ABOUT SHAGUN MITTAL

Shagun is an enthusiastic Java developer with 5 + years of Industry experience in various Java and Web technologies. She is a certified Java Programmer and has worked extensively with other programming languages and framework, such as Hibernate, Spring MVC, Spring Boot, RDBMS, Unix, etc. As part of delivery services she has created and implemented efficient applications and programs in various industries, including insurance, retail, public sector, telecommunication etc.

Hibernate Foreign Key Example

Posted by: Shagun Mittal in hibernate August 20th, 2018 1 Comment 7551 Views

1. Introduction

In this post, we feature a comprehensive Example on Hibernate Foreign Key. Foreign key refers to single column or group of columns in table that link data present in another table through its primary key. A Foreign key can't exist without its parent key but viceversa is not true.

Example – A Menu can have submenus. It can be represented in tabular form as shown below where column

```
MENU_ID
```

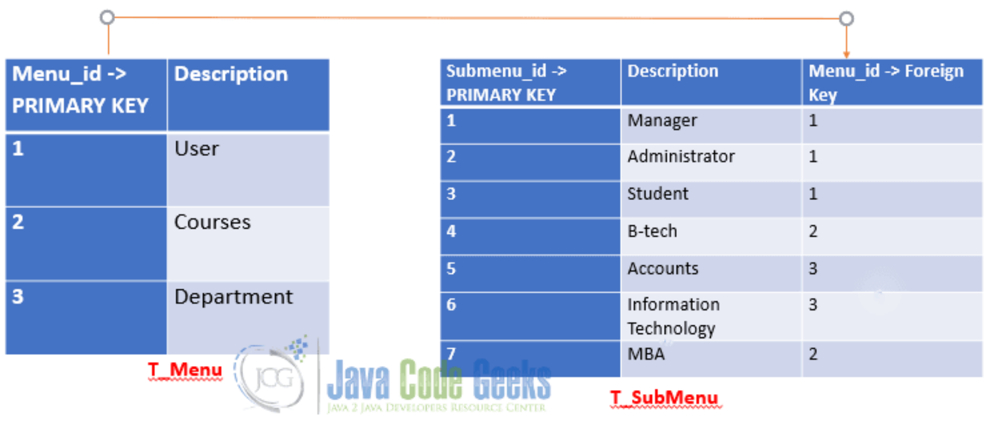
is Primary key of

```
T_MENU
```

table and it is acting as Foreign Key (link between both tables) for

```
T_SUBMENU
```

table:



Menu_id -> PRIMARY KEY	Description
1	User
2	Courses
3	Department

T_Menu

Submenu_id -> PRIMARY KEY	Description	Menu_id -> Foreign Key
1	Manager	1
2	Administrator	1
3	Student	1
4	B-tech	2
5	Accounts	3
6	Information Technology	3
7	MBA	2

T_SubMenu

Java Persistance Specifications provide different ways to create Foreign Key mappings as mentioned below:

1 – Using Association Mappings
2 – By Saving Collections using

```
@ElementCollection
```

In this article we will show Foreign Key Creation using **One to Many bi-directional Association Mapping**.

Association Mapping – It is a feature provided by JPA to link two tables using below associations. Each Association can be Uni-Directional or Bi-Directional.

Association	Example
One to One	One Person can have One Unique Identification Number
One to Many	One Menu can have Many Sub-Menu
Many to One	Many Sub-Menu can have One Parent Menu (Reverse of Many to One)
Many to Many	One Student can enrol for many courses and a course can be enrolled by many students.

