4 Primary Key Generation Strategies

By Ramesh Fadatare (Java Guides)

4 Primary Key Generation Strategies

1. GenerationType.AUTO

2. GenerationType.IDENTITY

3. GenerationType.SEQUENCE

4. GenerationType.TABLE

1. GenerationType.AUTO

The GenerationType.AUTO is the default generation type and lets the persistence provider choose the generation strategy.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "id")
private Long id;
```

If you use Hibernate as your persistence provider, it selects a generation strategy based on the database-specific dialect.

For most popular databases, it selects GenerationType.SEQUENCE which I will explain in a further section.

2. GenerationType.IDENTITY

```java
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id")
private Long id;
```

It relies on an auto-incremented database column and lets the database generate a new value with each insert operation.

From a database point of view, this is very efficient because the auto-increment columns are highly optimized, and it doesn’t require any additional statements.

Not good for JDBC batch operations

3. GenerationType.SEQUENCE

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,
        generator = "product_generator")
@SequenceGenerator(name = "product_generator",
        sequenceName = "product_sequence_name",
        allocationSize = 1)
```

The GenerationType.SEQUENCE is to generate primary key values and uses a database sequence to generate unique values.

It requires additional select statements to get the next value from a database sequence. But this has no performance impact on most applications.

The @SequenceGenerator annotation lets you define the name of the generator, the name, and schema of the database sequence and the allocation size of the sequence.

4. GenerationType.TABLE

```java
@Id
@GeneratedValue(strategy = GenerationType.TABLE)
@Column(name = "id")
private Long id;
```

The *GenerationType.TABLE* gets only rarely used nowadays.

It simulates a sequence by storing and updating its current value in a database table which requires the use of pessimistic locks which put all transactions into a sequential order.

This slows down your application, and you should, therefore, prefer the *GenerationType.SEQUENCE*, if your database supports sequences, which most popular databases do.