<https://www.devglan.com/hibernate/hibernate-one-to-many-relationship-example>

# Hibernate one to many mapping annotation example

**Hibernate one to many mapping** is made between two entities where first entity can have relation with multiple second entity instances but second can be associated with only one instance of first entity. Its **1 to N** relationship.

For example, in any company an employee can register multiple bank accounts but one bank account will be associated with one and only one employee. In this [hibernate](https://howtodoinjava.com/hibernate-tutorials/) one to many mapping annotation example, we will learn to make such mapping in database using hibernate.

Table of Contents

When to use one to many mapping

Hibernate one to many mapping solutions

1. Hibernate one to many mapping with foreign key association

2. Hibernate one to many mapping with join table

## When to use one to many mapping

Use one to mapping to create **1..N relationship** between entities or objects.

For example, we have to write two entities i.e. EmployeeEntity and AccountEntity such that multiple accounts can be associated with a single employee, but one single account can not be shared between two or more employees.

## Hibernate one to many mapping solutions

This problem can be solved in two different ways.

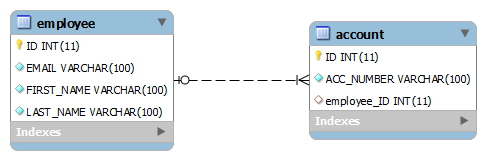
1. One is to have a **foreign key column** in account table i.e. EMPLOYEE\_ID. This column will refer to primary key of Employee table. This way no two accounts can be associated with multiple employees. Obviously, account number needs to be unique for enforcing this restriction.
2. Second approach is to have a common **join table** lets say EMPLOYEE\_ACCOUNT. This table will have two column i.e. EMP\_ID which will be foreign key referring to primary key in EMPLOYEE table and similarly ACCOUNT\_ID which will be foreign key referring to primary key of ACCOUNT table.

## 1. Hibernate one to many mapping with foreign key association

In this approach, **both entity will be responsible for making the relationship** and maintaining it. EmployeeEntity should declare that relationship is one to many, and AccountEntity should declare that relationship from its end is many to one.

#### 1.1. Design one to many mapping relationship

Lets first see the schema design.



#### 1.2. Entity classes

Write entity classes.

|  |
| --- |
| EmployeeEntity.java |
| package hibernate.test.oneToMany.foreignKeyAsso;    import java.io.Serializable;  import java.util.Set;    import javax.persistence.CascadeType;  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.OneToMany;  import javax.persistence.Table;  import javax.persistence.UniqueConstraint;    @Entity(name = "ForeignKeyAssoEntity")  @Table(name = "Employee", uniqueConstraints = {  @UniqueConstraint(columnNames = "ID"),  @UniqueConstraint(columnNames = "EMAIL") })  public class EmployeeEntity implements Serializable {        private static final long serialVersionUID = -1798070786993154676L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer employeeId;        @Column(name = "EMAIL", unique = true, nullable = false, length = 100)      private String email;        @Column(name = "FIRST\_NAME", unique = false, nullable = false, length = 100)      private String firstName;        @Column(name = "LAST\_NAME", unique = false, nullable = false, length = 100)      private String lastName;        @OneToMany(cascade=CascadeType.ALL)      @JoinColumn(name="EMPLOYEE\_ID")      private Set<AccountEntity> accounts;        //Getters and setters  } |

Write **AccountEntity.java**.

|  |
| --- |
| AccountEntity.java |
| package hibernate.test.oneToMany.foreignKeyAsso;    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.ManyToOne;  import javax.persistence.Table;  import javax.persistence.UniqueConstraint;    @Entity(name = "ForeignKeyAssoAccountEntity")  @Table(name = "ACCOUNT", uniqueConstraints = {  @UniqueConstraint(columnNames = "ID")})  public class AccountEntity implements Serializable  {      private static final long serialVersionUID = -6790693372846798580L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer accountId;        @Column(name = "ACC\_NUMBER", unique = true, nullable = false, length = 100)      private String accountNumber;        @ManyToOne      private EmployeeEntity employee;        //Getters and setters  } |

