5 Primary Key Mappings Every Developer Should Know

With JPA and Hibernate you can do much more than just mapping a simple, numeric primary key. You can:

- choose between different strategies to generate unique primary key values,
- use UUIDs and generate their values,
- map composite primary keys,
- share primary key values across associations and
- map natural IDs.

Generate Numeric Primary Key Values

You can either set primary key values programmatically or use one of JPA's generation strategies to create them automatically. The easiest way to do that is to annotate your primary key attribute with a @GeneratedValue annotation. Hibernate will then pick a strategy based on the database-specific dialect.

But using the auto strategy, or not referencing a strategy at all, is the simplest but not the best way. It's better to specify the strategy. You can choose between:

- GenerationType.AUTO – Let Hibernate pick one of the following strategies.
- GenerationType.SEQUENCE – Use a database sequence.
- GenerationType.IDENTITY – Use an autoincremented database columns.
- GenerationType.TABLE – Use a database table to simulate a sequence.

That ensures that a Hibernate update will not accidentally change your generation strategy and if you're using the GenerationType.SEQUENCE, it will also activate Hibernate's performance optimizations.

Defining the strategy is simple. You just need to provide it as the value of the strategy attribute of the @GeneratedValue annotation.

```
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    …
}
```

By default, Hibernate uses a sequence called hibernate_sequence. You can also tell Hibernate to use one of your own database sequences. I explained that in more details in Hibernate Tips: How to use a custom database sequence.

Generate UUID Primary Keys

UUIDs and numerical primary keys might seem very different. But with Hibernate, you can map and use them in almost the same way. The only difference is the type of the primary key attribute, which is a java.util.UUID instead of a java.lang.Long.

Here is a simple example. The Book entity maps an attribute of type UUID and uses one of Hibernate's generators to create primary key values automatically before persisting a new entity.

```
@Entity
public class Book {

    @Id
    @GeneratedValue
    private UUID id;

    …
}
```

Manage Composite Primary Keys

JPA and Hibernate also provide multiple ways to map composite primary keys that consist of multiple attributes. Let's take a look at my preferred option: the embedded id.

The embedded id approach uses an embeddable to map the primary key attributes. An embeddable is a pure Java class that is annotated with @Embeddable. It defines attribute mappings in a reusable way. If you want to use it as an embedded id, you also need to implement the equals and hashCode methods.

```
@Embeddable
public class AddressKey implements Serializable {

    private Long xId;
    private Long yId;

    public AddressKey() {}

    public AddressKey(Long xId, Long yId) {
        this.xId = xId;
        this.yId = yId;
    }

    // getter and setter methods

    @Override
    public int hashCode() {
        int result = 1;
        result = 31 * result + ((xId == null) ? 0 : xId.hashCode());
        result = 31 * result + ((yId == null) ? 0 : yId.hashCode());
        return result;
    }
```

```
     @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        AddressKey other = (AddressKey) obj;
        if (xId == null) {
            if (other.xId != null)
                return false;
        } else if (!xId.equals(other.xId))
            return false;
        if (yId == null) {
            if (other.yId != null)
                return false;
        } else if (!yId.equals(other.yId))
            return false;
        return true;
    }
}
```

You can then use the embeddable class as the type of your primary key attribute and annotate it with @EmbeddedId. The embeddable and all its attributes become part of the entity. It follows the same lifecycle, and all its attributes get mapped to the database table that's mapped by the entity.

```
@Entity
public class Address {

    @EmbeddedId
    private AddressKey id;

    private String city;

    private String street;

    private String country;

    @OneToOne(mappedBy = "address")
    private Person person;

    ...
}
```

Use Same Primary Key Values for Associated Entities

Another common primary key mapping is to use the same primary key value in a one-to-one association.

You can, of course, map this with JPA and Hibernate. The only things you need to do are to model the owning side of the association on the entity that shall reuse the primary key value and to add a @MapsId annotation to it.

```
@Entity
public class Manuscript {

    @Id
    private Long id;

    private byte[] file;

    @OneToOne
    @JoinColumn(name = "id")
    @MapsId
    private Book book;

    ...
}
```

Work with Natural IDs

Most teams prefer to use a surrogate key as the primary key. It's easier to manage in your code, and all involved systems can handle it more efficiently. But modeling a natural ID is still useful. You will, most likely, reference them very often in your use cases.

Hibernate provides an annotation to declare a natural ID and an API to retrieve entities by it. Let's take a quick look at the most important details. And if you want to dive deeper, please read my article @NaturalId – A good way to persist natural IDs with Hibernate?

You can specify a natural ID by annotating one or more entity attributes with @NaturalId. I use it in the following code snippet, to tell Hibernate that the isbn attribute is a natural ID of the Book entity.

```
Entity
public class Book {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private Long id;

  @NaturalId
  private String isbn;

  …
}
```

After you've done that, you can use the byNaturalId method on Hibernate's Session interface to create a query that loads an entity by its natural id.

In the next step, you need to provide the value of the natural id by calling the using method for each attribute that's part of the natural id. In this example, the natural id only consists of the isbn attribute, which I reference using the JPA metamodel class of the Book entity.

And after you provided the natural id value, you can call the load method to execute the query.

```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Session session = em.unwrap(Session.class);

Book b = session.byNaturalId(Book.class)
          .using(Book_.isbn.getName(), "978-0321356680").load();
```