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

Bidirectional Association

Bidirectional association allows us to fetch details of dependent object from both side. In such case, we have the reference of two classes in each other.

Let's take an example of Employee and Address, if Employee class has-a reference of Address and Address has a reference of Employee. Additionally, you have applied one-to-one or one-to-many relationship for the classes in mapping file as well, it is known as bidirectional association.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Hibernate One to One Mapping Annotation Example

If you are working on any hibernate project or you are planning to work on any in future, then you can easily understand the one-to-one relationships between several entities in your application. In this hibernate one to one mapping example, We will discuss 3 different variations of this mapping supported by hibernate.

Table fo contents

Various supported techniques for one to one mapping

1. Using foreign key association

2. Using common join table

3. Using shared primary key

For this hibernate one to one mapping example, I am extending the example written for [hibernate hello world example](https://howtodoinjava.com/hibernate/hibernate-3-introduction-and-writing-hello-world-application/). We have two entities here: Employee and Account.

An employee can have one account. Similarily, an account will be associated with one employee only. It’s one to one relationship for this example.

## Various supported techniques

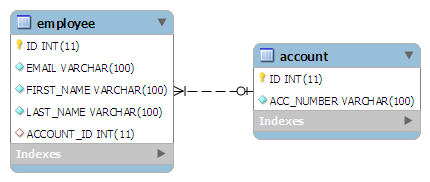
In hibernate there are 3 ways to create one-to-one relationships between two entities. Either way we have to use [**@OneToOne**](https://docs.oracle.com/javaee/5/api/javax/persistence/OneToOne.html) annotation.

1. First technique is most widely used and uses a **foreign key column in one of the tables**.
2. Second technique uses a rather known solution of having a **third table to store mapping** between first two tables.
3. Third technique is something new which uses a **common primary key value in both the tables**.

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

In this kind of association, a foreign key column is created in **owner entity**. For example, if we make **EmployeeEntity** owner, then a extra column "ACCOUNT\_ID" will be created in Employee table. This column will store the foreign key for Account table.

Table structure will be like this:

[](https://howtodoinjava.files.wordpress.com/2012/11/foreign-key-association-one-to-one.png)

To make such association, refer the Account entity in EmployeeEntity class as follow:

|  |
| --- |
| EmployeeEntity.java |
| @OneToOne  @JoinColumn(name="ACCOUNT\_ID")  private AccountEntity account; |

The join column is declared with the [**@JoinColumn**](https://docs.oracle.com/javaee/5/api/javax/persistence/JoinColumn.html) annotation which looks like the [**@Column**](https://docs.oracle.com/javaee/5/api/javax/persistence/Column.html) annotation. It has one more parameters named referencedColumnName. This parameter declares the column in the targeted entity that will be used to the join.

If no @JoinColumn is declared on the owner side, the defaults apply. A join column(s) will be created in the owner table and its name will be the concatenation of the name of the relationship in the owner side, \_ (underscore), and the name of the primary key column(s) in the owned side.(

In a bidirectional relationship, one of the sides (and only one) has to be the owner. The owner is responsible for the association column(s) update. To declare a side as not responsible for the relationship, the attribute ***[mappedBy](https://docs.oracle.com/javaee/5/api/javax/persistence/OneToOne.html" \l "mappedBy%28%29" \o "mappedBy)*** is used. ‘mappedBy’ refers to the property name of the association on the owner side.

|  |
| --- |
| AccountEntity.java |
| @OneToOne(mappedBy="account")  private EmployeeEntity employee; |

Above “mappedBy” attribute declares that it is dependent on owner entity for mapping.

Lets test above mappings in running code:

|  |
| --- |
| TestForeignKeyAssociation.java |
| public class TestForeignKeyAssociation {        public static void main(String[] args) {          Session session = HibernateUtil.getSessionFactory().openSession();          session.beginTransaction();            AccountEntity account = new AccountEntity();          account.setAccountNumber("123-345-65454");            // Add new Employee object          EmployeeEntity emp = new EmployeeEntity();          emp.setEmail("demo-user@mail.com");          emp.setFirstName("demo");          emp.setLastName("user");            // Save Account          session.saveOrUpdate(account);          // Save Employee          emp.setAccount(account);          session.saveOrUpdate(emp);            session.getTransaction().commit();          HibernateUtil.shutdown();      }  } |

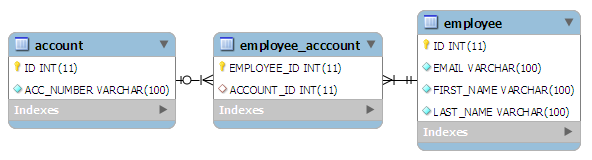
Running above code creates desired schema in database and run these SQL queries.

