How to generate UUIDs as primary keys with Hibernate

By Thorben Janssen — 7 Comments

tweet share share share share share e-mail

Most developers prefer numerical primary keys because they are efficient to use and easy to generate. But that doesn't mean that a primary key has to be a number.

UUIDs, for example, have gained some popularity over the recent years. The main advantage of a UUID is its (practical) global uniqueness which provides a huge advantage for distributed systems.

If you use the typical, numerical ID that gets incremented for each new record, you need to generate all IDs by the same component of your system or the components need to communicate with each other. With a globally unique UUID, you don't need all of this. Each component can generate a UUID and there will not be any conflicts.

Follow me on YouTube to not miss any new videos.

On the other hand, the UUID also has some disadvantages. The most obvious one is its size. It's 4 times larger than a numerical ID and can't be handled as efficiently. You should, therefore, decide carefully if you want to use UUIDs or numeric IDs and discuss it with your database administrator.

If you decide to use UUIDs, you can, of course, also persist them with Hibernate. With just a few additional annotations, you can even let Hibernate generate the UUIDs for you.

Generate UUIDs with Hibernate

The IETF RFC 4122 defines 4 different strategies to generate UUIDs. Hibernate's UUIDGenerator supports 2 of them:

1. The default strategy generates the UUID based on random numbers (IETF RFC 4122 Version 4).
2. You can also configure a generator that uses the IP address of the machine and a timestamp (IETF RFC 4122 Version 1).

Let's have a look at the default strategy first.

Random number based UUID (IETF RFC 4122 version 4)

By default, Hibernate uses a random number based generation strategy. As always, you don't have to do much to get the default behaviour. You just need to add a *@GeneratedValue* annotation that references the ID generator to your primary key attribute and define the generator with one of Hibernate's *@GenericGenerator* annotations. The *@GenericGenerator* annotation requires 2 parameters, the name of the generator and the name of the class that implements the generator. In this case, it's the *org.hibernate.id.UUIDGenerator*.

That's all you have to do to let Hibernate generate UUIDs as primary key values. You can see an example of it in the following code snippet.

```
	)
	@Column(name = "id", updatable = false, nullable = false)
	private UUID id;

	…
}
```

UUIDVersion4.java hosted with ❤ by GitHub view raw

When you now persist a new *Book* entity, Hibernate generates a UUID before writing the new record to the database.

```
Book b = new Book();
b.setTitle("The Hound of the Baskervilles");
b.setPublishingDate(LocalDate.of(1902, 4, 30));
em.persist(b);
```

PersistBookEntity.java hosted with ❤ by GitHub view raw

```
12:23:19,356 DEBUG AbstractSaveEventListener:118 – Generated identifier: d7cd23b8-991c-470f-ac63-d8fb106f391e, using
12:23:19,388 DEBUG SQL:92 – insert into Book (publishingDate, title, version, id) values (?, ?, ?, ?)
12:23:19,392 TRACE BasicBinder:65 – binding parameter [1] as [DATE] – [1902-04-30]
12:23:19,393 TRACE BasicBinder:65 – binding parameter [2] as [VARCHAR] – [The Hound of the Baskervilles]
12:23:19,393 TRACE BasicBinder:65 – binding parameter [3] as [INTEGER] – [0]
12:23:19,394 TRACE BasicBinder:65 – binding parameter [4] as [OTHER] – [d7cd23b8-991c-470f-ac63-d8fb106f391e]
```

UUIDVersion4.log hosted with ❤ by GitHub view raw

IP and timestamp based UUID (IETF RFC 4122 version 1)

Hibernate can also generate a UUID based on IETF RFC 4122 version 1. If you follow the specification, you should generate the UUID with the MAC address instead of the IP address. As long as nobody is messing around with it, the MAC address of each device should be unique and due to this help to create a unique UUID.

Hibernate uses the IP address instead of the MAC address. In general, this is not an issue. But if the servers of your distributed system are running in different networks you should make sure that none of them share the same IP address.

```java
public class Book {

        @Id
        @GeneratedValue(generator = “UUID”)
        @GenericGenerator(
                name = “UUID”,
                strategy = “org.hibernate.id.UUIDGenerator”,
                parameters = {
                        @Parameter(
                                name = “uuid_gen_strategy_class”,
                                value = “org.hibernate.id.uuid.CustomVersionOneStrategy”
                        )
                }
        )
        @Column(name = “id”, updatable = false, nullable = false)
        private UUID id;

        …
}
```

UUIDVersion1.java hosted with ❤ by GitHub view raw

When you now persist the new *Book* entity, Hibernate will use the *CustomVersionOneStrategy* class to generate the UUID based on IETF RFC 4122 version 1.

```java
Book b = new Book();
b.setTitle(“The Hound of the Baskervilles”);
b.setPublishingDate(LocalDate.of(1902, 4, 30));
em.persist(b);
```

