An Overview of Identifiers in Hibernate/JPA

1. Overview

Identifiers in Hibernate represent the primary key of an entity. This implies the values are unique so that they can identify a specific entity, that they aren't null and that they won't be modified.

Hibernate provides a few different ways to define identifiers. In this article, we'll review each method of mapping entity ids using the library.

2. Simple Identifiers

The most straightforward way to define an identifier is by using the *@Id* annotation.

Simple ids are mapped using *@Id* to a single property of one of these types: Java primitive and primitive wrapper types, *String*, *Date*, *BigDecimal* and *BigInteger*.

Let's see a quick example of defining an entity with a primary key of type *long*:

```
@Entity
public class Student {

    @Id
    private long studentId;

    // standard constructor, getters, setters
}
```

3. Generated Identifiers

If we want to automatically generate the primary key value, **we can add the *@GeneratedValue* annotation.**

This can use four generation types: AUTO, IDENTITY, SEQUENCE and TABLE.

If we don't explicitly specify a value, the generation type defaults to AUTO.

3.1. *AUTO* Generation

If we're using the default generation type, the persistence provider will determine values based on the type of the primary key attribute. This type can be numerical or *UUID*.

For numeric values, the generation is based on a sequence or table generator, while *UUID* values will use the *UUIDGenerator*.

Let's first map an entity primary key using AUTO generation strategy:

```
@Entity
public class Student {

    @Id
    @GeneratedValue
    private long studentId;

    // ...
}
```

In this case, the primary key values will be unique at the database level.

Now we'll look at the *UUIDGenerator*, which was introduced in Hibernate 5.

In order to use this feature, we just need to declare an id of type *UUID* with *@GeneratedValue* annotation:

```
@Entity
public class Course {
```

```
    @Id
    @GeneratedValue
    private UUID courseId;

    // ...
}
```

Hibernate will generate an id of the form "8dd5f315-9788-4d00-87bb-10eed9eff566".

3.2. *IDENTITY* Generation

This type of generation relies on the *IdentityGenerator*, which expects values generated by an *identity* column in the database. This means they are auto-incremented.

To use this generation type, we only need to set the *strategy* parameter:

```
@Entity
public class Student {

    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private long studentId;

    // ...
}
```

One thing to note is that IDENTITY generation disables batch updates.

3.3. *SEQUENCE* Generation

To use a sequence-based id, Hibernate provides the *SequenceStyleGenerator* class.

This generator uses sequences if our database supports them. It switches to table generation if they aren't supported.

In order to customize the sequence name, we can use the *@GenericGenerator* annotation with *SequenceStyleGenerator strategy*.

```
@Entity
public class User {
    @Id
    @GeneratedValue(generator = "sequence-generator")
    @GenericGenerator(
      name = "sequence-generator",
      strategy = "org.hibernate.id.enhanced.SequenceStyleGenerator",
      parameters = {
        @Parameter(name = "sequence_name", value = "user_sequence"),
        @Parameter(name = "initial_value", value = "4"),
        @Parameter(name = "increment_size", value = "1")
        }
    )
    private long userId;

    // ...
}
```

In this example, we've also set an initial value for the sequence, which means the primary key generation will start at 4.

SEQUENCE is the generation type recommended by the Hibernate documentation.

The generated values are unique per sequence. If we don't specify a sequence name, Hibernate will reuse the same *hibernate_sequence* for different types.

3.4. TABLE Generation

The *TableGenerator* uses an underlying database table that holds segments of identifier generation values.

Let's customize the table name using the *@TableGenerator* annotation:

```
@Entity
public class Department {
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.TABLE,
      generator = "table-generator")
    @TableGenerator(name = "table-generator",
      table = "dep_ids",
      pkColumnName = "seq_id",
      valueColumnName = "seq_value")
    private long depId;

    // ...
}
```