#### 1.3. Demo

|  |
| --- |
| package hibernate.test.oneToMany;    import hibernate.test.HibernateUtil;  import hibernate.test.oneToMany.foreignKeyAsso.AccountEntity;  import hibernate.test.oneToMany.foreignKeyAsso.EmployeeEntity;    import java.util.HashSet;  import java.util.Set;    import org.hibernate.Session;    public class TestForeignKeyAssociation  {        public static void main(String[] args)      {          Session session = HibernateUtil.getSessionFactory().openSession();          session.beginTransaction();            AccountEntity account1 = new AccountEntity();          account1.setAccountNumber("Account detail 1");            AccountEntity account2 = new AccountEntity();          account2.setAccountNumber("Account detail 2");            AccountEntity account3 = new AccountEntity();          account3.setAccountNumber("Account detail 3");            //Add new Employee object          EmployeeEntity firstEmployee = new EmployeeEntity();          firstEmployee.setEmail("demo-user-first@mail.com");          firstEmployee.setFirstName("demo-one");          firstEmployee.setLastName("user-one");            EmployeeEntity secondEmployee = new EmployeeEntity();          secondEmployee.setEmail("demo-user-second@mail.com");          secondEmployee.setFirstName("demo-two");          secondEmployee.setLastName("user-two");            Set<AccountEntity> accountsOfFirstEmployee = new HashSet<AccountEntity>();          accountsOfFirstEmployee.add(account1);          accountsOfFirstEmployee.add(account2);            Set<AccountEntity> accountsOfSecondEmployee = new HashSet<AccountEntity>();          accountsOfSecondEmployee.add(account3);            firstEmployee.setAccounts(accountsOfFirstEmployee);          secondEmployee.setAccounts(accountsOfSecondEmployee);          //Save Employee          session.save(firstEmployee);          session.save(secondEmployee);            session.getTransaction().commit();          HibernateUtil.shutdown();      }  } |

Program Output:

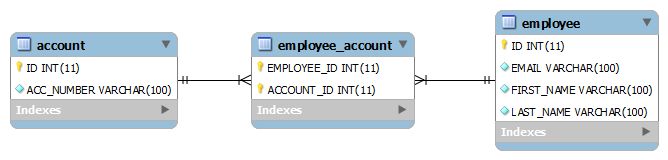
|  |
| --- |
| Console |
| Hibernate: insert into Employee (EMAIL, FIRST\_NAME, LAST\_NAME) values (?, ?, ?)  Hibernate: insert into ACCOUNT (ACC\_NUMBER, employee\_ID) values (?, ?)  Hibernate: insert into ACCOUNT (ACC\_NUMBER, employee\_ID) values (?, ?)  Hibernate: insert into Employee (EMAIL, FIRST\_NAME, LAST\_NAME) values (?, ?, ?)  Hibernate: insert into ACCOUNT (ACC\_NUMBER, employee\_ID) values (?, ?)  Hibernate: update ACCOUNT set EMPLOYEE\_ID=? where ID=?  Hibernate: update ACCOUNT set EMPLOYEE\_ID=? where ID=?  Hibernate: update ACCOUNT set EMPLOYEE\_ID=? where ID=? |

## 2. Hibernate one to many mapping with join table

This approach uses a **join table** to store the associations between account and employee entities. **@JoinTabl**e annotation has been used to make this association.

#### 2.1. Design

Lets see how the database schema will look like:

[](https://howtodoinjava.files.wordpress.com/2012/11/one-to-many-association-in-hiberate-using-join-table.png)one To Many association in hibernate using join table

