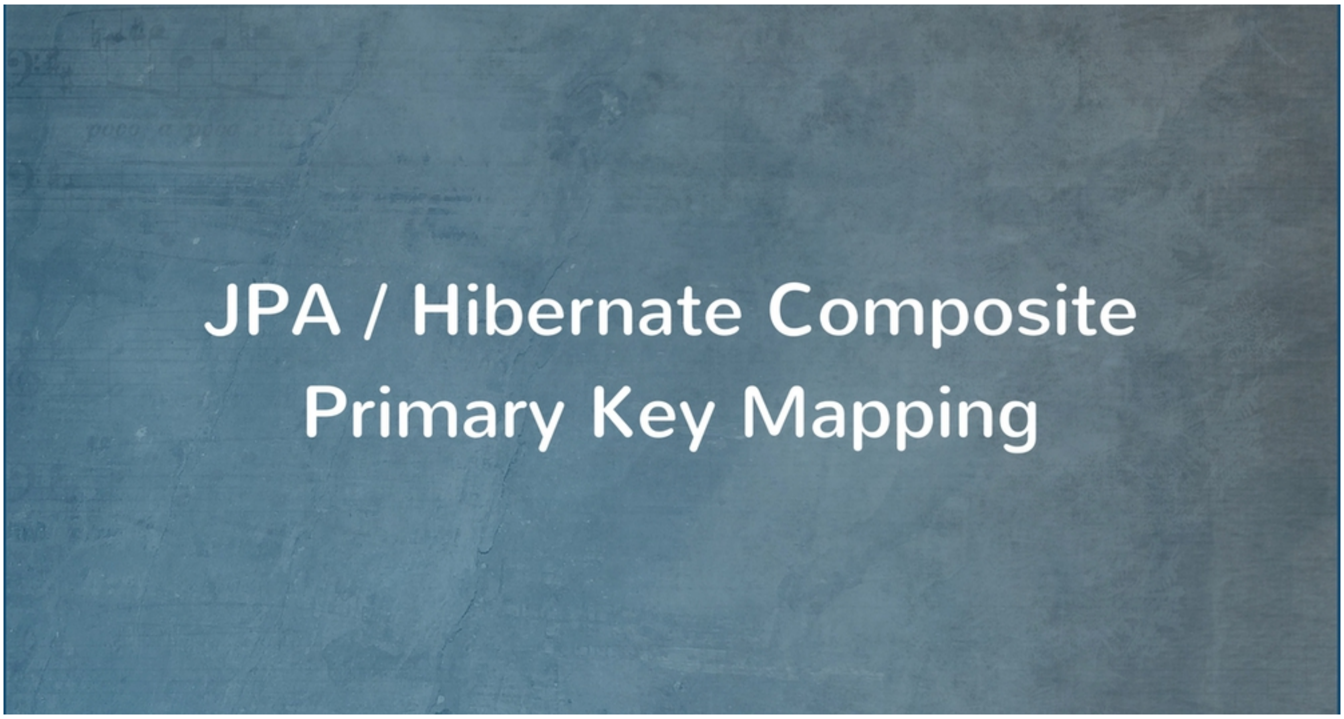
JPA / Hibernate Composite Primary Key Example with Spring Boot

Rajeev Singh • Spring Boot • Nov 26, 2017 • 6 mins read

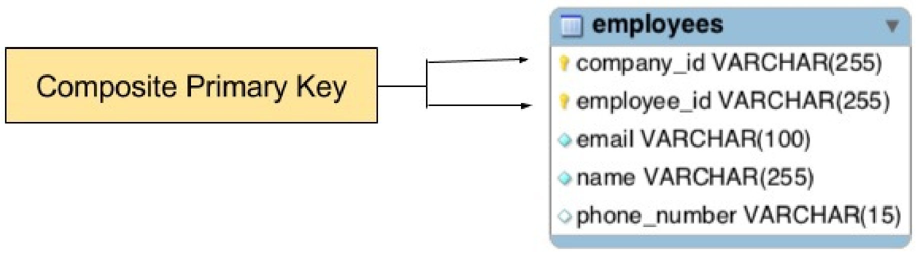


In this article, You'll learn how to map a composite primary key in Hibernate using JPA's `@Embeddable` and `@EmbeddedId` annotations.

