Hibernate-Primary Keys

Hibernate Primary keys

As we know @Id creates a primary keys in our previous post.
First lets understand what is a natural and a Surrogate keys

Natural Keys:
Suppose in your application you have a column which is mandatory and whose value will be **distinct for business reason** so you have defined it as a primary key.Such as in a Registration form you will have a mandatory email id which has to be distinct as well for business reason,so this column can be defined as primary key so this is called as a Natural Key.

Surrogate keys:
Suppose you don't have a column that can be marked as unique or you don't know if that column will be unique in that case you add another colour to it with a purpose to make it a primary key ,it **does not have a business significance** can be marked as Surrogate key,for example a column Serial Number which has only reason to represent each row as unique is a Surrogate Key
Lets see how Hibernate supports each of these.
Now suppose for example you have a login_id as primary key which is also used for authentication and so also has other business significance then we need to control it,but suppose we have a key which does not have a business significance then we can ask Hibernate to do the job for us.
So whenever a surrogate key is required as primary key we can ask the hibernate to generate it for us as it does not have any business significance and it only has to be unique.
so for this we have to use @GeneratedValue annotation like below.

```
@Id @GeneratedValue
private int userId;
```

Now there is no need to provide the userId while we are inserting record in DB,
Hibernate will check if the primary key value is present,if its not it will create one and then for the next entry will do a +1 to add the next entry.

```
package com.technicalstack.dto;

import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateTest {

 public static void main(String[] args) {
  UserDetails user1 = new UserDetails();
 // user1.setUserId(1);  as we are using @Id @GeneratedValue so we don't need to set it.
  user1.setName("Shailesh");

  UserDetails user2 = new UserDetails();
  user2.setName("Rajesh");
```

```
  SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
  Session session = sessionFactory.openSession();
  session.beginTransaction();

  session.save(user1);
  session.save(user2);

  session.getTransaction().commit();
  session.close();

  }

}
```

Output:

```
Hibernate: create table USER_DETAILS (userId int4 not null, USER_NAME varchar(255), primary key (userId))
 Sep 24, 2016 2:50:06 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
 INFO: HHH000230: Schema export complete
 Hibernate: select nextval ('hibernate_sequence')
 Hibernate: select nextval ('hibernate_sequence')
 Hibernate: insert into USER_DETAILS (USER_NAME, userId) values (?, ?)
 Hibernate: insert into USER_DETAILS (USER_NAME, userId) values (?, ?)
```

Notice : Hibernate has done a nextval to get the next primary key.
We can add strategy with @GeneratedValue as
@Id @GeneratedValue(strategy=.....)

Their are 4 Strategies that can be configured

- The Auto Strategy
- The Identity Strategy
- The Sequence Strategy
- The Table Strategy

The Auto Strategy

ObjectDB maintains a special global number generator for every database. This number generator is used to generate automatic object IDs for entity objects with no primary key fields defined (as explained in the previous section).
The same number generator is also used to generate numeric values for primary key fields annotated by @GeneratedValue with the AUTO strategy:

```
@Id @GeneratedValue(strategy=GenerationType.AUTO) int id;
}
```

AUTO is the default strategy, so the following definition is equivalent:

```
@Id @GeneratedValue int id;
}
```

During a commit the AUTO strategy uses the global number generator to generate a primary key for every new entity object. These generated values are unique at the database level and are never recycled, as explained in the previous section.

The Identity Strategy

The IDENTITY strategy is very similar to the AUTO strategy:
@Id @GeneratedValue(strategy=GenerationType.IDENTITY) int id;
}
The IDENTITY strategy also generates an automatic value during commit for every new entity object. The difference is that a separate identity generator is managed per type hierarchy, so generated values are unique only per type hierarchy.

The Sequence Strategy
The sequence strategy consists of two parts - defining a named sequence and using the named sequence in one or more fields in one or more classes. The @SequenceGenerator annotation is used to define a sequence and accepts a name, an initial value (the default is 1) and an allocation size (the default is 50). A sequence is global to the application and can be used by one or more fields in one or more classes. The SEQUENCE strategy is used in the @GeneratedValue annotation to attach the given field to the previously defined named sequence:
// Define a sequence - might also be in another class:
@SequenceGenerator(name="seq", initialValue=1, allocationSize=100)
public class UserDetails {
// Use the sequence that is defined above:
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq")
@Id int id;
}
Unlike AUTO and IDENTITY, the SEQUENCE strategy generates an automatic value as soon as a new entity object is persisted (i.e. before commit). This may be useful when the primary key value is needed earlier. To minimize round trips to the database server, IDs are allocated in groups. The number of IDs in each allocation is specified by the allocationSize attribute. It is possible that some of the IDs in a given allocation will not be used. Therefore, this strategy does not guarantee there will be no gaps in sequence values.

The Table Strategy
The TABLE strategy is very similar to the SEQUENCE strategy:
@Entity
@TableGenerator(name="tab", initialValue=0, allocationSize=50)
public class UserDetails {
@GeneratedValue(strategy=GenerationType.TABLE, generator="tab")
@Id int id;
ORM-based JPA providers (such as Hibernate) simulate a sequence using a table to support this strategy. ObjectDB does not have tables, so the TABLE and SEQUENCE strategies are almost identical.
}