

# Jakarta Persistence API

<https://jakarta.ee/specifications/persistence/3.1/jakarta-persistence-spec-3.1>

## Cài đặt MariaDB

<https://mariadb.org/>

[Download MariaDB Server](#)  
REST API

[Release Schedule](#)

[Reporting Bugs](#)

[MariaDB Server Statistics](#)

[Download MariaDB Server Documentation](#)

[View all releases for:](#)  
MariaDB Server  
Connector/C  
Connector/J  
Connector/ODBC  
Connector/Python  
Connector/Node.js

MariaDB Server Version  
MariaDB Server 11.2.2

Display older releases: ☐

Operating System  
Windows

Architecture  
x86\_64

Package Type  
MSI Package

Download

Mirror  
BKNS.VN - Hanoi

Release date: 2023-11-21  
File name: mariadb-11.2.2-winx64.msi  
File size: 72.4 MB  
• Download galera-26.4.16  
Display signature and checksums

MariaDB 11.2 (x64) Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

MariaDB Server

Database instance

Client Programs

Backup utilities

Development Components

Third party tools

HeidiSQL

Installs C/C++ header files and libraries

This feature requires 11MB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 396KB on your hard drive.


Location: C:\MariaDB 11.2\

Browse...

Reset Disk Usage Back Next Cancel

User settings

Default instance properties  
MariaDB 11.2 (x64) database configuration

MariaDB Server 

☒ **Modify password for database user 'root'**

New root password:  Enter new root password

Confirm:  Retype the password

☒ **Enable access from remote machines for 'root' user**

☒ **Use UTF8 as default server's character set**


Data directory

User name: root

Password: root

Database settings

Default instance properties  
MariaDB 10.3 (x64) database configuration

MariaDB Server 

☒ **Install as service**

Service Name:

☒ **Enable networking**

TCP port:

