Generators in Hibernate

Hinernate uses different primary key generators algorithms . For each algorithm internally a class is created by hibernate for its implementation .

And all these classes are implemented from '**org.hibernate.id.IdentifierGenerator** (Interface) '

We can use generators in two ways

1. Using <id> <generator class="……" / > </id>
2. Using annotations in the entity class .

Different types of generators used in Hibernate

auto -- @GeneratedValue(strategy = GenerationType.IDENTITY)

these automatically writes values into id(PK) column starting with 1 & incrementing by 1 .

'hibernate_sequence' named table too which contains column as 'next_val' contains next possible primary key

identity -- it uses auto incremented column algorithm of database & returns these value .

@GeneratedValue(strategy = GenerationType.IDENTITY)

these automatically writes values into id(PK) column starting with 1 & incrementing by 1 .

NOTE :: These generator works on MySQL but not with ORACLE

sequence -- it creates the id as sequence (incrementing by one) .

if you just use @GeneratedValue(strategy = GenerationType.SEQUENCE)

these automatically writes values into id(PK) column starting with 1 & incrementing by 1 .

'hibernate_sequence' named table too which contains column as 'next_val' contains next possible primary key

if you use @SequenceGenerator(name="mySqGen", sequenceName="mySq", initialValue=5, allocationSize=100)

@GeneratedValue (generator ="mySqGen")

These generator starts assigning id from 5 increments by 1 . it is a number after which the database query will be made again to get the next database sequence value.

Another table also created automatically name 'mySq' with column 'next_val' and value= 2*allocationSize + initialValue

table -- uses an underlying database table that holds segments of identifier generation values

we can use like @GeneratedValue(strategy = GenerationType.TABLE)

these automatically writes values into id(PK) column starting with 1 & incrementing by 1 .

'hibernate_sequence' named table too which contains column as 'next_val' contains max primary key value as 'default'

we can also use it like @GeneratedValue(strategy = GenerationType.TABLE, generator = "book_generator")

@TableGenerator(name="book_generator", table="id_generator", schema="bookstore")

we can implement our own generators classes in hibernate

==========

HQL Queries

Lets say enitity class name is --- EmployeeDetails

And table name is --- employee_details

You can use below given HQL queries in session.createQuery(-);

Select * from employee_details; ==
List<EmployeeDetails> listOfEmployees = session.createQuery(" from EmployeeDetails ").list();

Select name , department , email from employee_details; ==

```
Query query = session.createQuery("select  e.name , e.department , e.email from EmployeeDetails e");
List<Object[]> rows = query.getResultList();
```

Select * from employee_details where id=10; ==

```
EmployeeDetails e = (EmployeeDetails)session.createQuery("from EmployeeDetails e where e.id=10").getSingleResult();
```

Select * from employee_details where name='G'; == EmployeeDetails e =
(EmployeeDetails)session.createQuery("from EmployeeDetails e where e.name='G'").getSingleResult();

Select * from employee_details where id > 5; ==

```
listOfEmp = session.createQuery("from EmployeeDetails e where e.id>5");
```

update employee_details set password='somethingNew' where id = 10 ; ==

```
Query query = session.createQuery(" update EmployeeDetails set password =: pwd  where  id =: id ");
query.setParameter("pwd" , "somethingNew");
query.setParameter("id" , 10);
int  result =  query.executeUpdate();
```

==========