#### 2.2. Entity classes

|  |  |
| --- | --- |
| EmployeeEntity.java | |
| package hibernate.test.oneToMany.joinTable;    import java.io.Serializable;  import java.util.Set;    import javax.persistence.CascadeType;  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.JoinTable;  import javax.persistence.OneToMany;  import javax.persistence.Table;  import javax.persistence.UniqueConstraint;    @Entity(name = "JoinTableEmployeeEntity")  @Table(name = "Employee", uniqueConstraints = {  @UniqueConstraint(columnNames = "ID"),  @UniqueConstraint(columnNames = "EMAIL") })  public class EmployeeEntity implements Serializable  {      private static final long serialVersionUID = -1798070786993154676L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer employeeId;        @Column(name = "EMAIL", unique = true, nullable = false, length = 100)      private String email;        @Column(name = "FIRST\_NAME", unique = false, nullable = false, length = 100)      private String firstName;        @Column(name = "LAST\_NAME", unique = false, nullable = false, length = 100)      private String lastName;        @OneToMany(cascade=CascadeType.ALL)      @JoinTable(name="EMPLOYEE\_ACCOUNT", joinColumns={@JoinColumn(name="EMPLOYEE\_ID", referencedColumnName="ID")}      , inverseJoinColumns={@JoinColumn(name="ACCOUNT\_ID", referencedColumnName="ID")})      private Set<AccountEntity> accounts;        //Getters and setters  } | |
| AccountEntity.java |
| package hibernate.test.oneToMany.joinTable;    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.Table;  import javax.persistence.UniqueConstraint;    @Entity(name = "JoinTableAccountEntity")  @Table(name = "ACCOUNT", uniqueConstraints = {  @UniqueConstraint(columnNames = "ID")})  public class AccountEntity implements Serializable  {        private static final long serialVersionUID = -6790693372846798580L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer accountId;        @Column(name = "ACC\_NUMBER", unique = true, nullable = false, length = 100)      private String accountNumber;        //Getters and setters  } |

#### 2.3. Hibernate configuration

**We have available both entities to runtime, we have to add them in hibernate.cfg.xml file. Please note that only one set of entities should be configured in configuration file otherwise unexpected results can occur.**

|  |
| --- |
| hibernate.cfg.xml |
| < ?xml version="1.0" encoding="utf-8"?>  < !DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd>">  <hibernate-configuration>      <session-factory>          <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>          <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatetest</property>          <property name="hibernate.connection.password">XXXXXX</property>          <property name="hibernate.connection.username">root</property>          <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>          <property name="show\_sql">true</property>          <property name="hbm2ddl.auto">create</property>          <mapping clas="hibernate.test.oneToMany.foreignKeyAsso.AccountEntity"></mapping>          <mapping clas="hibernate.test.oneToMany.foreignKeyAsso.EmployeeEntity"></mapping>      </session-factory>  </hibernate-configuration> |

#### 2.4. Demo

Now, its time to test the code. I have written following code to test above entities.

|  |
| --- |
| TestJoinTable.java |
| package hibernate.test.oneToMany;    import hibernate.test.HibernateUtil;  import hibernate.test.oneToMany.joinTable.AccountEntity;  import hibernate.test.oneToMany.joinTable.EmployeeEntity;    import java.util.HashSet;  import java.util.Set;    import org.hibernate.Session;    public class TestJoinTable  {      public static void main(String[] args)      {          Session session = HibernateUtil.getSessionFactory().openSession();          session.beginTransaction();            AccountEntity account1 = new AccountEntity();          account1.setAccountNumber("123-345-65454");            AccountEntity account2 = new AccountEntity();          account2.setAccountNumber("123-345-6542222");            //Add new Employee object          EmployeeEntity emp = new EmployeeEntity();          emp.setEmail("demo-user@mail.com");          emp.setFirstName("demo");          emp.setLastName("user");            Set<AccountEntity> accounts = new HashSet<AccountEntity>();          accounts.add(account1);          accounts.add(account2);            emp.setAccounts(accounts);          //Save Employee          session.save(emp);            session.getTransaction().commit();          HibernateUtil.shutdown();      }  } |

Program Output:

|  |
| --- |
| Console |
| Hibernate: insert into Employee (EMAIL, FIRST\_NAME, LAST\_NAME) values (?, ?, ?)  Hibernate: insert into ACCOUNT (ACC\_NUMBER) values (?)  Hibernate: insert into ACCOUNT (ACC\_NUMBER) values (?)  Hibernate: insert into EMPLOYEE\_ACCOUNT (EMPLOYEE\_ID, ACCOUNT\_ID) values (?, ?)  Hibernate: insert into EMPLOYEE\_ACCOUNT (EMPLOYEE\_ID, ACCOUNT\_ID) values (?, ?) |

**Hibernate One To Many Mapping Example Using Annotation**

### Creating table

Create **AUTHOR** and **BOOK** Tables, simply Copy and Paste the following SQL query in the query editor to get the table created.