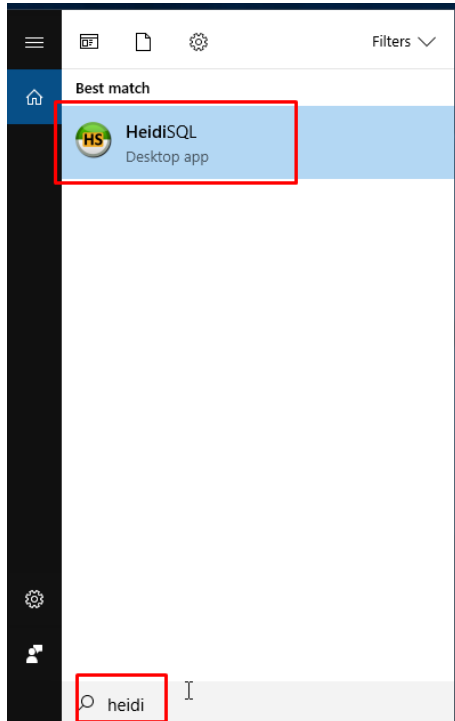
**InnoDB engine settings**

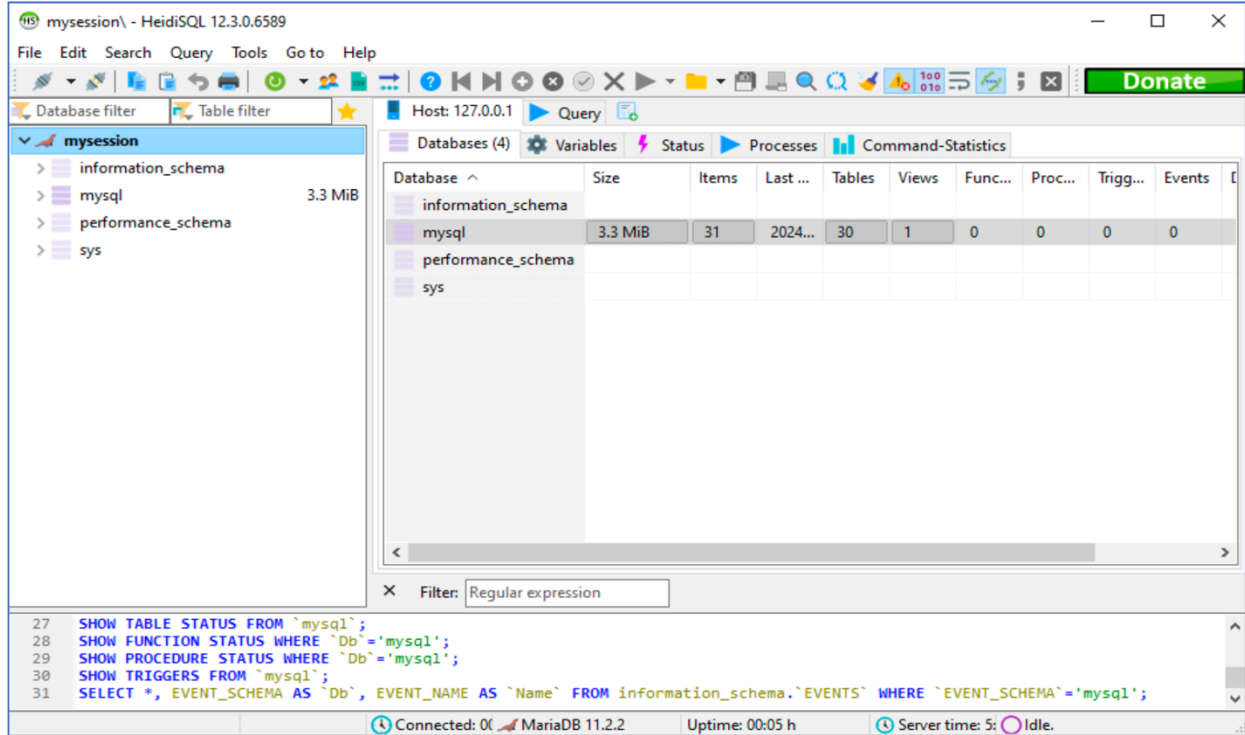
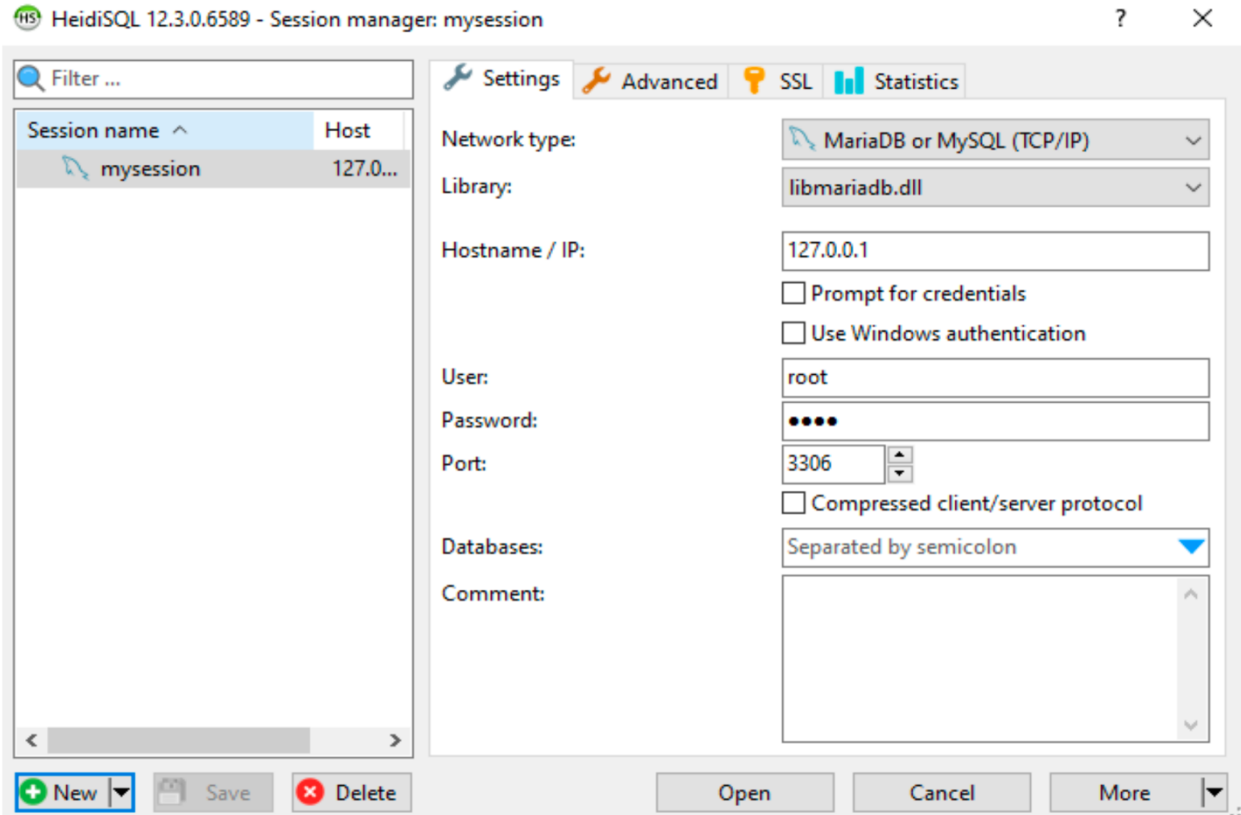
Buffer pool size:  MB

Page size:  KB

## HeidiSQL

- HeidiSQL là một công cụ quản trị cơ sở dữ liệu quan hệ mã nguồn mở.
- Nó hỗ trợ nhiều cơ sở dữ liệu quan hệ như MySQL, PostgreSQL, MariaDB, MSSQL, SQLite, Oracle, DB2, SQL Server, Sybase, ...
- Cho phép lưu trữ, cập nhật, xóa dữ liệu từ cơ sở dữ liệu quan hệ.





## Jakarta Persistence API (JPA)

### - JPA - ORM là gì?

- JPA là viết tắt của Java Persistence API
- ORM là viết tắt của Object-Relational Mapping
- Là một phần của Java EE và được sử dụng để ánh xạ mô hình đối tượng của Java với cơ sở dữ liệu quan hệ.
- JPA cho phép ánh các lớp thực thể của Java vào các bảng trong cơ sở dữ liệu quan hệ.
- Cho phép ánh xạ các quan hệ giữa các lớp thực thể của Java thành các quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ.
- Nó cung cấp một cách tiêu chuẩn để tạo, lưu trữ, cập nhật và xóa dữ liệu từ cơ sở dữ liệu quan hệ sử dụng các lớp thực thể của Java.

### - Anatomy of an Entity - Cấu trúc của một thực thể

- Một entity là POJO (Plain Old Java Object) được đánh dấu bằng @Entity annotation.
- Một class thực thể biểu diễn cho một table trong cơ sở dữ liệu quan hệ.
- Bắt buộc phải có một constructor không tham số (default constructor).
- Bắt buộc phải có một hoặc vài fields được đánh dấu bằng @Id annotation để biểu diễn là primary key trong bảng.
  - Khóa chính có một field gọi là khóa đơn (Single primary key)
  - Khóa chính có từ hai fields tham gia vào khóa gọi là khóa phức hợp (*Composite primary key*)
- Lớp thực thể không thể là final class hoặc enum.
- Những field được đánh dấu bằng @Transient annotation hoặc transient sẽ không được ánh xạ vào cơ sở dữ liệu.
- Mỗi đối tượng (instance) được tạo từ một lớp thực thể, được biểu diễn cho một hàng (row) trong table.
- Mỗi field của một lớp thực thể, được biểu diễn là một cột trong bảng.

### - Deployment tools

- Java: Java 1.8+
- RDBMS: MySQL, PostgreSQL, MariaDB, H2, MSSQL ...
- Tools for Java developers: IntelliJ IDEA, Eclipse IDE for Java EE Developers
- Framework/Library: Hibernate / Eclipselink

### - EclipseLink là gì?

- EclipseLink là một framework ORM (Object-Relational Mapping) mã nguồn mở được phát triển bởi Eclipse Foundation.
- Nó cung cấp một cách tiêu chuẩn để ánh xạ mô hình đối tượng của Java với cơ sở dữ liệu quan hệ.
- Nó cung cấp một cách tiêu chuẩn để tạo, lưu trữ, cập nhật và xóa dữ liệu từ cơ sở dữ liệu quan hệ sử dụng các lớp thực thể của Java.