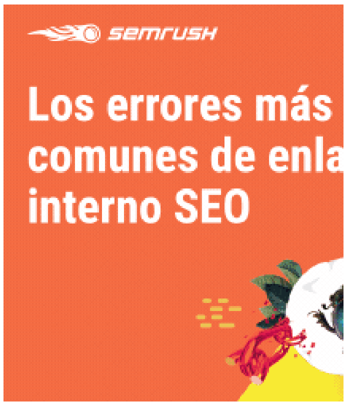
Let's say that We have an application that manages Employees of various companies. Every employee has a unique employeeId within his company. But the same employeeId can be present in other companies as well, So we can not uniquely identity an employee just by his employeeId.

To identify an employee uniquely, we need to know his employeeId and companyId both. Check out the following `Employees` table that contains a composite primary key which includes both the employeeId and companyId columns -

CALLICODER



Let’s create a project from scratch and learn how to map such composite primary key using JPA and Hibernate.



Creating the Project

You can generate the project quickly using Spring Boot CLI by typing the following command in the terminal -

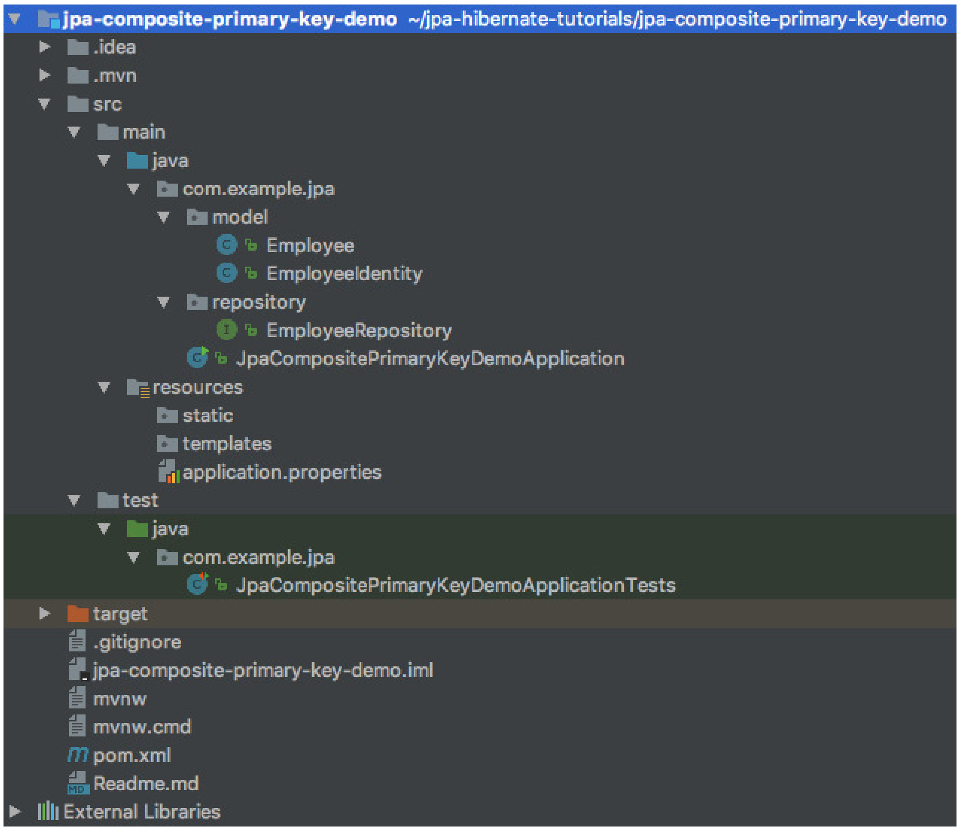
```
spring init -n=jpa-composite-primary-key-demo -d=web,jpa,mysql --package-name=com.example.jpa jpa-com
```

Alternatively, You can also use Spring Initializr web app to generate the project. Follow the instructions below to generate the app using Spring Initializr web app -

1. Open http://start.spring.io
2. Enter **Artifact** as “jpa-composite-primary-key-demo”
3. Click **Options** dropdown to see all the options related to project metadata.