CREATE TABLE "AUTHOR"

(

"AUTHOR\_ID" NUMBER(10,0) NOT NULL ENABLE,

"AUTHOR\_NAME" VARCHAR2(40 BYTE) NOT NULL ENABLE,

PRIMARY KEY (AUTHOR\_ID)

);

CREATE TABLE "BOOK"

( "BOOK\_ID" NUMBER(10,0) NOT NULL ENABLE,

"AUTHOR\_ID" NUMBER(10,0) NOT NULL ENABLE,

"BOOK\_TITLE" VARCHAR2(255 CHAR),

"BOOK\_DESCRIPTION" VARCHAR2(255 CHAR),

PRIMARY KEY ("BOOK\_ID"),

CONSTRAINT fk\_book FOREIGN KEY("AUTHOR\_ID") REFERENCES AUTHOR("AUTHOR\_ID")

);

### ****Author.java****

Create a new Java file **Author.java** under the package **com.javainterviewpoint** and add the following code

package com.javainterviewpoint;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

@Entity

@Table(name="AUTHOR")

public class Author

{

@Id

@GeneratedValue

@Column(name="AUTHOR\_ID")

private int authorId;

@Column(name="AUTHOR\_NAME")

private String authorName;

@OneToMany(mappedBy="author",cascade = CascadeType.ALL)

private Set<Book> books;

public Author()

{

super();

}

public Author(int authorId, String authorName, Set books)

{

super();

this.authorId = authorId;

this.authorName = authorName;

this.books = books;

}

public int getAuthorId()

{

return authorId;

}

public void setAuthorId(int authorId)

{

this.authorId = authorId;

}

public String getAuthorName()

{

return authorName;

}

public void setAuthorName(String authorName)

{

this.authorName = authorName;

}

public Set<Book> getBooks()

{

return books;

}

public void setBooks(Set<Book> books)

{

this.books = books;

}

@Override

public String toString()

{

return "Author [authorId=" + authorId + ", authorName=" + authorName + ", books=" + books + "]";

}

}

# Hibernate One To Many Mapping Example Using Annotation

December 27, 2016 by **[javainterviewpoint](https://www.javainterviewpoint.com/author/javainterviewpoint/)** [**Leave a Comment**](https://www.javainterviewpoint.com/hibernate-one-to-many-mapping-example-annotation/#respond)

Previously we have learned about [**Hibernate One To One Mapping Using annotation**](https://www.javainterviewpoint.com/hibernate-one-to-one-bidirectional-mapping-foreign-key/)**.**In this [**Hibernate One To Many mapping Example**](https://www.javainterviewpoint.com/hibernate-one-many-mapping-example-xml-mapping/), we will learn about **One To Many** mapping between Java objects and database tables using [**Hibernate framework**](https://www.javainterviewpoint.com/category/hibernate/) (Annotation Mapping).

### Creating table

Create **AUTHOR** and **BOOK** Tables, simply Copy and Paste the following SQL query in the query editor to get the table created.

CREATE TABLE "AUTHOR"

(

"AUTHOR\_ID" NUMBER(10,0) NOT NULL ENABLE,

"AUTHOR\_NAME" VARCHAR2(40 BYTE) NOT NULL ENABLE,

PRIMARY KEY (AUTHOR\_ID)

);

CREATE TABLE "BOOK"

( "BOOK\_ID" NUMBER(10,0) NOT NULL ENABLE,

"AUTHOR\_ID" NUMBER(10,0) NOT NULL ENABLE,

"BOOK\_TITLE" VARCHAR2(255 CHAR),

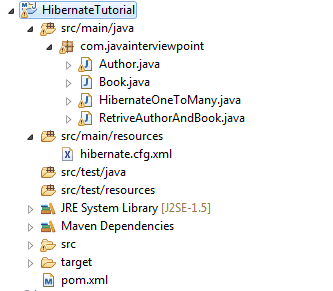
"BOOK\_DESCRIPTION" VARCHAR2(255 CHAR),

PRIMARY KEY ("BOOK\_ID"),

CONSTRAINT fk\_book FOREIGN KEY("AUTHOR\_ID") REFERENCES AUTHOR("AUTHOR\_ID")

);

