The JPA specification supports 4 different primary key generation strategies that generate the primary key values programmatically or use database features, like auto-incremented columns or sequences.

**4 Options to Generate Primary Keys**

The JPA specification supports 4 different primary key generation strategies that generate the primary key values programmatically or use database features, like auto-incremented columns or sequences. The only thing you have to do is to add the *@GeneratedValue* annotation to your primary key attribute and choose a generation strategy.

@Id

@GeneratedValue

@Column(name = "id", updatable = false, nullable = false)

private Long id;

**1. GenerationType.AUTO**

The *GenerationType.AUTO* is the default generation type and lets the persistence provider choose the generation strategy.  
For example, consider we have a *Student* JPA entity class with *GenerationType.AUTO* as generation type:

@Entity

@Table(name = "student")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(name = "id", updatable = false, nullable = false)

private Long id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "email")

private String email;

public Student() {

}

// getter and setters

}

If you use [**Hibernate**](http://www.javaguides.net/p/hibernate-tutorial.html) 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**

The *GenerationType.IDENTITY* is the easiest to use but not the best one from a performance point of view. 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.  
  
For example, consider we have a *Student* JPA entity class with *GenerationType.IDENTITY* as a generation type:

@Entity

@Table(name = "student")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id", updatable = false, nullable = false)

private Long id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "email")

private String email;

public Student() {

}

// getter and setters

}

**3. GenerationType.SEQUENCE**

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. And if your application has to persist a huge number of new entities, you can use some [**Hibernate**](http://www.javaguides.net/p/hibernate-tutorial.html) specific optimizations to reduce the number of statements.  
  
For example, consider we have a *Student* JPA entity class with *GenerationType.SEQUENCE*as generation type:

@Entity

@Table(name = "student")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

@Column(name = "id", updatable = false, nullable = false)

private Long id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "email")

private String email;

public Student() {

}

// getter and setters

}

If you don’t provide any additional information, [**Hibernate**](http://www.javaguides.net/p/hibernate-tutorial.html) will request the next value from its default sequence. You can change that by referencing the name of a *@SequenceGenerator* in the generator attribute of the *@GeneratedValue* annotation. 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.

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "book\_generator")

@SequenceGenerator(name="book\_generator", sequenceName = "book\_seq", allocationSize=50)

@Column(name = "id", updatable = false, nullable = false)

private Long id;

**4.GenerationType.TABLE**

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 that put all transactions into 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.  
  
For example, consider we have a *Student* JPA entity class with *GenerationType.TABLE*as generation type:

@Entity

@Table(name = "student")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.TABLE)

@Column(name = "id", updatable = false, nullable = false)

private Long id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "email")

private String email;

public Student() {

}

// getter and setters

}

You can use the *@TableGenerator* annotation in a similar way as the already explained *@SequenceGenerator* annotation to specify the database table which [**Hibernate**](http://www.javaguides.net/p/hibernate-tutorial.html) shall use to simulate the sequence.

@Id

@GeneratedValue(strategy = GenerationType.TABLE, generator = "book\_generator")

@TableGenerator(name="book\_generator", table="id\_generator", schema="bookstore")

@Column(name = "id", updatable = false, nullable = false)

private Long id;

**Summary**

As you’ve seen, JPA offers 4 different ways to generate primary key values:

1. *GenerationType.AUTO*: Hibernate selects the generation strategy based on the used dialect,
2. *GenerationType.IDENTITY*: Hibernate relies on an auto-incremented database column to generate the primary key,
3. *GenerationType.SEQUENCE*: Hibernate requests the primary key value from a database sequence,
4. *GenerationType.TABLE*: Hibernate uses a database table to simulate a sequence.