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

You can verify the data and mappings in both tables when you run above program.

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

This approach is not new to all of us. Lets start with targeted DB structure in this technique.

[](https://howtodoinjava.files.wordpress.com/2012/11/join-table-one-to-one-mapping.png)

In this technique, main annotation to be used is [**@JoinTable**](https://docs.oracle.com/javaee/5/api/javax/persistence/JoinTable.html). **This annotation is used to define the new table name (mandatory) and foreign keys from both of the tables**. Lets see how it is used:

|  |
| --- |
| EmployeeEntity.java |
| @OneToOne(cascade = CascadeType.ALL)  @JoinTable(name="EMPLOYEE\_ACCCOUNT", joinColumns = @JoinColumn(name="EMPLOYEE\_ID"),  inverseJoinColumns = @JoinColumn(name="ACCOUNT\_ID"))  private AccountEntity account; |

**@JoinTable** annotation is used in EmployeeEntity class. It declares that a new table EMPLOYEE\_ACCOUNT will be created with two columns EMPLOYEE\_ID (primary key of EMPLOYEE table) and ACCOUNT\_ID (primary key of ACCOUNT table).

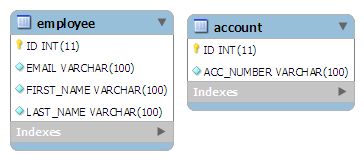
Testing above entities generates following SQL queries in log files:

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

## 3. Hibernate one to one mapping with shared primary key

In this technique, hibernate will ensure that it will use a common primary key value in both the tables. This way primary key of EmployeeEntity can safely be assumed the primary key of AccountEntity also.

Table structure will be like this:

[](https://howtodoinjava.files.wordpress.com/2012/11/shared-primary-key-one-to-one.png)

In this approach, [**@PrimaryKeyJoinColumn**](https://docs.oracle.com/javaee/5/api/javax/persistence/PrimaryKeyJoinColumn.html) is the main annotation to be used. Let see how to use it.

|  |
| --- |
| EmployeeEntity.java |
| @OneToOne(cascade = CascadeType.ALL)  @PrimaryKeyJoinColumn  private AccountEntity account; |

In AccountEntity side, it will remain dependent on owner entity for the mapping.

|  |
| --- |
| AccountEntity.java |
| @OneToOne(mappedBy="account", cascade=CascadeType.ALL)  private EmployeeEntity employee; |

Testing above entities generates following SQL queries in log files:

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

So, we have seen all **3 different ways to create one to one mapping** supported in hibernate. I will suggest you to download the source code and play with it.

Happy Learning !!

<https://www.javainterviewpoint.com/hibernate-one-one-bidirectional-primary-key/>

# Hibernate One To One Bidirectional Mapping Example – Foreign Key(Annotation)

In this approach, we will have two tables with different**primary keys**. The **primary key**of **EMPLOYEE** table **EMP\_ID** will act as a **foreign key** for the **EMPLOYEE\_ADDRESS** table and **EMPLOYEE\_ADDRESS** table will have its own **primary key ADDR\_ID**.

### Creating table

Create **EMPLOYEE** and **EMPLOYEE\_ADDRESS** Tables, simply Copy and Paste the following SQL query in the query editor to get the table created.

CREATE TABLE "EMPLOYEE"

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

"NAME" VARCHAR2(255 CHAR),

PRIMARY KEY ("EMP\_ID")

);

CREATE TABLE "EMPLOYEE\_ADDRESS"

(

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

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

"STREET" VARCHAR2(255 CHAR),

"CITY" VARCHAR2(255 CHAR),

"STATE" VARCHAR2(255 CHAR),

"COUNTRY" VARCHAR2(255 CHAR),

PRIMARY KEY ("ADDR\_ID"),

CONSTRAINT fk\_emp FOREIGN KEY ("EMP\_ID") REFERENCES EMPLOYEE ("EMP\_ID")

);

### ****Employee.java****

Create a new Java file **Employee.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.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.Table;

@Entity

@Table(name="EMPLOYEE")

public class Employee

{

@Id

@GeneratedValue

@Column(name="EMP\_ID")

private int empId;