### ****Folder Structure:****

**[](https://javainterviewpoint-7ac9.kxcdn.com/wp-content/uploads/2016/12/Hibernate-One-To-Many-Mapping-Example.png)**

1. Create a simple Maven*Project* **“HibernateTutorial”** and create a package for our source files **“com.javainterviewpoint”**under**src/main/java**
2. Now add the following dependency in the **POM.xml**
3. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6. <groupId>HibernateTutorial</groupId>
7. <artifactId>HibernateTutorial</artifactId>
8. <version>0.0.1-SNAPSHOT</version>
9. <properties>
10. <hibernate.version>4.3.11.Final</hibernate.version>
11. <oracle.connector.version>11.2.0</oracle.connector.version>
12. </properties>
13. <dependencies>
14. <!-- Hibernate -->
15. <dependency>
16. <groupId>org.hibernate</groupId>
17. <artifactId>hibernate-core</artifactId>
18. <version>${hibernate.version}</version>
19. </dependency>
20. <!-- Oracle -->
21. <dependency>
22. <groupId>com.oracle</groupId>
23. <artifactId>ojdbc14</artifactId>
24. <version>${oracle.connector.version}</version>
25. </dependency>
26. </dependencies>
27. <build>
28. <sourceDirectory>src</sourceDirectory>
29. <plugins>
30. <plugin>
31. <artifactId>maven-compiler-plugin</artifactId>
32. <version>3.3</version>
33. <configuration>
34. <source>1.7</source>
35. <target>1.7</target>
36. </configuration>
37. </plugin>
38. </plugins>
39. </build>

</project>

1. Create the Java classes **Author*.j*ava, Book.java, HibernateOneToMany*.java*** and **RetriveAuthorAndBook.java**under  ***com.javainterviewpoint*** folder.
2. Place the ***hibernate.cfg.xml***under the**src/main/resources** directory

#### ****Other interesting articles which you may like …****

* [**Hibernate Hello World Example in Eclipse (XML Mapping)**](https://www.javainterviewpoint.com/hibernate-hello-world-example-xml-mapping/)
* [**Hibernate Hello World Example in Eclipse (Annotation)**](https://www.javainterviewpoint.com/hibernate-hello-world-example-annotation/)
* [**Hibernate One To One Bidirectional Mapping XML Example with Primary Key**](https://www.javainterviewpoint.com/hibernate-one-one-mapping-xml/)
* [**Hibernate One To One Mapping XML Example with Foreign Key**](https://www.javainterviewpoint.com/hibernate-one-one-foreign-key-xml/)
* [**Hibernate Many To Many Mapping Example – XML Mapping**](https://www.javainterviewpoint.com/hibernate-many-to-many-mapping-xml/)
* [**Hibernate One To One Bidirectional Mapping – Primary Key(Annotation)**](https://www.javainterviewpoint.com/hibernate-one-one-bidirectional-primary-key/)
* [**Hibernate Many To Many Mapping Example – Annotation**](https://www.javainterviewpoint.com/hibernate-many-many-mapping-example-annotation/)
* [**Hibernate CRUD Example in Eclipse (XML Mapping) with Maven + Oracle**](https://www.javainterviewpoint.com/hibernate-crud-example-xml/)
* [**Hibernate Inheritance – Table Per Class Hierarchy (XML Mapping & Annotation)**](https://www.javainterviewpoint.com/table-per-class-hierarchy/)
* [**Hibernate Inheritance – Table Per Subclass Hierarchy (XML Mapping & Annotation)**](https://www.javainterviewpoint.com/hibernate-inheritance-table-per-subclass-hierarchy/)
* [**Hibernate Inheritance – Table Per Concrete Class Hierarchy Example(XML Mapping & Annotation)**](https://www.javainterviewpoint.com/hibernate-inheritance-table-per-concrete-class-hierarchy/)
* [**Hibernate Composite Primary Key Tutorial – Using composite-id tag & Annotations**](https://www.javainterviewpoint.com/hibernate-composite-primary-key-tutorial/)
* [**Hibernate Embeddable Composite Primary Key | @Embeddable, @EmbeddedId**](https://www.javainterviewpoint.com/hibernate-embeddable-composite-primary-key/)
* [**Component Mapping in Hibernate Using Annotations | @Embeddable & @Embedded**](https://www.javainterviewpoint.com/component-mapping-in-hibernate-annotations/)
* [**Hibernate Component Mapping using XML**](https://www.javainterviewpoint.com/hibernate-component-mapping-xml/)
* [**Difference between session.get() and session.load() in Hibernate**](https://www.javainterviewpoint.com/difference-between-session-get-and-session-load-hibernate/)