6. Click **Generate** to generate and download the project.

Following is the directory structure of the complete application for your reference -



```
jpa-composite-primary-key-demo ~/jpa-hibernate-tutorials/jpa-composite-primary-key-demo
  .idea
  .mvn
  src
    main
      java
        com.example.jpa
          model
            Employee
            EmployeeIdentity
          repository
            EmployeeRepository
          JpaCompositePrimaryKeyDemoApplication
      resources
        static
        templates
        application.properties
    test
      java
        com.example.jpa
          JpaCompositePrimaryKeyDemoApplicationTests
  target
  .gitignore
  jpa-composite-primary-key-demo.iml
  mvnw
  mvnw.cmd
  pom.xml
  Readme.md
External Libraries
```

(*Your bootstrapped project won't have* `model` *and* `repository` *packages and all the other classes. We'll create them as we proceed to next sections*)

Configuring the Database and Hibernate Log Levels

Let's add the MySQL database URL, username and password configurations in `src/main/resources/application.properties` file -

```
# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url=jdbc:mysql://localhost:3306/jpa_composite_pk_demo?useSSL=false&serverTimezone=U
spring.datasource.username=root
spring.datasource.password=root
```

CALLICODER

```
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=TRACE
```

Apart from MySQL database configurations, I've also specified hibernate log levels and other properties.

The property `spring.jpa.hibernate.ddl-auto = update` keeps the Entity types in your application and the mapped database tables in sync. Whenever you update a domain entity, the corresponding mapped table in the database will also get updated when you restart the application next time.

This is great for development because you don't need to manually create or update the tables. They will automatically be created/updated based on the Entity classes in your application.

Before proceeding to the next section, Please make sure that you create a MySQL database named `jpa_composite_pk_demo` and change `spring.datasource.username` and `spring.datasource.password` properties as per your MySQL installation.

Defining the Domain model

A composite primary key is mapped using an Embeddable type in hibernate. We'll first create an Embeddable type called `EmployeeIdentity` containing the employeeId and companyId fields, and then create the `Employee` entity which will embed the `EmployeeIdentity` type.

Create a new package named `model` inside `com.example.jpa` package and then add the following classes inside the `model` package -

1. EmployeeIdentity - Embeddable Type

```
package com.example.jpa.model;

import javax.persistence.Embeddable;
import javax.validation.constraints.NotNull;
```

CALLICODER

```
@Embeddable
public class EmployeeIdentity implements Serializable {
    @NotNull
    @Size(max = 20)
    private String employeeId;

    @NotNull
    @Size(max = 20)
    private String companyId;

    public EmployeeIdentity() {

    }

    public EmployeeIdentity(String employeeId, String companyId) {
        this.employeeId = employeeId;
        this.companyId = companyId;
    }

    public String getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(String employeeId) {
        this.employeeId = employeeId;
    }

    public String getCompanyId() {
        return companyId;
    }

    public void setCompanyId(String companyId) {
        this.companyId = companyId;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!employeeId.equals(that.employeeId)) return false;
        return companyId.equals(that.companyId);
    }

    @Override
    public int hashCode() {
        int result = employeeId.hashCode();
        result = 31 * result + companyId.hashCode();
        return result;
    }
}
```

2. Employee - Domain model

```
package com.example.jpa.model;

import org.hibernate.annotations.NaturalId;
import javax.persistence.Column;
import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Email;
import javax.validation.constraints.Size;

@Entity
@Table(name = "employees")
public class Employee {

    @EmbeddedId
    private EmployeeIdentity employeeIdentity;

    @NotNull
    @Size(max = 60)
    private String name;
```

CALLICODER

```
    @Size(max = 60)
    private String email;

    @Size(max = 15)
    @Column(name = "phone_number", unique = true)
    private String phoneNumber;

    public Employee() {

    }

    public Employee(EmployeeIdentity employeeIdentity, String name, String email, String phoneNumber)
        this.employeeIdentity = employeeIdentity;
        this.name = name;
        this.email = email;
        this.phoneNumber = phoneNumber;
    }

    // Getters and Setters (Omitted for brevity)
}
```

In the `Employee` class, We use `@EmbeddedId` annotation to embed the `EmployeeIdentity` type and mark it as a primary key.