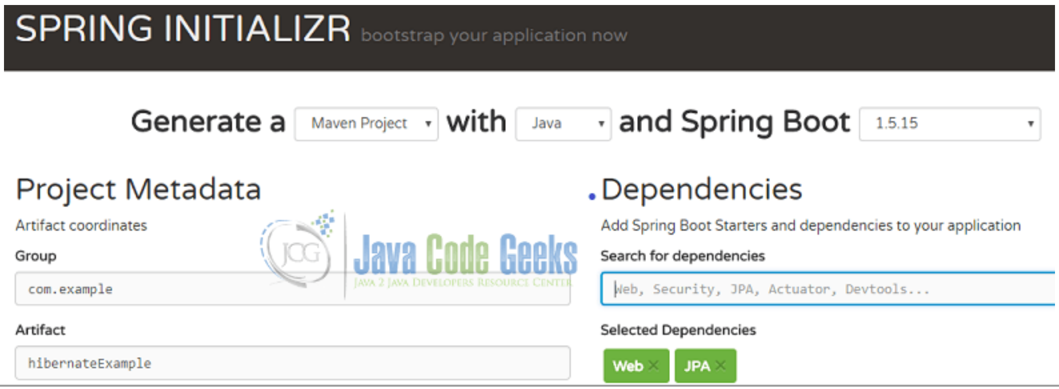
2. Technologies Used

We will be building this project from scratch using following tools and technologies:

- Eclipse
- Spring Boot 1.5.10
- Maven
- Oracle
- Hibernate
- Java 8 or above

3. Create Project

We are creating Spring Boot project using Spring initializer. Steps are mentioned below:
1 – Go to http://start.spring.io/
2 – Select the following:



SPRING INITIALIZR bootstrap your application now
Project Metadata
Artifact coordinates
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies

Generate a Maven Project with Java and Spring Boot 1.5.15

Group: com.example

Artifact: hibernateExample

Web ✕ JPA ✕

✖

This spring project is ready to deploy and you can run it as Java Application in Eclipse. Now we will build our One To Many Mapping Example. For Simplicity, we'll be creating Service, Repository and Model classes in same package –

```
com.example.hibernateExample
```

.

3.1 Project Configurations

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>hibernateExample</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>hibernateExample</name>

<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.5.16.BUILD-SNAPSHOT</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<java.version>1.8</java.version>
</properties>

<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
<groupId>javax.xml.bind</groupId>
<artifactId>jaxb-api</artifactId>
<version>2.3.0</version>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>
</project>
```

Dependencies used in pom.xml: Spring Boot MVC(

```
spring-boot-starter-web
```

), Hibernate (

```
spring-boot-starter-data-jpa
```

) and

```
jaxb-api
```

.

✖

```
# logging
logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n
logging.level.org.hibernate.SQL=debug
```

```
application.properties
```

file is present in

```
src/main/resources
```

folder of a Spring Boot project. We are doing Hibernate Configurations here using Oracle JDBC driver (Since Oracle restricts automatic download of OJDBC dependency by Maven, one need to explicitly download

```
ojdbc6.jar/ojdbc7.jar
```

from Oracle's site and need to include it in

```
ClassPath
```

)

3.2 Model Classes – MainMenu and SubMenu

In this section, we will design our model or entity classes using JPA and Hibernate provided annotations. Hibernate framework will be using these annotations to create tables and their Foreign Key Relationship in database. Variables of Entity class will be created as **Columns** in database table.

MainMenu.java

```
01  package com.example.hibernateExample;
02  import java.io.Serializable;
03  import java.util.HashSet;
04  import java.util.List;
05  import java.util.Set;
06  import javax.persistence.CascadeType;
07  import javax.persistence.Column;
08  import javax.persistence.Entity;
09  import javax.persistence.FetchType;
10  import javax.persistence.GeneratedValue;
11  import javax.persistence.GenerationType;
12  import javax.persistence.Id;
13  import javax.persistence.OneToMany;
14  import javax.persistence.Table;
15
16  @Entity
17  @Table(name = "T_Menu")
18  public class MainMenu implements Serializable{
19
20      @Id
21      @GeneratedValue(strategy=GenerationType.AUTO)
22      private int id;
23
24      private String description;
25
26      @OneToMany(mappedBy="mainMenu", cascade = CascadeType.ALL)
27      Set subMenu = new HashSet();
28
29      public MainMenu() {
30      }
31
32      public MainMenu(String description) {
33          this.description = description;
34      }
35
36  // Getters and Setters (Omitted for brevity)
```

