Learn Spring framework:

- Learn Spring on YouTube
- Understand the core of Spring
- Understand Spring MVC
- Understand Spring AOP
- Understand Spring Data JPA
- Spring Dependency Injection (XML)
- Spring Dependency Injection (Annotations)
- Spring Dependency Injection (Java config)
- Spring MVC beginner tutorial
- Spring MVC Exception Handling
- Spring MVC and log4j
- Spring MVC Send email
- Spring MVC File Upload
- Spring MVC Form Handling
- Spring MVC Form Validation
- Spring MVC File Download
- Spring MVC JdbcTemplate
- Spring MVC CSV view
- Spring MVC Excel View
- Spring MVC PDF View
- Spring MVC XstlView
- Spring MVC + Spring Data JPA + Hibernate - CRUD
- Spring MVC Security (XML)
- Spring MVC Security (Java config)
- Spring & Hibernate Integration (XML)
- Spring & Hibernate Integration (Java config)
- Spring & Struts Integration (XML)
- Spring & Struts Integration (Java config)
- 14 Tips for Writing Spring MVC Controller

Spring Data JPA Composite Primary Key Examples

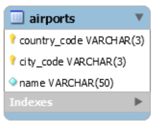
Written by Nam Ha Minh
Last Updated on 21 November 2022 | Print Email

You know, composite primary key would be used in case the combined values of two or more columns uniquely identify rows in a database table. In this Spring framework article, you will learn how to map composite primary key in database with composite ID field in Java code with Spring Data JPA and Hibernate.

NOTE: for more details about why composite primary key are used, read this article.

1. Spring Data JPA Composite Primary Key Example with @IdClass Annotation

Let's see how to use the `@IdClass` annotation for mapping a composite primary key in Spring Data JPA. Suppose that we have the following table:



This table has a composite primary key that consists of two columns *country_code* and *city_code*. So in Java code, create the following class that represents a composite ID field:

```java
package net.codejava.airport;

import java.io.Serializable;

public class AirportID implements Serializable {
    private String countryCode;
    private String cityCode;

    // getters and setters...
    // equals and hashCode...
}
```

This class is called ID class which must meet the following requirements:

- The ID class must be public

- It must implement `java.io.Serializable` interface

- It must have no-argument constructor

- It must override `equals()` and `hashCode()` methods

And then code the entity class using JPA annotations as follows:

```java
package net.codejava.airport;

import javax.persistence.*;

@Entity
@Table(name = "airports")
@IdClass(AirportID.class)
public class Airport {

    @Id @Column(length = 3)
    private String countryCode;

    @Id @Column(length = 3)
    private String cityCode;

    @Column(length = 50, nullable = false)
    private String name;

    // getters and setters...

    // equals and hash Code

}
```

You see, we use the `@IdClass` annotation to specify the name of the ID class that maps to the composite primary key in database. And in this approach, the entity class must redefine the fields in the ID class.

Next, code the repository interface as follows:

```java
package net.codejava.airport;

import org.springframework.data.repository.CrudRepository;

public interface AirportRepository extends CrudRepository<Airport, AirportID> {

}
```

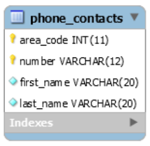
And below is a unit test class that demonstrates how to persist, list and query `Airport` objects:

```java
package net.codejava.airport;

import static org.assertj.core.api.Assertions.assertThat;

import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase.Replace;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.annotation.Rollback;

@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Rollback(false)
public class AirportRepositoryTests {

    @Autowired private AirportRepository repo;

    @Test
    public void testSaveNew() {
        Airport airport = new Airport();
        airport.setCountryCode("VN");
        airport.setCityCode("HAN");
        airport.setName("Noi Bai International Airport");

        Airport savedAirport = repo.save(airport);

        assertThat(savedAirport).isNotNull();
        assertThat(savedAirport.getCountryCode()).isEqualTo("VN");
        assertThat(savedAirport.getCityCode()).isEqualTo("HAN");
    }

    @Test
    public void testListAll() {
        Iterable<Airport> airports = repo.findAll();

        assertThat(airports).isNotEmpty();
        airports.forEach(System.out::println);
    }

    @Test
    public void testFindById() {
        AirportID id = new AirportID();
        id.setCityCode("HAN");
        id.setCountryCode("VN");

        Optional<Airport> result = repo.findById(id);
        assertThat(result).isPresent();
    }
}
```

As you can see, using `@IdClass` approach we have to redefine the fields that makeup composite ID in the entity class, which is somewhat inconvenient.