@Column(name="NAME")

private String empName;

@OneToOne(mappedBy="employee")

private Employee\_Address employeeAddress;

public Employee()

{

super();

}

public Employee(int empId, String empName, Employee\_Address employeeAddress)

{

super();

this.empId = empId;

this.empName = empName;

this.employeeAddress = employeeAddress;

}

public int getEmpId()

{

return empId;

}

public void setEmpId(int empId)

{

this.empId = empId;

}

public String getEmpName()

{

return empName;

}

public void setEmpName(String empName)

{

this.empName = empName;

}

public Employee\_Address getEmployeeAddress()

{

return employeeAddress;

}

public void setEmployeeAddress(Employee\_Address employeeAddress)

{

this.employeeAddress = employeeAddress;

}

@Override

public String toString()

{

return "Employee [empId=" + empId + ", empName=" + empName + ", employeeAddress=" + employeeAddress + "]";

}

}

### ****Employee\_Address.java****

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

package com.javainterviewpoint;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OneToOne;

import javax.persistence.Table;

@Entity

@Table(name="EMPLOYEE\_ADDRESS")

public class Employee\_Address

{

@Id

@Column(name = "ADDR\_ID")

@GeneratedValue

private int addrId;

@Column(name="STREET")

private String street;

@Column(name="CITY")

private String city;

@Column(name="STATE")

private String state;

@Column(name="COUNTRY")

private String country;

@OneToOne(cascade= CascadeType.ALL)

@JoinColumn(name = "EMP\_ID")

private Employee employee;

public Employee\_Address()

{

super();

}

public Employee\_Address(int addrId, String street, String city, String state, String country, Employee employee)

{

super();

this.addrId = addrId;

this.street = street;

this.city = city;

this.state = state;

this.country = country;

this.employee = employee;

}

public int getAddrId()

{

return addrId;

}

public void setAddrId(int addrId)

{

this.addrId = addrId;

}

public String getStreet()

{

return street;

}

public void setStreet(String street)

{

this.street = street;

}

public String getCity()

{

return city;

}

public void setCity(String city)

{

this.city = city;

}

public String getState()

{

return state;

}

public void setState(String state)

{

this.state = state;

}

public String getCountry()

{

return country;

}

public void setCountry(String country)

{

this.country = country;

}

public Employee getEmployee()

{

return employee;

}

public void setEmployee(Employee employee)

{

this.employee = employee;

}

@Override

public String toString()

{

return "Employee\_Address [addrId=" + addrId + ", street=" + street + ", city=" + city + ", state=" + state

+ ", country=" + country + ", employee=" + employee + "]";

}

}

**Hibernate One To One Bidirectional Mapping – Primary Key(Annotation)**

In this approach, we will have two tables sharing the**same primary key**. In our example, we have two tables **EMPLOYEE** and **EMPLOYEE\_ADDRESS** sharing the same **primary key** **EMP\_ID**. This gives us the possibility of traversing **EMPLOYEE\_ADDRESS** from **EMPLOYEE** and vice-versa and hence called as Bidirectional relationship.

### Creating table

Create **EMPLOYEE** and **EMPLOYEE\_ADDRESS** Tables, simply Copy and Paste the following SQL query in the query editor to get the table created.

CREATE TABLE "EMPLOYEE"

(

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

"EMP\_NAME" VARCHAR2(255 CHAR),

PRIMARY KEY (EMP\_ID)

);

CREATE TABLE "EMPLOYEE\_ADDRESS"

(

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

"STREET" VARCHAR2(255 CHAR),

"CITY" VARCHAR2(255 CHAR),

"STATE" VARCHAR2(255 CHAR),

"COUNTRY" VARCHAR2(255 CHAR),

PRIMARY KEY (EMP\_ID)

);

### ****Employee.java****

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

package com.javainterviewpoint;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.PrimaryKeyJoinColumn;

import javax.persistence.Table;

@Entity

@Table(name="EMPLOYEE")

public class Employee

{

@Id

@GeneratedValue

@Column(name="EMP\_ID")

private int empId;

@Column(name="EMP\_NAME")

private String empName;

@OneToOne(cascade = CascadeType.ALL)

@PrimaryKeyJoinColumn

private Employee\_Address employeeAddress;

public Employee()

{

super();

}

public Employee(int empId, String empName, Employee\_Address employeeAddress)