### - Hibernate ORM là gì?

- Hibernate ORM là một framework ORM (Object-Relational Mapping) mã nguồn mở được phát triển bởi Red Hat.
- Nó cung cấp một cách tiêu chuẩn để ánh xạ mô hình đối tượng của Java với cơ sở dữ liệu quan hệ.
- Nó cung cấp một cách tiêu chuẩn để tạo, lưu trữ, cập nhật và xóa dữ liệu từ cơ sở dữ liệu quan hệ sử dụng các lớp thực thể của Java.

### - Ưu điểm:

- Hỗ trợ nhiều cơ sở dữ liệu quan hệ như: MySQL, PostgreSQL, MariaDB, H2, MSSQL, Oracle...
- Hỗ trợ nhiều tính năng như: lazy loading, caching, inheritance, validation, logging, querying, sorting, filtering, JPQL, criteria, ...

## Tạo JPA project

*Tạo một Maven Project*

*Hoặc tạo JPA Project, sau đó Convert to Maven Project*

## Installation with ORM (EclipseLink and Hibernate JPA) and (MariaDB or MSSQL)

<https://eclipse.dev/eclipselink/documentation/4.0/solutions/solutions.html#CHDDDDAB>

Maven → pom.xml

```
<dependencies>  
<!--
```

```

https://mvnrepository.com/artifact/jakarta.persistence/jakarta.persistence-api -->
<dependency>
  <groupId>jakarta.persistence</groupId>
  <artifactId>jakarta.persistence-api</artifactId>
  <version>3.1.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>6.4.4.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-testing -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-testing</artifactId>
  <version>6.4.4.Final</version>
</dependency>
<!--https://mvnrepository.com/artifact/org.eclipse.persistence/eclipselink -->
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>eclipselink</artifactId>
  <version>4.0.2</version>
</dependency>
<!--https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>10.2.1.jre17</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.30</version>
  <scope>provided</scope>
</dependency>
<!--https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client -->
<dependency>
  <groupId>org.mariadb.jdbc</groupId>
  <artifactId>mariadb-java-client</artifactId>
  <version>3.3.3</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.10.2</version>
</dependency>
</dependencies>

```

## Chi tiết về persistence.xml

Trong JPA, **persistence.xml** được dùng để cấu hình project, trong đó phải có tối thiểu một *persistence-unit* nơi định nghĩa tất cả các *metadata* mà **EntityManagerFactory** cần như entity mapping, thông tin kết nối cơ sở dữ liệu, transactions và các cấu hình khác của JPA provider.

**persistence.xml**:

- Phải chứa tối thiểu một *persistence-unit*.

- Đặt trong thư mục `src/main/resource/META-INF/persistence.xml` (nếu dùng Maven)

Mỗi persistence unit sẽ được khai báo trong một tag tên là **<persistence-unit>**

- Thuộc tính **name** của **<persistence-unit>** tag khai báo tên gọi của persistence-unit và tên gọi này sẽ được sử dụng khi các bạn khởi tạo **EntityManagerFactory**
- Thuộc tính **transaction-type: RESOURCE\_LOCAL (desktop)** và **JTA (web)**

### Provider

- Dùng để chỉ định một JPA provider triển khai tất cả các API của JPA (Hibernate, EclipseLink, ...)

### Entity Mapping

- Mặc định Hibernate/EclipseLink... tìm kiếm tất cả các *entity mapping* (các class ánh xạ từ Java object xuống table trong csdl) dựa trên *@Entity* annotation → không cần khai báo các *entity mapping* class.

### *exclude-unlisted-classes*

- Nếu không muốn sử dụng cơ chế tự động tìm kiếm các *entity mapping* class → có thể gán **exclude-unlisted-classes** thành **true** để tắt tính năng này.

### *class*

- Sau khi gán giá trị **exclude-unlisted-classes = true** → cần chỉ định các *entity mapping class* thủ công thông qua thuộc tính **class**.  
`<class>entities.Message</class>`

### *Thông tin kết nối database:*

- *jakarta.persistence.jdbc.driver*
- *jakarta.persistence.jdbc.dialect*
- *jakarta.persistence.jdbc.url*
- *jakarta.persistence.jdbc.user*
- *jakarta.persistence.jdbc.password*

### *jakarta.persistence.schema-generation.database.action:*

properties này dùng để chỉ định JPA sẽ làm gì với database khi ứng dụng được triển khai

- none: Không có gì xảy ra với database.
- create: JPA sẽ tạo mới database
- drop-and-create: JPA sẽ xóa database cũ và tạo lại database
- drop: database sẽ bị delete

*hibernate.format\_sql*: hiển thị câu lệnh SQL ra console

*hibernate.use\_sql\_comments*: hiển thị comment cho SQL



src/main/resources/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0"
xmlns="https://jakarta.ee/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">
<persistence-unit name="JPA_ORM_Student_MSSQL" transaction-type="RESOURCE_LOCAL">
    <!-- Persistence Provider -->
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
        <!-- Database connection settings -->
        <property name="jakarta.persistence.jdbc.driver"
value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
        <property name="jakarta.persistence.jdbc.dialect"
value="org.hibernate.dialect.SQLServerDialect" />
        <property name="jakarta.persistence.jdbc.url"
value="jdbc:sqlserver://localhost:1433;databaseName=StudentDB;trustServerCertificate=true" />
        <property name="jakarta.persistence.jdbc.user" value="sa" />
        <property name="jakarta.persistence.jdbc.password"
value="sapassword" />
        <!-- Automatically export the schema -->
        <property name="jakarta.persistence.schema-generation.database.action" value="none" />
        <!-- Echo all executed SQL to console -->
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.format_sql" value="true" />
    </properties>
</persistence-unit>

