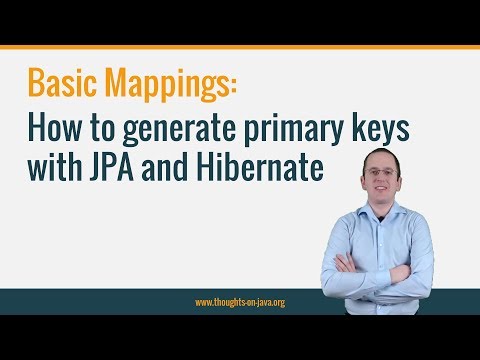
**How to generate primary keys with JPA and Hibernate[[1]](#footnote-1)**

How do you get the primary key values in your application? Do you use natural keys or do you generate technical IDs?

**Don’t want to read? You can watch it here!**



I prefer to generate simple, numerical, technical IDs like you can see in the following code snippet instead of using natural keys which often require the combination of multiple attributes.

|  |
| --- |
| @Id  @Column(name = "id", updatable = false, nullable = false)  private Long id; |

Technical IDs are easier to manage and all involved systems, mainly the database and Hibernate, can index them very efficiently. This allows you to focus on the business logic of your application and avoids performance issues.

**4 options to generate primary keys**

The JPA specification supports 4 different primary key generation strategies which 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; |

**GenerationType.AUTO**

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

|  |
| --- |
|  |
| @Id | |
|  | |

|  |
| --- |
| @GeneratedValue(strategy = GenerationType.AUTO) |
|  |

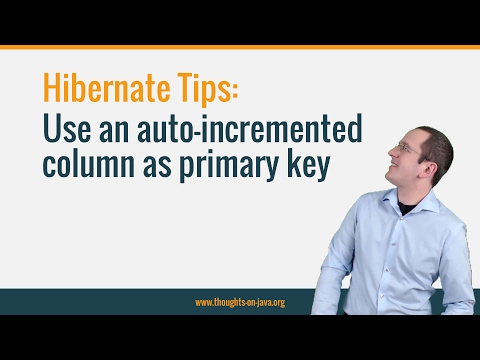
|  |
| --- |
| @Column(name = "id", updatable = false, nullable = false) |
|  |

|  |
| --- |
| private Long id; |

[view raw](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101/raw/ccfb68155cb886942682446bf82d388c4a22623a/GenerationTypeAuto.java) [GenerationTypeAuto.java](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101#file-generationtypeauto-java) hosted with ❤ by [GitHub](https://github.com)

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 later.

Watch it on YouTube



**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.

|  |
| --- |
|  |
| @Id | |
|  | |

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

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

|  |
| --- |
| private Long id; |

[view raw](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101/raw/ccfb68155cb886942682446bf82d388c4a22623a/GenerationTypeIdentity.java) [GenerationTypeIdentity.java](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101#file-generationtypeidentity-java) hosted with ❤ by [GitHub](https://github.com)

This approach has a significant drawback if you use Hibernate. Hibernate requires a primary key value for each managed entity and therefore has to perform the insert statement immediately. This prevents it from using [different optimization techniques](https://thoughts-on-java.org/course-hibernate-performance-tuning/) like JDBC batching.

Watch it on YouTube



**GenerationType.SEQUENCE**

The *GenerationType.SEQUENCE* is my preferred way 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 for most applications. And if your application has to persist a huge number of new entities, you can use some [Hibernate specific optimizations](https://thoughts-on-java.org/course-hibernate-performance-tuning/) to reduce the number of statements.

|  |
| --- |
|  |
| @Id | |
|  | |

|  |
| --- |
| @GeneratedValue(strategy = GenerationType.SEQUENCE) |
|  |

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

|  |
| --- |
| private Long id; |

[view raw](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101/raw/ccfb68155cb886942682446bf82d388c4a22623a/GenerationTypeSequence.java) [GenerationTypeSequence.java](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101#file-generationtypesequence-java) hosted with ❤ by [GitHub](https://github.com)

If you don’t provide any additional information, Hibernate 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; |

[view raw](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101/raw/ccfb68155cb886942682446bf82d388c4a22623a/SequenceGenerator.java) [SequenceGenerator.java](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101#file-sequencegenerator-java) hosted with ❤ by [GitHub](https://github.com)

**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 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.

|  |
| --- |
|  |
| @Id | |
|  | |

|  |
| --- |
| @GeneratedValue(strategy = GenerationType.TABLE) |
|  |

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

|  |
| --- |
| private Long id; |

[view raw](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101/raw/ccfb68155cb886942682446bf82d388c4a22623a/GenerationTypeTable.java) [GenerationTypeTable.java](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101#file-generationtypetable-java) hosted with ❤ by [GitHub](https://github.com)

You can use the *@TableGenerator* annotation in a similar way as the already explained *@SequenceGenerator* annotation to specify the database table which Hibernate 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; |

[view raw](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101/raw/ccfb68155cb886942682446bf82d388c4a22623a/TableGenerator.java) [TableGenerator.java](https://gist.github.com/thjanssen/75d9c920ce3c6c67b136043ca0860101#file-tablegenerator-java) hosted with ❤ by [GitHub](https://github.com)

2 Free Ebooks

​to boost your Hibernate skills

Sign up below to join the Thoughts on Java Newsletter and to get your free ebooks:

* Java 8 Support in Hibernate 5
* Native Queries with Hibernate

Top of Form

Bottom of Form

Top of Form

Bottom of Form

I will collect, use and protect your data in accordance with my [Privacy Policy](https://thoughts-on-java.org/datenschutz).

Already a member? [Login here.](https://thoughts-on-java.org/login/?redirect_to=http%3A%2F%2Fwww.thoughts-on-java.org%2Fcheat-sheets%2F%23977149b6406f10d261aa6a5f8a8339f2)

**Summary**

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

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

I prefer to use the *GenerationType.SEQUENCE* because it is very efficient and allows Hibernate to decide when to perform the insert statement. This provides the required flexibility to use other performance optimization techniques like JDBC batching.  
When you like to learn more about performance tuning and how Hibernate can optimize the *GenerationType.SEQUENCE*, have a look at my [Hibernate Performance Tuning Online Training](https://thoughts-on-java.org/course-hibernate-performance-tuning/).

1. https://thoughts-on-java.org/jpa-generate-primary-keys/ [↑](#footnote-ref-1)