{

super();

this.empId = empId;

this.empName = empName;

this.employeeAddress = employeeAddress;

}

public int getEmpId()

{

return empId;

}

public void setEmpId(int empId)

{

this.empId = empId;

}

public String getEmpName()

{

return empName;

}

public void setEmpName(String empName)

{

this.empName = empName;

}

public Employee\_Address getEmployeeAddress()

{

return employeeAddress;

}

public void setEmployeeAddress(Employee\_Address employeeAddress)

{

this.employeeAddress = employeeAddress;

}

@Override

public String toString()

{

return "Employee [empId=" + empId + ", empName=" + empName + ", employeeAddress=" + employeeAddress + "]";

}

}

### ****Employee\_Address.java****

Create a new Java file **Employee\_Address.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.OneToOne;

import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

import org.hibernate.annotations.Parameter;

@Entity

@Table(name="EMPLOYEE\_ADDRESS")

public class Employee\_Address

{

@Id

@GeneratedValue(generator="fkgen")

@GenericGenerator(name="fkgen",strategy="foreign",

parameters=@Parameter(name="property",value="employee"))

@Column(name="EMP\_ID")

private int empId;

@Column(name="STREET")

private String street;

@Column(name="CITY")

private String city;

@Column(name="STATE")

private String state;

@Column(name="COUNTRY")

private String country;

@OneToOne(mappedBy="employeeAddress")

private Employee employee;

public Employee\_Address()

{

super();

}

public Employee\_Address(int empId, String street, String city, String state, String country, Employee employee)

{

super();

this.empId = empId;

this.street = street;

this.city = city;

this.state = state;

this.country = country;

this.employee = employee;

}

public int getEmpId()

{

return empId;

}

public void setEmpId(int empId)

{

this.empId = empId;

}

public String getStreet()

{

return street;

}

public void setStreet(String street)

{

this.street = street;

}

public String getCity()

{

return city;

}

public void setCity(String city)

{

this.city = city;

}

public String getState()

{

return state;

}

public void setState(String state)

{

this.state = state;

}

public String getCountry()

{

return country;

}

public void setCountry(String country)

{

this.country = country;

}

public Employee getEmployee()

{

return employee;

}

public void setEmployee(Employee employee)

{

this.employee = employee;

}

@Override

public String toString()

{

return "Employee\_Address [empId=" + empId + ", street=" + street + ", city=" + city + ", state=" + state

+ ", country=" + country + ", employee=" + employee + "]";

}

}

**each employee have a unique employeedetails.**

public class Employee

@OneToOne(mappedBy="employee", fetch = FetchType.LAZY, cascade=CascadeType.ALL)

private EmployeeDetail employeeDetail;

public class EmployeeDetail

@OneToOne

@PrimaryKeyJoinColumn

private Employee employee;

public class User

@OneToOne(fetch = FetchType.LAZY,cascade = CascadeType.ALL,mappedBy = "user")

private UserProfile userProfile;

-----------------------------------------

public class UserProfile

@OneToOne(fetch = FetchType.LAZY, optional = false)

@JoinColumn(name = "user\_id", nullable = false)

private User user;

public class Library

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(unique = true)

private Address address;

---------------------------------------

public class Address

@OneToOne(mappedBy = "address")

private Library library;

public class Book

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "book\_detail\_id")

private BookDetail bookDetail;

--------------------------------------

public class BookDetail

@OneToOne(mappedBy = "bookDetail")

private Book book;

public class Person

No Need

------------------------

public class PersonDetail

@OneToOne(fetch=FetchType.LAZY)

@PrimaryKeyJoinColumn

private Person person;

**public** **class** Student

1. @OneToOne(cascade = CascadeType.ALL)
2. @JoinColumn(name = "ADDRESS\_ID")
3. **private** Address address;

----------------------

**public** **class** Address

1. @OneToOne(mappedBy = "address")
2. **private** Student student;

public class Employee

@OneToOne(mappedBy="employee")

private Employee\_Address employeeAddress;