```
MainMenu
```

class is One(Reference) side of relationship and

```
SubMenu
```

class represents Many(owning) side of relationship as 'One Menu can have many Sub Menu'. In Database terminology, the table that has foreign key is Owner of association mapping. Let's understand few annotations in detail which are used by Hibernate framework to create and manage Entity classes.
Line 16:

```
@Entity
```

will be same as entity class name.
Line 20:

```
@Id
```

specify the variable as Primary key column for database table.
Line 21:

```
@GeneratedValue
```

specify the Generation strategy for Primary Key.
Line 26:

```
mappedBy
```

is used with

```
@OnetoMany
```

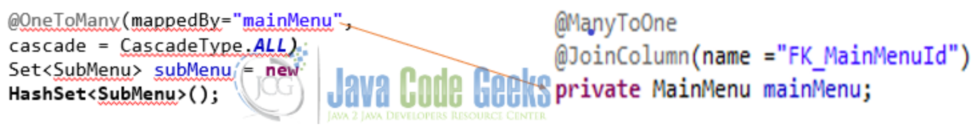
side of association. It indicates that the entity in this side is the **inverse** of the relationship, and the owner resides in the "other" entity. It is used to make a relationship Bi-directional, that means the SubMenu class can be persisted or fetched through Menu class as well.

```
mainMenu
```

in

```
mappedBy="mainMenu"
```

is the ManyToOne annotated field/variable of SubMenu class as shown below:



```
@OneToMany(mappedBy="mainMenu",
cascade = CascadeType.ALL)
Set<SubMenu> subMenu = new
HashSet<SubMenu>();
```

```
@ManyToOne
@JoinColumn(name ="FK_MainMenuId")
private MainMenu mainMenu;
```

Association Mapping

```
CascadeType.ALL
```

will perform all **EntityManager** operations (

```
PERSIST, REMOVE, REFRESH, MERGE, DETACH
```

) to the related entities/ collection e.g when Menu will be Persisted, SubMenu will also be Persisted.

SubMenu.java

```
package com.example.hibernateExample;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "T_SubMenu")
public class SubMenu implements Serializable{

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;

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


    @ManyToOne
    @JoinColumn(name ="FK_MainMenuId")
    private MainMenu mainMenu;

    public SubMenu() {
```

will be used by Hibernate to create

```
T_Submenu
```

table in database.

```
@JoinColumn
```

annotation in line 27 indicates that this entity is the ***owner*** of the relationship (which will contain Foreign Key in Database perspective). This annotation is always used with

```
@ManyToOne
```

side of association.

```
name
```

attribute is used to give logical name to Foreign Key column, though it is not mandatory.

3.3 Repository Interface

MainMenuRepository.java

```java
package com.example.hibernateExample;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface MainMenuRepository extends CrudRepository<MainMenu, Integer>{

}
```

In this section we are creating

```
MainMenuRepository
```

interface that is a Marker interface(which doesn't define any methods). When using Spring Data we need to define a *Repository* interface corresponding to each domain Entity. It will be extending Spring Data's

```
CrudRepository
```

interface which declares standard CRUD operations that can be performed on an entity. Use of

```
CrudRepository
```

interface will prevent us from writing a lot of boilerplate code to access data source, writing SQL queries, Result Set etc. It will accept two parameters:
1 – Entity class corresponding to the Marker interface.
2 – Data type of Primary key defined within Entity class.

3.4 Runner

HibernateExampleApplication.java

```java
package com.example.hibernateExample;

import java.util.List;
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 HibernateExampleApplication implements CommandLineRunner
{
     @Autowired
       MenuService menuService;

    public static void main( String[] args )
    {
        SpringApplication.run(App.class, args);
    }

        @Override
    public void run(String... args) throws Exception {
        menuService.addMenu();
```

`@SpringBootApplication`

that is equivalent of using

`@Configuration`

,

`@EnableAutoConfiguration`

, and

`@ComponentScan`

. We will be adding new Menus and subMenus in

`addMenu()`

of service class, which is invoked in overrided

`run()`

of

`CommandLineRunner`

interface.

3.5 Service Layer

In this section we will be creating new Menus and their Sub-Menus using methods provided by Spring Data's

`CrudRepository`

interface. The newly created Menus and their associated Sub-Menus will be added as rows in

`T_menu`

and

`T_submenu`

table by Hibernate framework.

MenuService.java