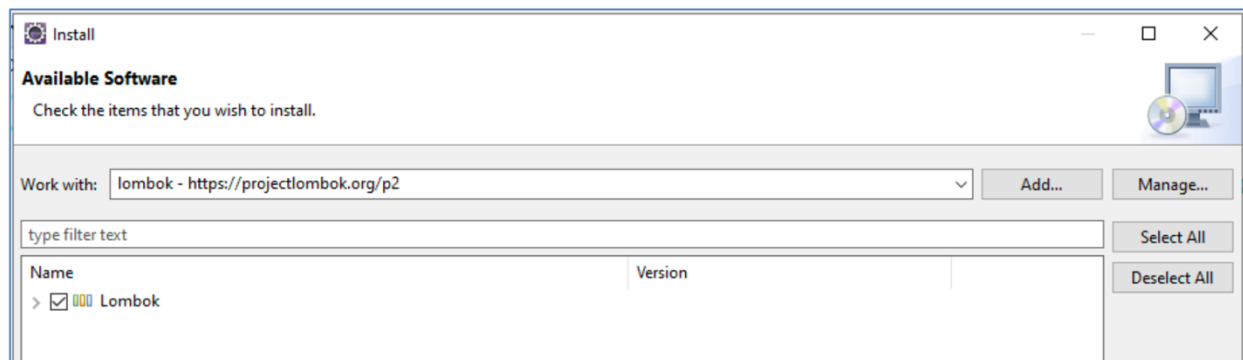
<persistence-unit name="JPA_ORM_Student_MYSQL" transaction-type="RESOURCE_LOCAL">
    <!-- Persistence Provider -->
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <class>entities.Student</class>
    <class>entities.Clazz</class>
    <class>entities.Course</class>
    <class>entities.Enrollment</class>