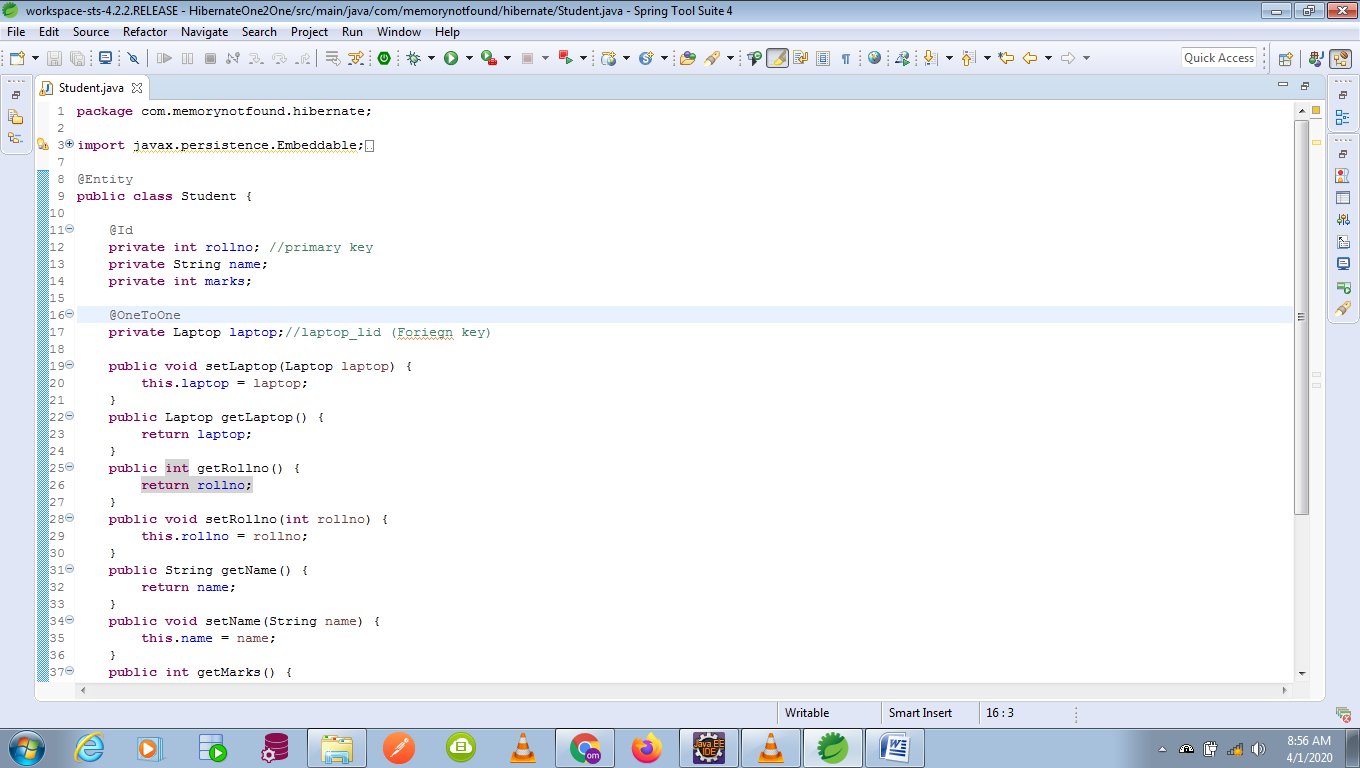
---------------------------------------

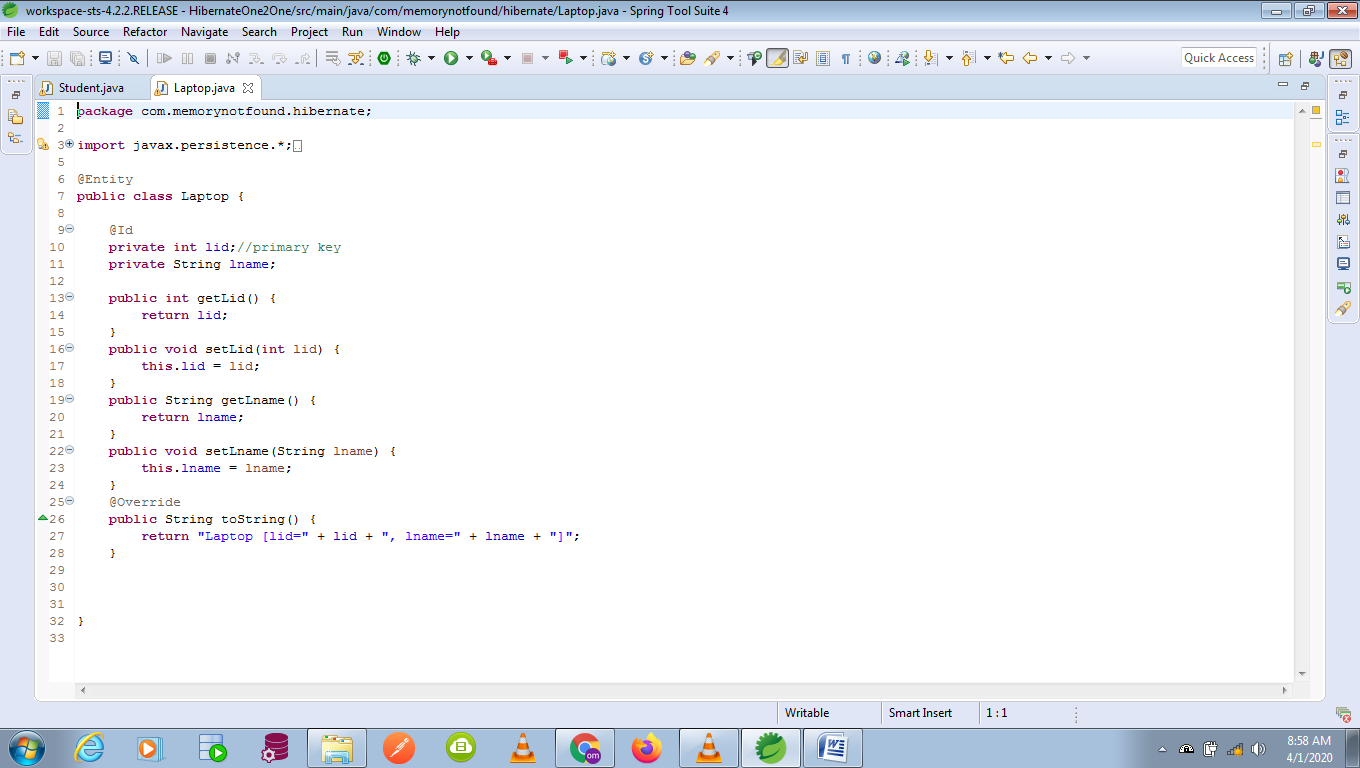