PersistBookEntity.java hosted with ❤ by GitHub view raw

```
12:35:22,760 DEBUG AbstractSaveEventListener:118 - Generated identifier: c0a8b214-578f-131a-8157-8f431d060000, using
12:35:22,792 DEBUG SQL:92 - insert into Book (publishingDate, title, version, id) values (?, ?, ?, ?)
12:35:22,795 TRACE BasicBinder:65 - binding parameter [1] as [DATE] - [1902-04-30]
12:35:22,795 TRACE BasicBinder:65 - binding parameter [2] as [VARCHAR] - [The Hound of the Baskervilles]
12:35:22,796 TRACE BasicBinder:65 - binding parameter [3] as [INTEGER] - [0]
12:35:22,797 TRACE BasicBinder:65 - binding parameter [4] as [OTHER] - [c0a8b214-578f-131a-8157-8f431d060000]
```

UUIDVersion1.log hosted with ❤ by GitHub view raw

Summary

As you've seen, you can also use UUIDs as primary keys and let Hibernate handle the value generation. Hibernate's UUIDGenerator supports the creation of version 1 and version 4 UUIDs as defined by IETF RFC 4122. By default, it generates version 4 UUIDs which is a good fit for most use cases.

Unfortunately, this is not part of the JPA specification and will require some adaptions, if you need to switch your JPA implementation.

Implement Your Persistence Layer with Ease

Learn More About Hibernate

Need Some Help with Your Project?

Comments

Daniel Pozzi says
October 6, 2016 at 1:25 pm

Does hibernate automatically know the best way to store the UUID? I've seen this suggestion on stackoverflow:

@Column(name = "id", columnDefinition = "BINARY(16)")

Reply

Thorben Janssen says
October 6, 2016 at 1:46 pm

That depends on the database specific dialect. If you're using PostgreSQL (as I do in this example), Hibernate PostgreSQL dialect will map it to the PostgreSQL-specific UUID datatype.

Reply

Steve Ebersole says
October 14, 2016 at 2:08 am

donald.coffin@reminetworks.com says
November 5, 2017 at 5:06 pm

Is it possible to generate a Version 5 UUID using Hibernate?

Reply

Thorben Janssen says
November 21, 2017 at 7:48 pm

No, unfortunately not.

Reply

Menai Ala Eddine says
March 21, 2018 at 4:21 pm

Hi Thorben Janssen ,You have greal tuto ,this post helped me to understand UUID ,i will read all posts about Hibernate ,you give a great descriptions .
Best regards

Reply

Thorben Janssen says
April 21, 2018 at 2:58 pm

Thanks 🙂

Reply

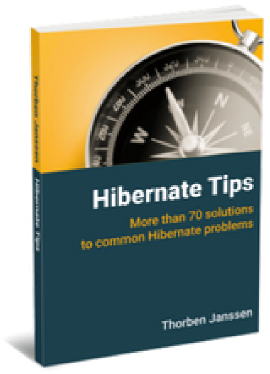
Name *

Email *

Website

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed.



LET'S CONNECT

Thorben Janssen
Independent consultant, trainer and author

Hibernate für Fortgeschrittene (1-day Workshop - German)

12th September 2018 Workshop-Tage 2018 (Switzerland):
Hibernate + jOOQ + Flyway = Die besten relationalen Persistenzframeworks in einem Stack (1-day Workshop - German)

13th September 2018 Workshop-Tage 2018 (Switzerland):
Spring Data JDBC – Der neue Stern am Persistenzhimmel? (1-day Workshop - German)

9th-10th October 2018 Geecon (Poland):
Performance Tuning with JPA 2.2 and Hibernate Workshop (2-day Workshop - English)

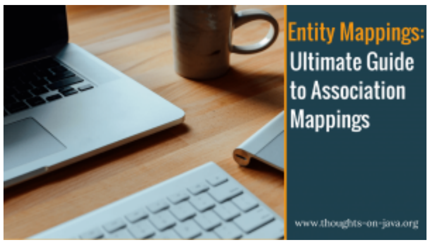
5th-9th November 2018 W-JAX 2018 (Germany):
Hibernate-Workshop – Performance-Tuning für Enterprise-Anwendungen (1-day Workshop - German

5th-9th November 2018 W-JAX 2018 (Germany):
Das Hibernate-Universum – Fernab bekannter CRUD-Galaxien (Talk - German)

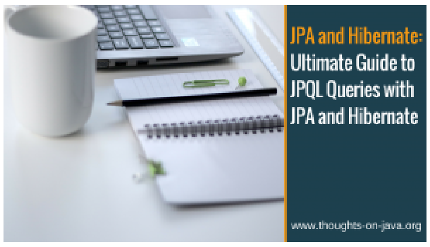
Looking for an on-site training?

Getting Started With Hibernate

12 Java YouTube Channels You Should Follow In 2018



Ultimate Guide – Association Mappings with JPA and Hibernate



Ultimate Guide to JPQL Queries with JPA and Hibernate

Hibernate Best Practices

RECENT POSTS

Hibernate Tips: Permanently remove records when using soft delete

How to lazily load non-relational attributes in a portable way

Hibernate Tips: Map an Unidirectional One-to-Many Association Without a Junction Table

Hibernate Tips: Get the SQL Connection used by your Hibernate Session

5 Reasons and 101 Bugfixes – Why You Should Use Hibernate 5.3

Update your Database Schema Without Downtime

Hibernate Tips: How to Call a Function that returns a SYS_REFCURSOR

THOUGHTS ON JAVA