## Hibernate One To Many Mapping Example

### ****Author.java****

Create a new Java file **Author.java** under the package **com.javainterviewpoint** and add the following code

package com.javainterviewpoint;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

@Entity

@Table(name="AUTHOR")

public class Author

{

@Id

@GeneratedValue

@Column(name="AUTHOR\_ID")

private int authorId;

@Column(name="AUTHOR\_NAME")

private String authorName;

@OneToMany(mappedBy="author",cascade = CascadeType.ALL)

private Set<Book> books;

public Author()

{

super();

}

public Author(int authorId, String authorName, Set books)

{

super();

this.authorId = authorId;

this.authorName = authorName;

this.books = books;

}

public int getAuthorId()

{

return authorId;

}

public void setAuthorId(int authorId)

{

this.authorId = authorId;

}

public String getAuthorName()

{

return authorName;

}

public void setAuthorName(String authorName)

{

this.authorName = authorName;

}

public Set<Book> getBooks()

{

return books;

}

public void setBooks(Set<Book> books)

{

this.books = books;

}

@Override

public String toString()

{

return "Author [authorId=" + authorId + ", authorName=" + authorName + ", books=" + books + "]";

}

}

Our **Author** class is a simple **POJO** class consisting of the **getters** and **setters** for the **Author** properties **(authorId, authorName, books)**.

In the POJO class, we have used the below **JPA Annotations**.

1. **@Entity** – This annotation will mark our **Employee** class as an **Entity Bean**.
2. **@Table** – **@Table** annotation will map our class to the corresponding database table. You can also specify other attributes such as **indexes, catalog, schema, uniqueConstraints**. The **@Table**annotation is an optional annotation if this annotation is not provided then the class name will be used as the table name.
3. **@Id** –  The **@Id** annotation marks the particular field as the primary key of the **Entity**.
4. **@GeneratedValue –**This annotation is used to specify how the primary key should be generated. Here **SEQUENCE** Strategy will be used as this the default strategy for Oracle
5. **@OneToMany** – We have used the **mappedBy** attribute – This denotes the property which will be used for mapping purpose, here we have an attribute **“author”** so in our **Book** class we should have this attribute. This is a mandatory annotation.
6. **@Column** – This annotation maps the corresponding fields to their respective columns in the database table.

### ****Book.java****

Create a new Java file **Book.java** under the package **com.javainterviewpoint** and add the following code

package com.javainterviewpoint;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

@Entity

@Table(name="BOOK")

public class Book

{

@Id

@GeneratedValue

@Column(name="BOOK\_ID")

private int bookId;

@Column(name="BOOK\_TITLE")

private String bookTitle;

@Column(name="BOOK\_DESCRIPTION")

private String bookDescription;

@ManyToOne

@JoinColumn(name="AUTHOR\_ID")

private Author author;

public Book()

{

super();

}

public Book(int bookId, String bookTitle, String bookDescription, Author author)

{

super();

this.bookId = bookId;

this.bookTitle = bookTitle;

this.bookDescription = bookDescription;

this.author = author;

}

public int getBookId()

{

return bookId;

}

public void setBookId(int bookId)

{

this.bookId = bookId;

}

public String getBookTitle()

{

return bookTitle;

}

public void setBookTitle(String bookTitle)

{

this.bookTitle = bookTitle;

}

public String getBookDescription()

{

return bookDescription;

}

public void setBookDescription(String bookDescription)

{

this.bookDescription = bookDescription;

}

public Author getAuthor()

{

return author;

}

public void setAuthor(Author author)

{

this.author = author;

}

@Override

public String toString()

{

return "Book [bookId=" + bookId + ", bookTitle=" + bookTitle + ", bookDescription=" + bookDescription

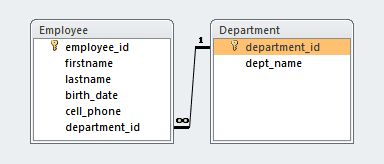
+ ", author=" + author + "]";

}

}

Employee and Department table exhibits One-to-many relationship.

Each Department can be associated with multiple Employees and each Employee can have only one Department.



public class Department

@OneToMany(mappedBy="department")

private Set<Employee> employees;

public class Employee

@ManyToOne

@JoinColumn(name="department\_id")

private Department department;