public class Employee\_Address

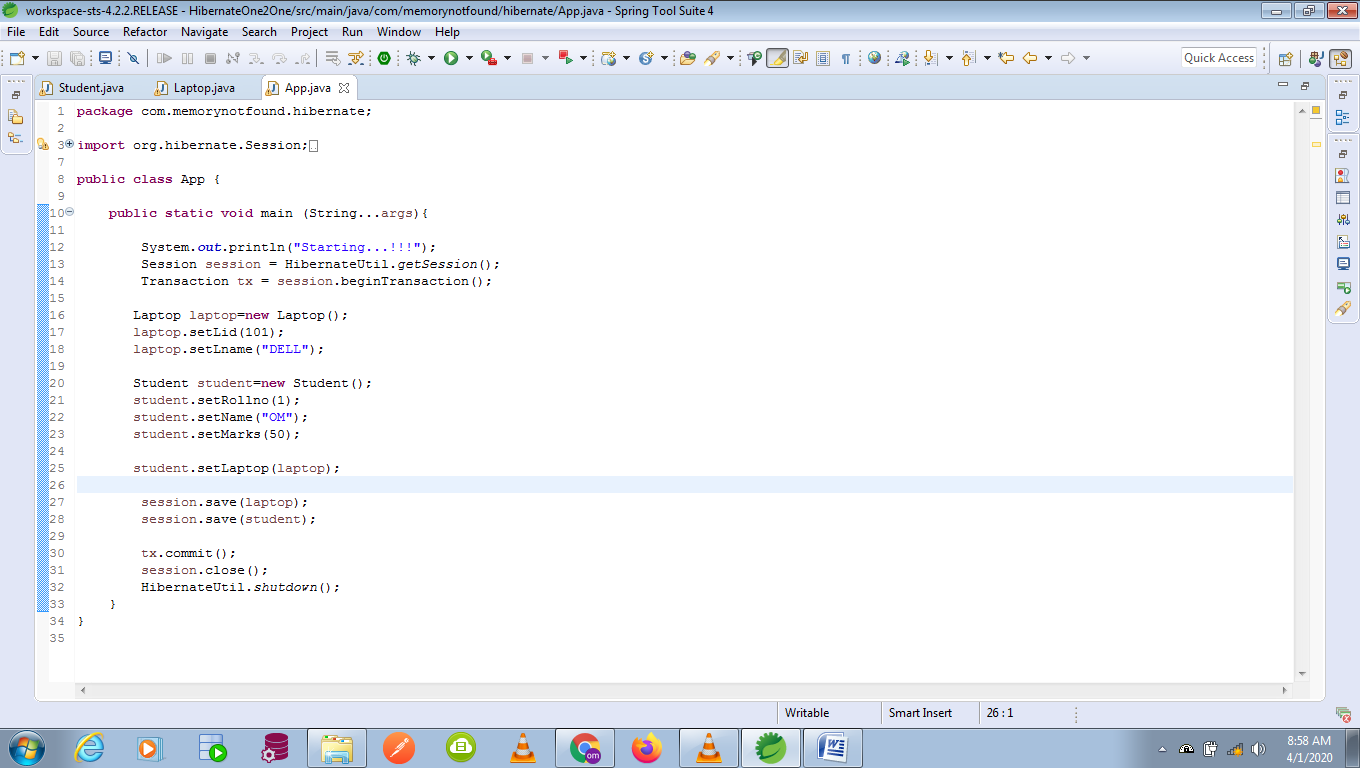
@OneToOne(cascade= CascadeType.ALL)

