner sc=new Scanner(System.in);

		System.out.println("Enter Account number");
		int ano=sc.nextInt();

		Account acc= dao.findAccount(ano);

		if(acc == null)
			System.out.println("Account does not exist..");
		else{


			System.out.println("Enter the Amount to Deposit");
			int amt=sc.nextInt();
```

```
                    acc.setBalance(acc.getBalance()+amt);

                    boolean f =dao.updateAccount(acc);

                    if(f)
                        System.out.println("Deposited Sucessfully...");
                    else
                        System.out.println("Technical Error .....");

                }
        }

}
```

WithdrawUseCase.java:-

```
public class WithdrawUseCase {

        public static void main(String[] args) {

                AccountDao dao=new AccountDaoImpl();

                Scanner sc=new Scanner(System.in);

                System.out.println("Enter Account number");
                int ano=sc.nextInt();

                Account acc= dao.findAccount(ano);

                if(acc == null)
                        System.out.println("Account does not exist..");
                else{

                        System.out.println("Enter the withdrawing amount");
                        int amt=sc.nextInt();

                        if(amt <= acc.getBalance()){

                                acc.setBalance(acc.getBalance()-amt);
                                boolean f=dao.updateAccount(acc);
                                if(f)
                                        System.out.println("please collect the cash...");
```

```
                                else
                                        System.out.println("Technical Error...");


                        }else
                                System.out.println("Insufficient Amount..");
                }
        }

}
```

limitation of EM methods in performing CRUD operations:-

---

persist();
find()
merge();
remove();

1.Retrieving Entity obj based on only ID field(PK field) @Id

2.multiple Entity obj retrival is not possible (multiple record)

3.bulk update and bulk delete is also not possible

4.to access Entity obj we can not specify some extra condition.

--to overcome the above limitation JPA has provided JPQL (java persistence query language).

JPQL:

=====

similarities bt JPQL and sql:-

--keywords in the both the langauges are case insensetive

--GROUP BY,ORDER BY,where clause r similar

--aggregrate function r similar

--the way we express the condition to perform the CRUD operation is almost simmilar.

diff bt JPQL and sql:-

--sql queries are expressed in the term of table and columns, where as jpql query is expressed in the term of Entity class names and its variables.

--the name of the class and its variables are case sensitive.

--sql is not portable across multiple dbms, where jpql is portable.

sql> select name,marks from student; (name and marks are the column name and student is the table name)

jpql> select name,marks from Student; (here name and marks are the variables defined inside the Student class)

Note: we should not use * in jpql:
ex:

sql>select * from student;

jpql>from Student; //projecting all the column
or
jpql>select s from Student s;

jpql> select s.name,s.marks from Student s;

steps to use the jpql in JPA application:-
---

step 1:- develop the JPQL query as string.

step 2:- create javax.persistnce.Query(I) object by calling "createQuery(-)" method on the EM object.

ex:-

Query q =em.createQuery("JPQL query");

Query object the Object Oriented representation of JPQL.

step 3:- bind the values if any placeholders are used.(here we have 2 types of place holders 1.positional 2.named placeholders).

step 4:- submit the jpql query by calling either one of the following methods:-

for select statments:-

List getResultList(); (if more than one record.)
Object getSingleResult(); (if atmost one record)

for insert/update/delete :-

int executeUpdate(); //this method should be called inside the tx area.

ex:-

in sql :-

select * from account;

in jpql:-

select a from Account a;

from Account; //it is a shortcut

ex:- getting all the records from the DB:-

JPQLMain.java:-

```
public class JPQLMain {

	public static void main(String[] args) {

		EntityManager em= EMUtil.provideEntityManager();


		//String jpql= "select a from Account a";
		String jpql= "from Account";
		Query q= em.createQuery(jpql);

		List<Account> list= q.getResultList();

		for(Account acc:list){
			System.out.println(acc);
		}
	}
}
```

search on non-pk:-

```
	EntityManager em= EMUtil.provideEntityManager();


		//String jpql= "select a from Account a where a.name='Ram' ";
		String jpql= "from Account where name='Ram'";
		Query q= em.createQuery(jpql);
```

```
		List<Account> list= q.getResultList();

		for(Account acc:list){
			System.out.println(acc);
		}
	}
```

if we conform that only one row will come then :-

```
	EntityManager em= EMUtil.provideEntityManager();


		//String jpql= "select a from Account a where a.name='Ram' ";
		String jpql= "from Account where name='Ram'";
		Query q= em.createQuery(jpql);

		//Object obj= q.getSingleResult();
		//Account acc= (Account)obj;

		Account acc= (Account)q.getSingleResult();

		System.out.println(acc);
```

--if the above query will return more that one result then it will throw a runtime exception