```java
package com.example.hibernateExample;

public interface MenuService {
    public void addMenu();
}
```

MenuServiceImpl.java

```java
package com.example.hibernateExample;

import java.util.HashSet;
import java.util.Set;
import javax.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MenuServiceImpl implements MenuService{

    @Autowired
    MainMenuRepository mainMenuRepository;

    @Transactional
    public void addMenu(){
        // For User MainMenu
         MainMenu menu1 = new MainMenu("User");
         //Creating sub-menus for user
         Set subMenu1 = new HashSet();
         subMenu1.add(new SubMenu("Manager", menu1));
         subMenu1.add(new SubMenu("Administrator", menu1));
         subMenu1.add(new SubMenu("Student", menu1));
         menu1.setSubMenu(subMenu1);

```

```
                subMenu3.add(new SubMenu("Accounts", menu3));
                subMenu3.add(new SubMenu("Information Technology", menu3));
                subMenu3.add(new SubMenu("Sports", menu3));
                menu3.setSubMenu(subMenu3);

        //Save MainMenu
          Set mainMenu = new HashSet();
          mainMenu.add(menu1);
          mainMenu.add(menu2);
          mainMenu.add(menu3);
        mainMenuRepository.save(mainMenu);

    }
}
```

`addMenu()`

of

`MenuServiceImpl`

class is adding 3 MainMenu named as Course, Department and User and their submenus using CrudRepository's

`save()`

.
On Executing this project as a Java Application in Eclipse, we will get following output where

`FK_MAIN_MENU_ID`

is foreign key in

`T_submenu`

table:

ID	DESCRIPTION
1	Department
5	Course
9	User

ID	SUBMENU_DESC	FK_MAIN_MENU_ID
2	Sports	1
3	Information Technology	1
4	Accounts	1
6	B-Tech	5
7	BCA	5
8	MBA	5
10	Manager	9
11	Student	9
12	Administrator	9

4. Summary

To Summarize, we have created a Spring Boot project that is adding 3 mainMenu in

`T_menu`

table i.e Course, Department and User. Each mainMenu can have multiple submenu which are stored in

`T_submenu`

✖

Entity classes.

5. Download the Source Code

This was an example of creating a Hibernate Foreign Key.

Download
You can download the full source code of this example here: **hibernateExample.zip**

Tagged with: CORE JAVA HIBERNATE SPRING SPRING BOOT SPRING DATA

(**0** rating, **2** votes) 1 Comment 7551 Views Tweet it!

1 Leave a Reply

Join the discussion...

1 0 0 1

Subscribe ▾ ▴ newest ▴ oldest ▴ most voted

Michael Murphy

Guest

I have no idea if you guys monitor these comments but I just wanted to say you've saved me with this tutorial, I was having trouble finding something to explain this simple functionality! This was very clear, and very easy to extend to my own project!

✖

✖

- Courses
- Minibooks
- News
- Resources
- Tutorials

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- System Code Geeks
- Web Code Geeks

- Android Alert Dialog Example
- Android OnClickListener Example
- How to convert Character to String and a String to Character Array in Java
- Java Inheritance example
- Java write to File Example
- java.io.FileNotFoundException – How to solve File Not Found Exception
- java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception
- java.lang.NoClassDefFoundError – How to solve No Class Def Found Error
- JSON Example With Jersey + Jackson
- Spring JdbcTemplate Example

JCGs (Java Code Geeks) is an independent online community focused on crea
ultimate Java to Java developers resource center; targeted at the technical ar
technical team lead (senior developer), project manager and junior developer
JCGs serve the Java, SOA, Agile and Telecom communities with daily news w
domain experts, articles, tutorials, reviews, announcements, code snippets ar
source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geeks are
property of their respective owners. Java is a trademark or registered tradem
Oracle Corporation in the United States and other countries. Examples Java C
is not connected to Oracle Corporation and is not sponsored by Oracle Corpor

✖