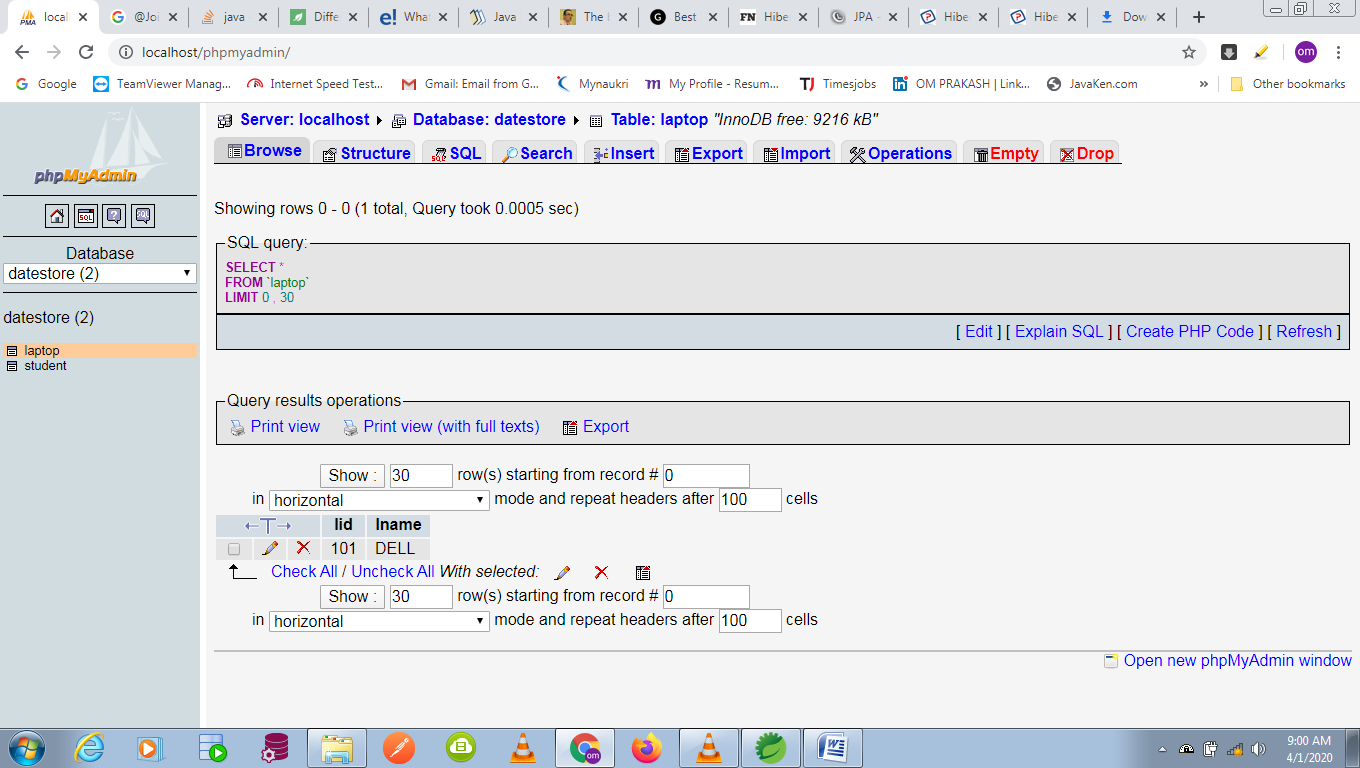
@JoinColumn(name = "EMP\_ID")