    <properties>
        <!-- Database connection settings -->
        <property name="jakarta.persistence.jdbc.dialect"
value="org.hibernate.dialect.MariaDBDialect" />
        <property name="jakarta.persistence.jdbc.driver"
value="org.mariadb.jdbc.Driver" />
        <property name="jakarta.persistence.jdbc.url"
value="jdbc:mariadb://localhost:3306/testdb" />
        <property name="jakarta.persistence.jdbc.user" value="root" />
        <property name="jakarta.persistence.jdbc.password"
value="root" />
        <!-- Automatically export the schema -->
        <property name="jakarta.persistence.schema-generation.database.action" value="none" />
        <!-- Echo all executed SQL to console -->
        <property name="eclipselink.logging.level.sql" value="FINE"/>
        <property name="eclipselink.logging.level" value="FINE"/>
        <property name="eclipselink.logging.parameters" value="true"/>
    </properties>
</persistence-unit>
</persistence>
```

## Cài đặt lombok trong eclipse

<https://projectlombok.org/setup/eclipse>

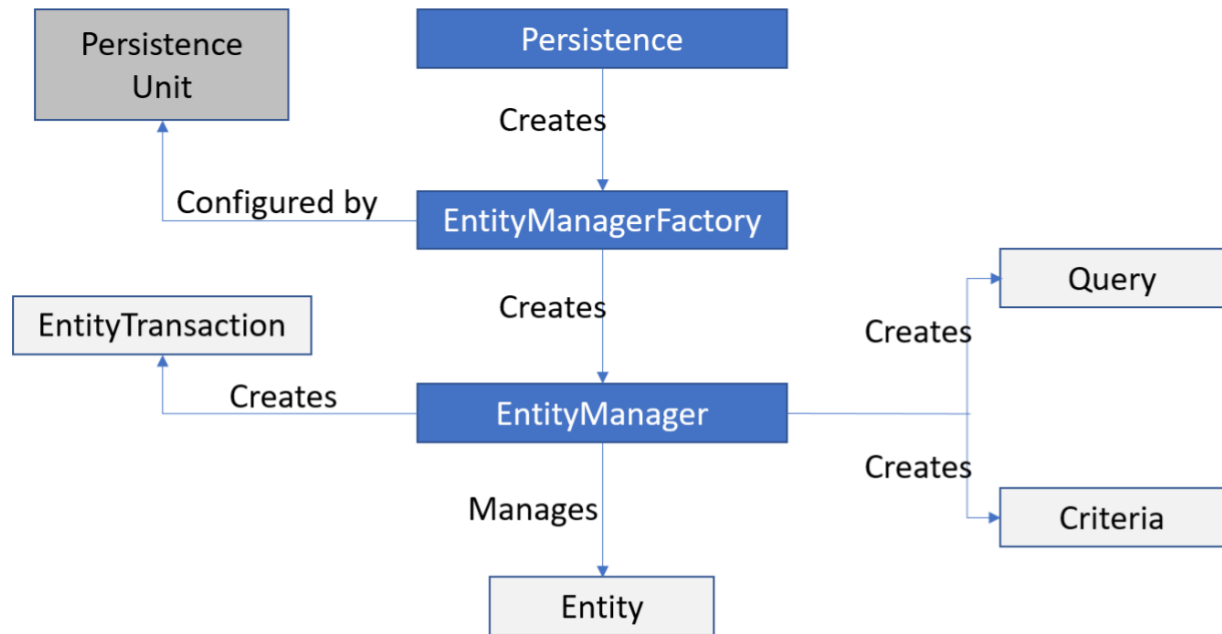
Help -> Install new Software ... ☐ Add: <https://projectlombok.org/p2>



## JPA Bootstrapping

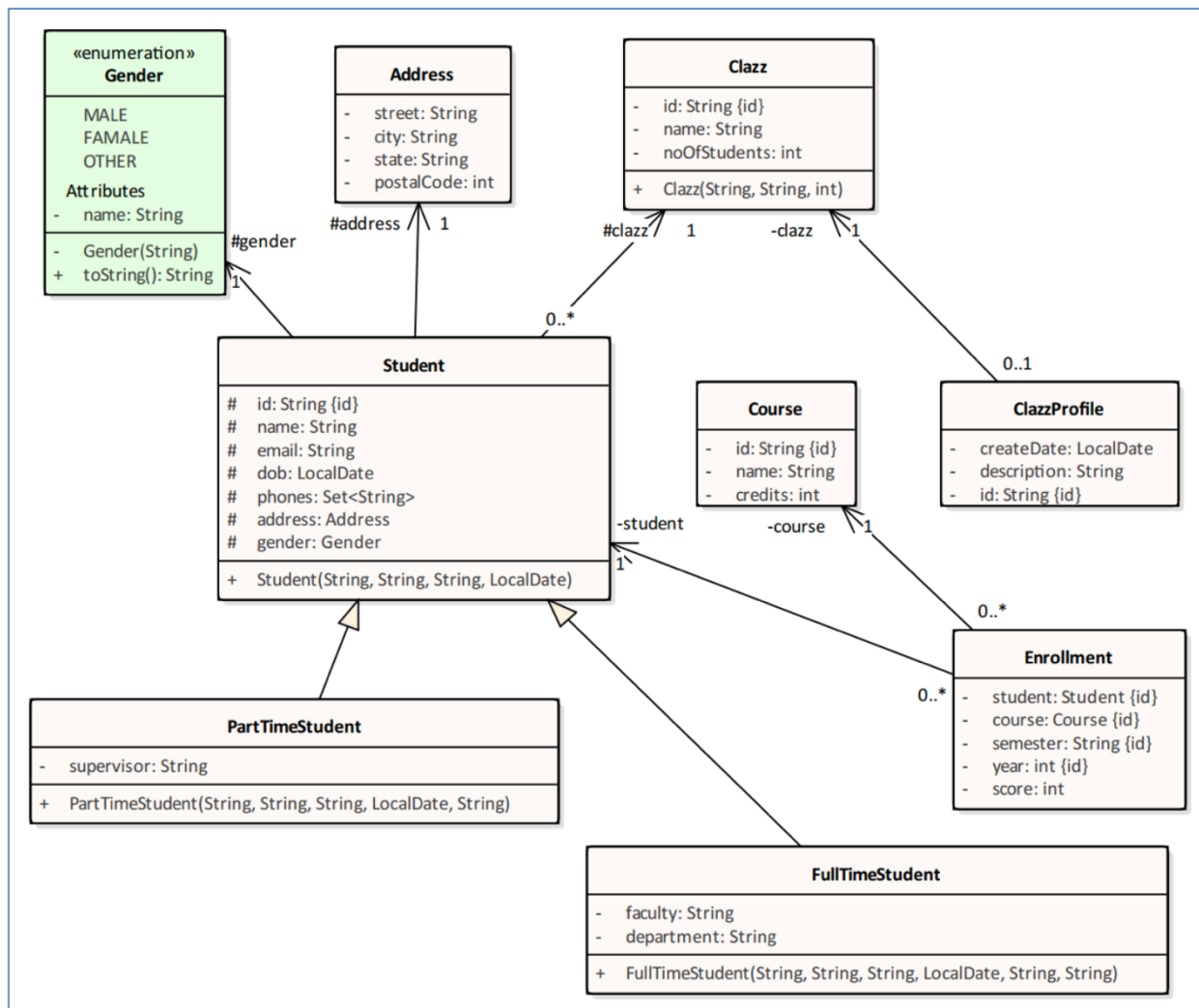
- Đặc tả JPA có 2 cách tiếp cận chính để tạo đối tượng `jakarta.persistence.EntityManager` từ `jakarta.persistence.EntityManagerFactory`, đó là: Java EE và Java SE.
- Container-bootstrapping (EE)
  - Container sẽ tạo `EntityManagerFactory` cho mỗi persistent-unit được cấu hình trong META-INF/persistence.xml.
- Application-bootstrapping (SE)
  - Ứng dụng tạo `EntityManagerFactory` thông qua phương thức `createEntityManagerFactory` của lớp `bootstrap jakarta.persistence.Persistence`.
- `EntityManager` quản lý các entity, cung cấp các phương thức cho các thao tác thêm, xóa, sửa và tìm kiếm dữ liệu như `persist`, `merge`, `remove`, `find` ...

## JPA Architecture



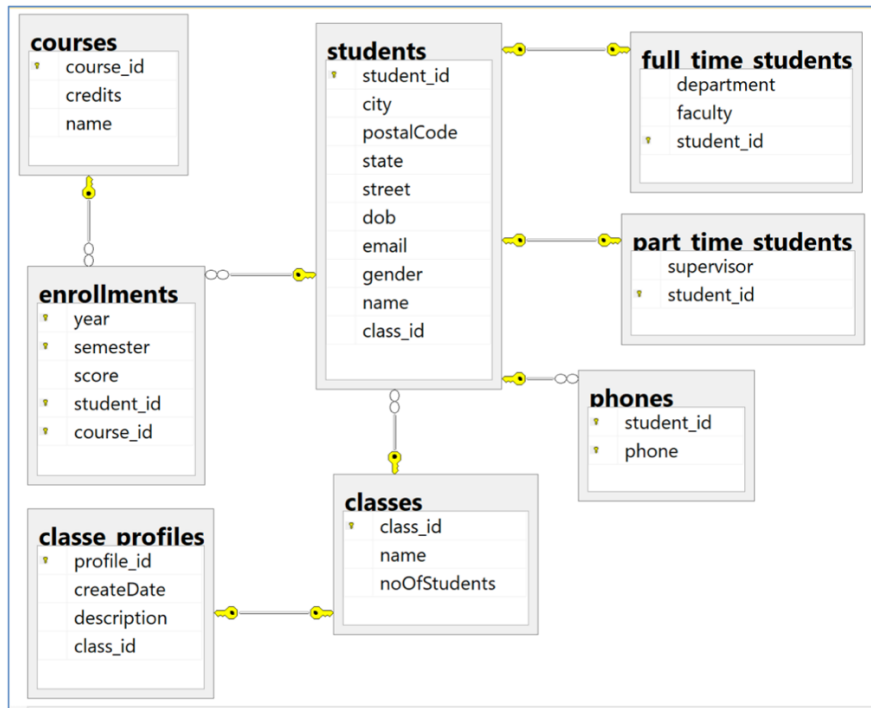
## Minh họa

Cho sơ đồ lớp sau:



## Yêu cầu:

- Ánh xạ mô hình lớp trên sang mô hình CSDL quan hệ tương ứng.



(Java Programming) Viết các phương thức với các thao tác CRUD

1. Thực hiện các thao tác Create, Update, Delete, Find by Id trên từng đối tượng.
2. Tính sĩ số sinh viên theo từng lớp học, giảm dần theo sĩ số.

+ getSisoByLophoc() : Map<Lophoc, Integer>

3. Tính điểm trung bình của các môn học của các sinh viên

+ listSinhvienDiemTB() : Map<Sinhvien, Float>

4. Những lớp học chưa có sinh viên

+ listLophocNull() : List<Lophoc>

5. Những sinh viên học môn “Lập Trình Phân tán với Công Nghệ Java” có điểm cao nhất.

+ listSinhvienGioiJava(): List<Sinhvien>

## Hướng dẫn

### Lớp entity

Các lớp entity phải (*chưa xét tới mối liên kết*):

- Có default constructor
- Khai báo @Entity và @Id
- Serializable (không required)

### Thuộc tính đa trị

- Cơ sở dữ liệu quan hệ sẽ tách ra thành một bảng mới, khóa chính của bảng mới là khóa chính của bảng ban đầu và thuộc tính đa trị.
- Khai báo @ElementCollection cho thuộc tính đa trị.
- Dùng @CollectionTable để mô tả thông tin cho bảng mới sẽ được tách ra.

Ví dụ: phone = ["123456789", "012345678"]