2. Spring Data JPA Composite Primary Key Example with @EmbeddedId Annotation

Next, let's see another way of mapping composite primary key using **`@Embeddable`** and **`@EmbeddedId`** annotations. Given the following table that has a composite primary key *area_code* and *number*:


phone_contacts
area_code INT(11)
number VARCHAR(12)
first_name VARCHAR(20)
last_name VARCHAR(20)
Indexes

So in Java code, you need to code the ID class as follows:

```java
package net.codejava.phone;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
public class PhoneID implements Serializable {

    @Column(name = "area_code")
    private int areaCode;

    @Column(length = 12)
    private String number;

    public PhoneID() {
    }

    public PhoneID(int areaCode, String number) {
        this.areaCode = areaCode;
        this.number = number;
    }

    // getters and setters...

    // equals() and hashCode()...

}
```

See the `@Embeddable` annotation used? And you can also use mapping annotations like `@Column` in the ID class. And code the entity class as follows:

```java
package net.codejava.phone;

import javax.persistence.*;

@Entity
@Table(name = "phone_contacts")
public class PhoneContact {

    @EmbeddedId
    private PhoneID id;

    @Column(name = "first_name", nullable = false, length = 20)
    private String firstName;

    @Column(name = "last_name", nullable = false, length = 20)
    private String lastName;

    public PhoneID getId() {
        return id;
    }

    // getters and setters...

    // equals() and hashCode()...

}
```

You see the `@EmbeddedId` annotation is used for the composite ID field? And code the repository interface as below:

```java
package net.codejava.phone;

import org.springframework.data.jpa.repository.JpaRepository;

public interface PhoneContactRepository extends JpaRepository<PhoneContact, PhoneID> {

}
```

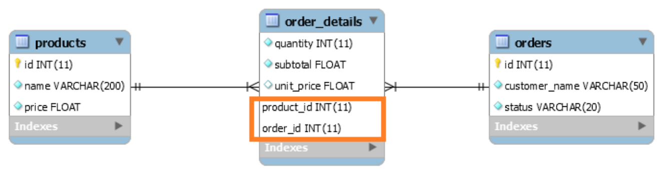
And below is an example unit test class:

```java
package net.codejava.phone;

import static org.assertj.core.api.Assertions.assertThat;

import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase.Replace;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.annotation.Rollback;

@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Rollback(false)
public class PhoneContactRepositoryTests {

    @Autowired private PhoneContactRepository repo;

    @Test
    public void testSaveNew() {
        PhoneID id = new PhoneID(1, "12027933129");
        PhoneContact newContact = new PhoneContact();
        newContact.setId(id);
        newContact.setFirstName("John");
        newContact.setLastName("Kellyson");

        PhoneContact savedContact = repo.save(newContact);

        assertThat(savedContact).isNotNull();
        assertThat(savedContact.getId().getAreaCode()).isEqualTo(1);
        assertThat(savedContact.getId().getNumber()).isEqualTo("12027933129");
    }

    @Test
    public void testListAll() {
        List<PhoneContact> contacts = repo.findAll();

        assertThat(contacts).isNotEmpty();
        contacts.forEach(System.out::println);
    }

    @Test
    public void testFindById() {
        PhoneID id = new PhoneID(1, "12027933129");
        Optional<PhoneContact> result = repo.findById(id);

        assertThat(result).isPresent();
    }
}
```

You see, this class tests for save, list and query against entity objects that have composite ID field. And in this approach, the ID class and entity class do not have duplicate fields.

3. Spring Data JPA Composite Primary Key Example with Foreign Key Reference

And how about mapping a composite primary key that consists of foreign keys? Let's see the code example below. Given the following 3 tables that form a many-to-many relationship:



Here, the *order_details* table realizes many to many association between *products* and *orders* tables. It has the composite primary key that consists of two foreign keys. So in Java code, we have the `Product` entity class as below:

```java
package net.codejava.sale;

import javax.persistence.*;

@Entity
@Table(name = "products")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false, length = 200)
    private String name;

    private float price;

    // getters and setters

    // equals() and hashCode()
}
```

Code of the `Order` entity class:

```java
package net.codejava.sale;

import javax.persistence.*;

@Entity
@Table(name = "orders")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "customer_name", length = 50, nullable = false)
    private String customerName;

    @Column(length = 20, nullable = false)
    private String status;

}
```

And code of the ID class that maps to the composite primary key in the database table:

```java
package net.codejava.sale;

import java.io.Serializable;

import javax.persistence.*;

@Embeddable
public class OrderDetailID implements Serializable {

    @ManyToOne
    @JoinColumn(name = "order_id")
    private Order order;

    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;

    // getters and setters...

    // equals() and hashCode()...
}
```