Creating the Repository

CALLICODER

`repository` package -

```
package com.example.jpa.repository;

import com.example.jpa.model.Employee;

import com.example.jpa.model.EmployeeIdentity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, EmployeeIdentity> {

}
```

Code to test the Composite Primary Key Mapping

Finally, Let’s write some code to test the composite primary key mapping. Open the main class `JpaCompositePrimaryKeyDemoApplication.java` and replace it with the following code -

```
package com.example.jpa;

import com.example.jpa.model.Employee;
import com.example.jpa.model.EmployeeIdentity;
import com.example.jpa.repository.EmployeeRepository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class JpaCompositePrimaryKeyDemoApplication implements CommandLineRunner {

    @Autowired
    private EmployeeRepository employeeRepository;
```

CALLICODER

```
    @Override
    public void run(String... args) throws Exception {
        // Cleanup employees table
        employeeRepository.deleteAllInBatch();

        // Insert a new Employee in the database
        Employee employee = new Employee(new EmployeeIdentity("E-123", "D-457"),
                "Rajeev Kumar Singh",
                "rajeev@callicoder.com",
                "+91-9999999999");

        employeeRepository.save(employee);
    }
}
```

We first clean up the `Employee` table and then insert a new Employee record with an employeeId and a companyId to test the setup.

You can run the application by typing `mvn spring-boot:run` from the root directory of the project. The `Employee` record will be inserted in the database once the application is successfully started.

Querying using the Composite Primary Key

Let’s now see some query examples using the composite primary key -

1. Retrieving an Employee using the composite primary key - (employeeId and companyId)

```
// Retrieving an Employee Record with the composite primary key
employeeRepository.findById(new EmployeeIdentity("E-123", "D-457"));
```

2. Retrieving all employees of a particular company

Let’s say that you want to retrieve all the employees of a company by companyId. For doing this, just add the following method in the `EmployeeRepository` interface.

```
@Repository
```

```
        and create a query from it
    */
    List<Employee> findByEmployeeIdentityCompanyId(String companyId);
}
```

That’s all! You don’t need to implement anything. Spring Data JPA will dynamically generate a query using the method name. You can use the above method in the main class to retrieve all the employees of a company like this -

```
// Retrieving all the employees of a particular company
employeeRepository.findByEmployeeIdentityCompanyId("D-457");
```

Conclusion

Congratulations guys! In this article, you learned how to implement a composite primary key in hibernate using `@Embeddable` and `@EmbeddedId` annotations.

You can find the entire source code for the sample project that we built in this article in my jpa-hibernate-tutorials github repository.

Thanks for reading. See you in the next post.

Liked the Article? Share it on Social media!

CALLICODER

CalliCoder

Software Development Tutorials written from the heart!

RESOURCES

About & Contact

Advertise

Recommended Books

Recommended Courses

Privacy Policy

Sitemap

TOOLS

URL Encoder

URL Decoder

Base64 Encoder

Base64 Decoder

QRCodeBit

ASCII Table

JSON Formatter

CONNECT

Twitter

Github

Facebook

Linkedin

Reddit

ALSO ON **CALLICODER**

Two Number Sum Problem solution in …	**Java forEach method examples**	**Spring Boot Actuator metrics monitoring …**	**Java Singleton Design Pattern Example**
a year ago • 2 comments	8 months ago • 1 comment	2 years ago • 14 comments	2 years ago • 6 comments
Given an array of integers, return the indices of the two numbers whose sum is …	The Java forEach method is a utility method that can be used to iterate over a …	Learn how to monitor your spring boot application's metrics over time using …	Singleton design pattern is a creational design pattern. It is used when you want to …

3 Comments **CalliCoder** **Disqus' Privacy Policy** 1 **Login**

Recommend Tweet Share **Sort by Newest**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

CALLICODER

Stefan M • a year ago

Hi, how to delete some records by specifying a partial key , lets say by email-address? how do I qualify the field within the embeddable? thx

Reply • Share ›

Melvin Jones • a year ago

Thanks for posting this. For me, this didn't work until I added @column names for the EmployeeIdentity fields.

Reply • Share ›

Dave Whitla • 3 years ago

semrush
Descubre 3 formas de integrar PPC y S
[Con consejos de expertos]

CALLICODER