```
@ElementCollection
@CollectionTable(name="phones", joinColumns = @JoinColumn(name="student_id"))
@Column(name="phone", nullable = false)
protected Set<String> phones;
```

### Entity associations/ relationships

#### 1. Mối quan hệ 1 - 1 (One to One)

Nếu xét một chiều, đi từ phía phụ thuộc sang chủ thể.

**Trường hợp 1:** Khóa chính bên bảng phụ thuộc chính là khóa chính bên chủ thể.

Ví dụ:

```
@Entity
@Table(name = "classe_profiles")
public class ClazzProfile {
    @Id
    @OneToOne
    @JoinColumn(name = "class_id")
    private Clazz clazz;
    private LocalDate createDate;
    private String description;
}
```

**Trường hợp 2:** Bảng phía phụ thuộc có khóa chính riêng, khóa ngoại trong bảng này phải đồng thời là unique.

Ví dụ:

```
@Entity
@Table(name = "classe_profiles")
public class ClazzProfile {
    @Id
    @Column(name = "profile_id")
    private String id;

    @OneToOne
    @JoinColumn(name = "class_id", unique = true, nullable = false)
    private Clazz clazz;

    private LocalDate createDate;
    private String description;
}
```

## 2. Mối quan hệ 1 – n (*One to Many hoặc Many to One*)

- Phía One:
  - Thêm @OneToMany với thuộc tính mappedBy.
- Phía Many thêm:
  - @ManyToOne
  - @JoinColumn với thuộc tính name

*Nếu xét một hướng, thì hướng đi từ many to one*

Ví dụ:

```
@Entity
@Table(name = "students")
public class Student {
    @Id
    @Column(name="student_id")
    protected String id;
    @Column(columnDefinition = "nvarchar(100)", nullable = false)
    protected String name;
    @Column(unique = true, nullable = false)
    protected String email;
    protected LocalDate dob;

    @ManyToOne(fetch = FetchType.LAZY)
    protected Clazz clazz;
}
```

## 3. Mối quan hệ n – n (*Many to Many*)

Nguyên tắc đối với mối quan hệ Many to Many, cơ sở dữ liệu quan hệ sẽ tách thêm một thực thể mới (2 mối quan hệ *One to Many*).

Khóa của thực thể kết hợp chứa ít nhất 2 khóa của 2 thực thể tham gia vào mối liên kết này. Phải có equals và hashCode trên tất cả các thuộc tính tham gia vào khóa.

Ví dụ:

```
@Entity
@Table(name = "enrollments")
public class Enrollment {
    @Id
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "student_id")
    private Student student;

    @Id
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "course_id")
    private Course course;
    @Id
    private String semester;
    @Id
    private int year;
    private int score;
}
```

#### 4. Mối quan hệ n – n (không có thuộc tính của mối liên kết)

- Cả 2 bên điều khai báo @ManyToMany
- Bên chủ thể với thuộc tính mappedBy
- Bên còn lại thêm @JoinTable, với các thuộc tính:
  - name = "Table\_Name",
  - joinColumns = @JoinColumn(name = "Column\_name"),
  - inverseJoinColumns = @JoinColumn(name = "Column\_name")

#### 5. Mối quan hệ n – n (viết lớp mô tả khóa riêng)

– Lớp thực thể mới tách thêm:

- Khai báo @Entity và @IdClass
- Id của lớp là một Composite key.
  - Gồm ít nhất 2 field là khóa của 2 lớp thực thể tham gia vào mối liên kết.
  - Mỗi field khai báo @Id, @ManyToOne và @JoinColumn
- Có thể có thêm các thuộc tính của mối liên kết

– Lớp mô tả composite key: các thuộc tính của lớp này là các thuộc tính tham gia vào khóa. Composite Primary key còn được gọi tắt là *composite key* – chứa 2 hoặc nhiều hơn các thuộc tính trong *primary key* cho một table trong database.



Có 2 cách để định nghĩa một *composite key* với `@IdClass` và `@EmbeddedId` annotation.

Trước khi định nghĩa một *composite key*, chú ý một số quy định sau:

- Access modifiers composite primary key class phải là **public**.
- Nó phải có một no-arg constructor.
- Phải định nghĩa `equal()` and `hashCode()` method.
- Cuối cùng là phải implement `Serializable`

### **@IdClass annotation**