As you can see, for foreign key mapping we use the `@ManyToOne` and `@JoinColumn` annotations as usual in the ID class. And below is code of the entity class that makes use of this composite ID:

```java
package net.codejava.sale;

import javax.persistence.*;

@Entity
@Table(name = "order_details")
public class OrderDetail {

    @EmbeddedId
    private OrderDetailID id;

    private int quantity;

    @Column(name = "unit_price")
    private float unitPrice;

    private float subtotal;

    // getters and setters...

    // equals() and hashCode()...
}
```

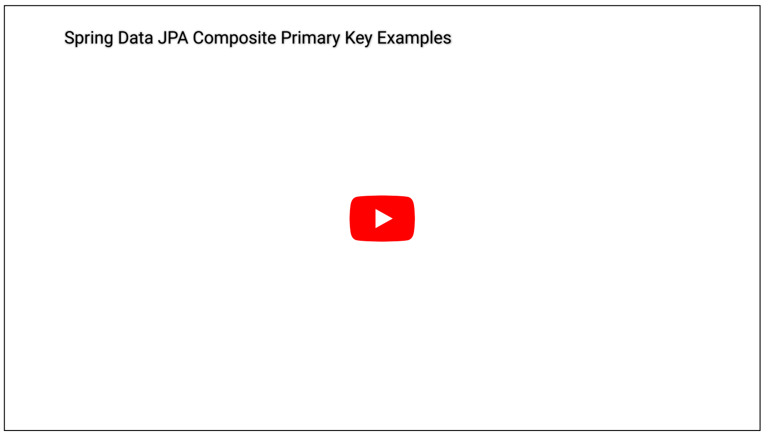
For Spring Data JPA, code the repository interface as follows:

```java
package net.codejava.sale;

import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderDetailRepository extends JpaRepository<OrderDetail, OrderDetailID> {

}
```

And for your reference, below is code example of a unit test class that demonstrates how to persist and list objects:

```java
package net.codejava.sale;

import static org.assertj.core.api.Assertions.assertThat;

import java.util.List;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase.Replace;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.annotation.Rollback;

@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Rollback(false)
public class OrderDetailRepositoryTests {

    @Autowired private OrderDetailRepository orderDetailRepo;
    @Autowired private ProductRepository productRepo;
    @Autowired private OrderRepository orderRepo;

    @Test
    public void testListAll() {
        List<OrderDetail> orderDetails = orderDetailRepo.findAll();

        assertThat(orderDetails).isNotEmpty();
        orderDetails.forEach(System.out::println);
    }

    @Test
    public void testAddOrderDetail() {
        Product product = new Product();
        product.setName("iPhone 15");
        product.setPrice(1199);

        productRepo.save(product);

        Order order = new Order();
        order.setCustomerName("Nam Ha Minh");
        order.setStatus("In Progress");

        orderRepo.save(order);

        OrderDetail orderDetail = new OrderDetail();

        OrderDetailID orderDetailId = new OrderDetailID();
        orderDetailId.setOrder(order);
        orderDetailId.setProduct(product);

        orderDetail.setId(orderDetailId);
        orderDetail.setQuantity(2);
        orderDetail.setUnitPrice(1199);
        orderDetail.setSubtotal(2398);

        OrderDetail savedOrderDetail = orderDetailRepo.save(orderDetail);

        assertThat(savedOrderDetail).isNotNull();
        assertThat(savedOrderDetail.getId().getProduct()).isEqualTo(product);
        assertThat(savedOrderDetail.getId().getOrder()).isEqualTo(order);
    }
}
```

That's my guide about mapping composite primary key in Spring Data JPA. I hope you found the code examples helpful. To see the coding in action, watch the following video:



Related Spring and Database Tutorials:

- Spring Data JPA EntityManager Examples (CRUD Operations)
- JPA EntityManager: Understand Differences between Persist and Merge
- Understand Spring Data JPA with Simple Example
- Spring Data JPA Custom Repository Example
- Spring MVC with JdbcTemplate Example
- How to configure Spring MVC JdbcTemplate with JNDI Data Source in Tomcat
- Spring and Hibernate Integration Tutorial (XML Configuration)
- Spring MVC + Spring Data JPA + Hibernate - CRUD Example

About the Author:

Nam Ha Minh is certified Java programmer (SCJP and SCWCD). He started programming with Java in the time of Java 1.4 and has been falling in love with Java since then. Make friend with him on Facebook and watch his Java videos you YouTube.

Attachments:
SpringDataCompositeKey.zip [Sample Spring Boot project] 98 kB

Add comment

Name E-mail

comment

☐ Notify me of follow-up comments

☐ Je ne suis pas un robot
reCAPTCHA
Confidentialité - Conditions

Send

Comments

#1**Tyler** 2023-11-16 18:02
Can you give an example of a relationship between two entities that both have composite keys?
Quote

Refresh comments list