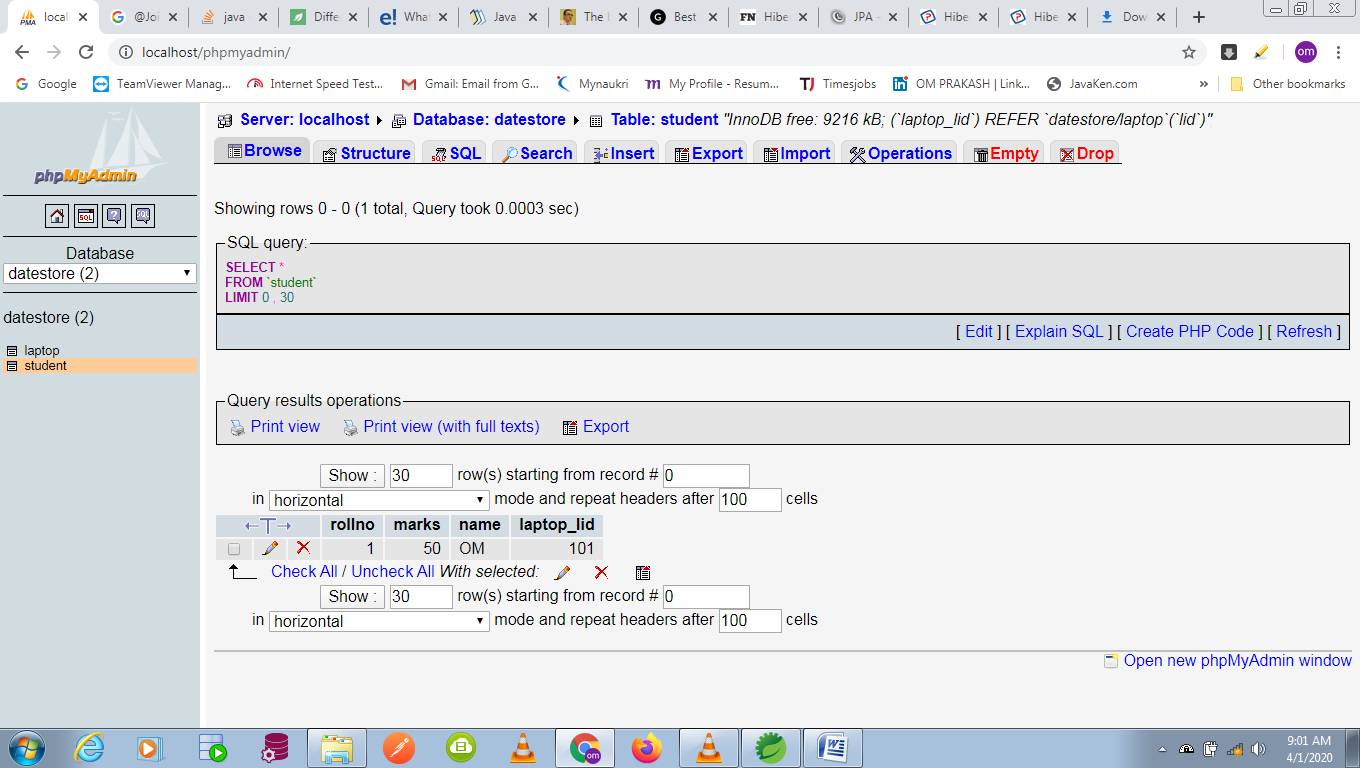
private Employee employee;











# [JPA, Hibernate: OneToOne mapping with foreign key only](https://stackoverflow.com/questions/25376101/jpa-hibernate-onetoone-mapping-with-foreign-key-only)

public class Company {

@Id

@GeneratedValue

private Long id;

private String name;

@OneToOne(mappedBy = "company",cascade = CascadeType.ALL)

private Address companyAddress;

//getters and setters

}

public class Address {

@Id

@GeneratedValue

private Long id;

private String address;

@OneToOne

@PrimaryKeyJoinColumn

private Company company;

//getters and setters

}