```
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode
public class AccountPK implements Serializable {
    private static final long serialVersionUID = 1L;
    private String accountNumber;
    private String accountType;
}
@Getter
@Setter
@AllArgsConstructor
@Entity
@IdClass(AccountPK.class)
public class Account {
    @Id
    private String accountNumber;
    @Id
    private String accountType;
    private String name;
}
```

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa-
demo");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Account account = new Account("842582", "CREDIT", "ACB");
em.persist(account);
em.getTransaction().commit();
em.close();
emf.close();
```

## @EmbeddedId annotation

```
@Embeddable
@EqualsAndHashCode
@NoArgsConstructor
@AllArgsConstructor
public class BookPK implements Serializable {
    private static final long serialVersionUID = 1L;
    private String title;
    private String language;
}

@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Book {
    @EmbeddedId
    private BookPK bookId;
    private String author;
}

EntityManagerFactory emf =
Persistence.createEntityManagerFactory("jpa-demo");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
BookPK bookPK = new BookPK("BOOK_001", "VN");
Book book = new Book(bookPK, "Book 001");
em.persist(book);
em.getTransaction().commit();
em.close();
emf.close();
```

Đối với **@IdClass** khi muốn truy vấn sử dụng một trong số các thuộc tính của composite key:

```
SELECT account.accountNumber FROM Account account
```

*@EmbeddedId* : phải thông qua *BookPK* class

```
SELECT book.bookId.title FROM Book book
```

## 6. Mối quan hệ kế thừa (*Inheritance relationship*)

Có 3 chiến lược để ánh xạ:

- InheritanceType.**SINGLE\_TABLE** → class parent là 1 entity
  - Là một trong những cách đơn giản và hiệu quả nhất để xác định việc triển khai kế thừa. Theo cách tiếp cận này, các instance của nhiều entities chỉ được lưu trữ dưới dạng thuộc tính trong một bảng duy nhất.
- InheritanceType.**JOINED** → mỗi class là 1 table, yêu cầu JOIN table
- InheritanceType.**TABLE\_PER\_CLASS** → mỗi class là 1 table riêng với tất cả thuộc tính
  - đối với mỗi sub entity class → tạo 1 table riêng. Không giống như InheritanceType.**JOINED**, không có bảng riêng biệt nào được tạo cho lớp thực thể cha trong chiến lược mỗi bảng.

Ví dụ:

```
@Entity
@Table(name = "students")
@Inheritance(strategy = InheritanceType.JOINED)
Public abstract class Student {
    @Id
    @Column(name="student_id")
    protected String id;
    @Column(columnDefinition = "nvarchar(100)", nullable = false)
    protected String name;
    @Column(unique = true, nullable = false)
    protected String email;
    protected LocalDate dob;
}

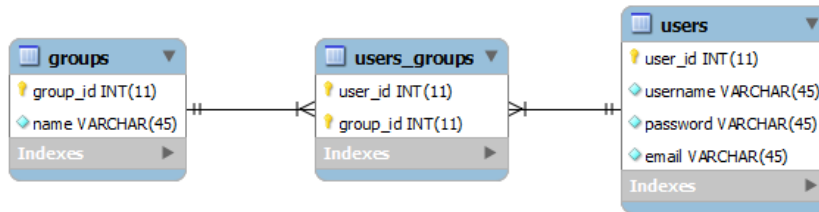
@Entity
@Table(name = "part_time_students")
public class PartTimeStudent extends Student {
    private String supervisor;
}

@Entity
@Table(name = "full_time_students")
public class FullTimeStudent extends Student{
    private String faculty;
    private String department;
}
```

