JPA Inheritance

Rodrigo García Carmona (v1.1.3)

Relational databases don't feature object-oriented inheritance, so using super and subclasses with JPA can be tricky. There is no standard way to implement inheritance in a database, and therefore we must choose how we're going to represent this feature in the relational model.

JPA supports three inheritance models, defined trough the Inheritance annotation:

- **One table for all classes:** *SINGLE_TABLE*
- **One table for each class:** *JOINED*
- **One table for each subclass:** *TABLE_PER_CLASS*

Single Table Inheritance

This is the default inheritance strategy used if none is specified. It's the simplest and typically the best performing solution... if you can tune your tables to add a discriminator column.

In this strategy, a single table is used to store all instances of the entire inheritance hierarchy. The table will have a column for every attribute of every class in the hierarchy and a discriminator column; used to determine which class the particular row belongs to. This column will contain a value specific to each class.

A single table looks like this:

project

|id |proj_type |name |budget |----|----------|-----------|------ |1 |L |Marketing |5000.0 |2 |S |Legal |null

And is implemented using these classes:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="proj_type")
@ForceDiscriminator
@Table(name="projects")
public abstract class Project {
    @Id
    // Other Id annotations as needed
    private Integer id;
    ...
}

@Entity
@DiscriminatorValue("L")
public class LargeProject extends Project, implements Serializable {
    private Double budget;
}

@Entity
@DiscriminatorValue("S")
public class SmallProject extends Project, implements Serializable {
}
```

One Table Per Class Inheritance

This inheritance strategy is used when we have a relational model that closely follows the object model, so it's usually employed when the object model is designed first.

With this strategy, a table is defined for each class in the inheritance hierarchy, including the parent class. Each table only contains the attributes specific to the class it represents, and the parent class' table must contain a discriminator column, like in the single table strategy.

Each table must also store the object's primary key, but this is only defined in the root class. All classes in the hierarchy must share the same id attribute.

A set of tables for this strategy look like this:

projects

|id |proj_type |name |---|-----------|--------- |1 |L |Marketing |2 |S |Legal

smallprojects

|id| |--- |2 |

largeprojects

|id |budget |---|------ |1 |5000.0

And are implemented using these classes:

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="proj_type")
@Table(name="projects")
public abstract class Project {
    @Id
    // Other Id annotations as needed
    private integer id;
    ...
}

@Entity
@DiscriminatorValue("L")
@Table(name="largeprojects")
public class LargeProject extends Project, implements Serializable {
    private Double budget;
}

@Entity
@DiscriminatorValue("S")
@Table(name="smallprojects")
public class SmallProject extends Project, implements Serializable {
}
```

One Table Per Subclass Inheritance

This inheritance strategy is used when we want to keep the relational model oblivious to it, since no discriminator column is required and each class is in a different, completely separated, table.

In this strategy, we use an abstract parent class and several children classes. Each child class is realized as a table that stores all the attributes of that class and the superclass.

Here's an example of a possible table structure:

smallprojects

|id |name |---|----- |2 |Legal

largeprojects

|id |name |budget |---|-----------|------ |1 |Marketing |5000.0

Which is implemented using these classes:

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Project {
    @Id
    // No other Id annotations should be put here
    private Integer id;
    @Column(name="name")
    private String name;
    ...
}

@Entity
@Table(name="largeprojects")
public class LargeProject extends Project, implements Serializable {
    @Id
    // Other Id annotations as needed
    private Integer id;
    private Double budget;
    ...
}

@Entity
@Table(name="smallprojects")
public class SmallProject extends Project, implements Serializable {
    @Id
    // Other Id annotations as needed
    private Integer id;
    ...
}
```