## Bài tập thực hành

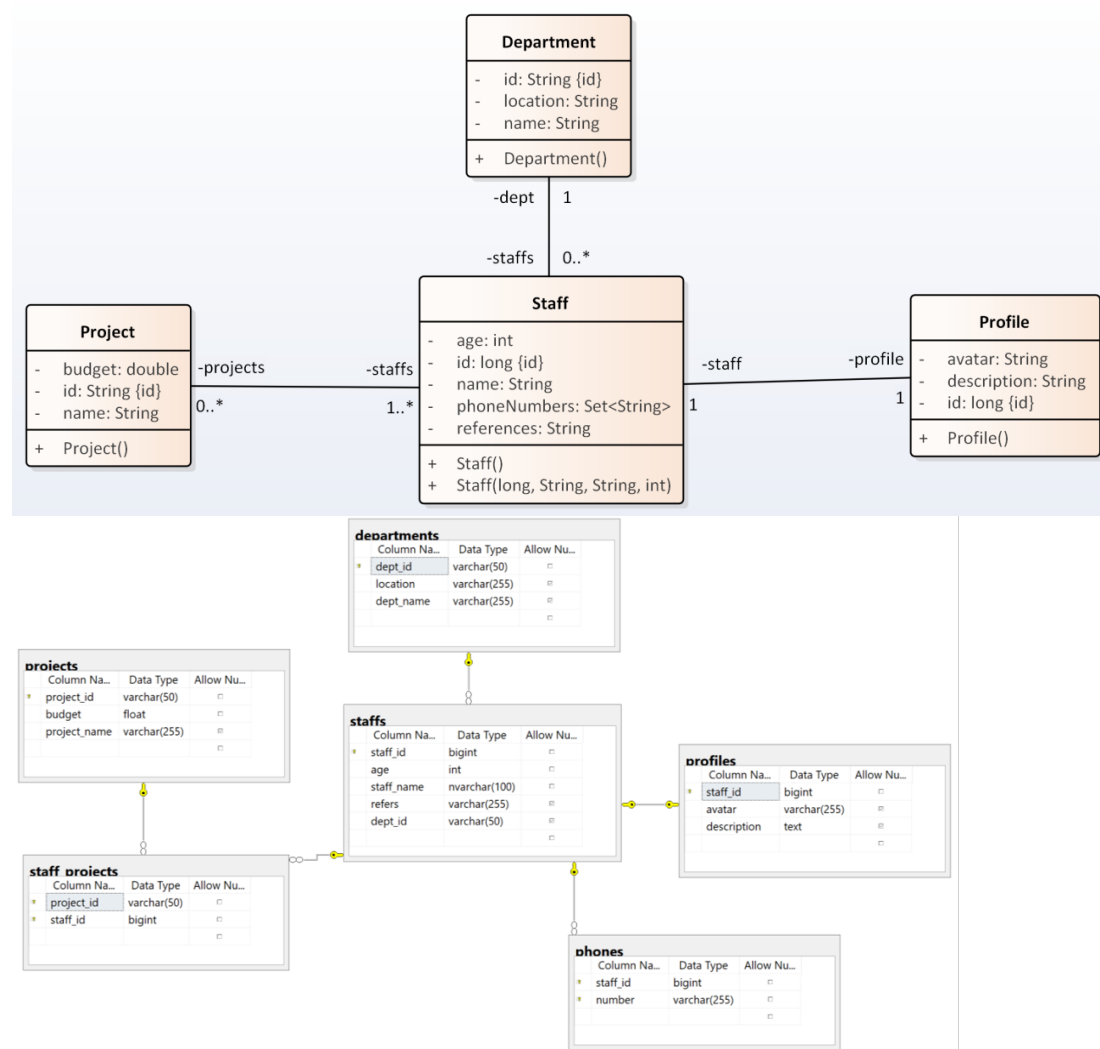
### Bài tập 1

Viết các lớp persistence entity, sau cho ánh xạ ra được mô hình CSDL quan hệ sau:



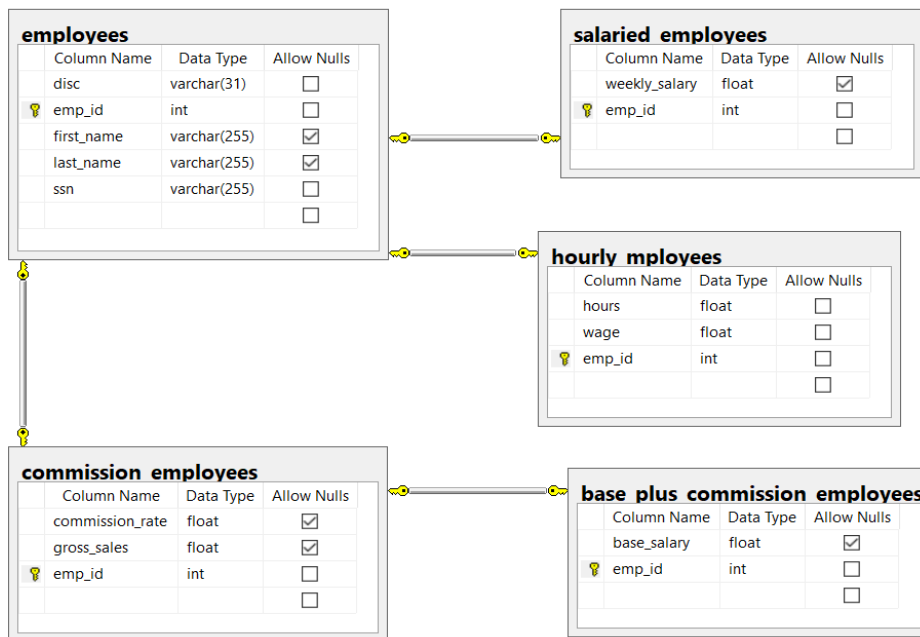
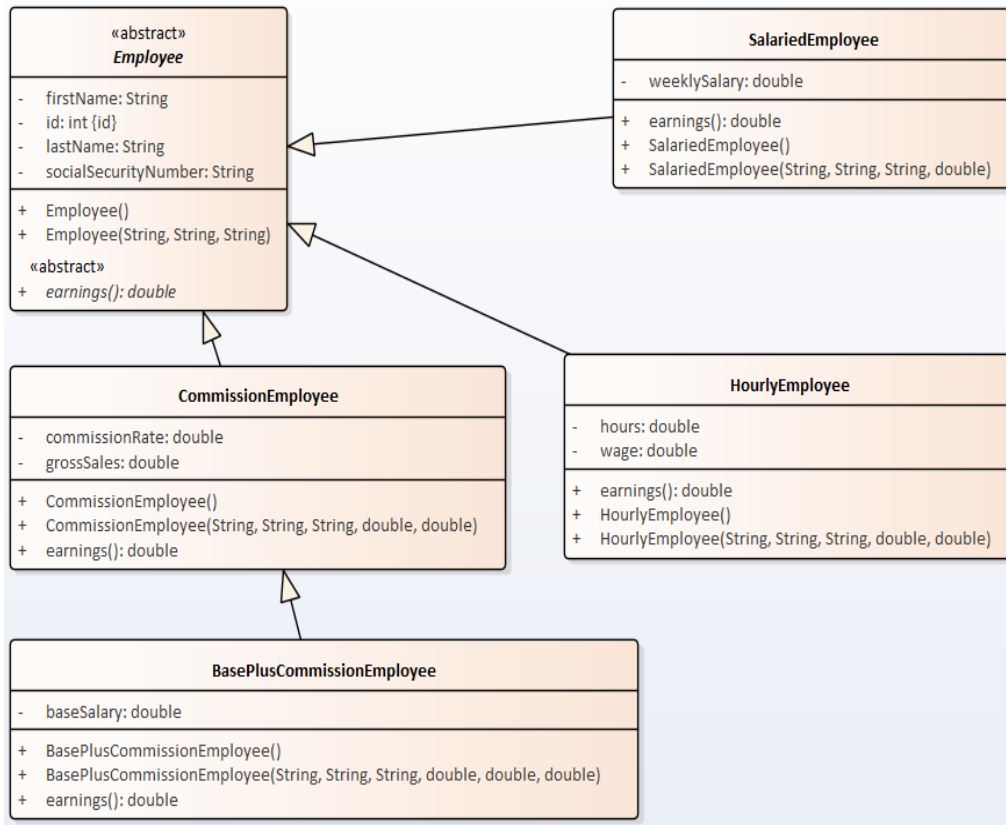
### Bài tập 2

Thuộc tính đa trị, mối quan hệ 1 – 1 cùng id, mối quan hệ 1 – n và mối quan hệ n – n

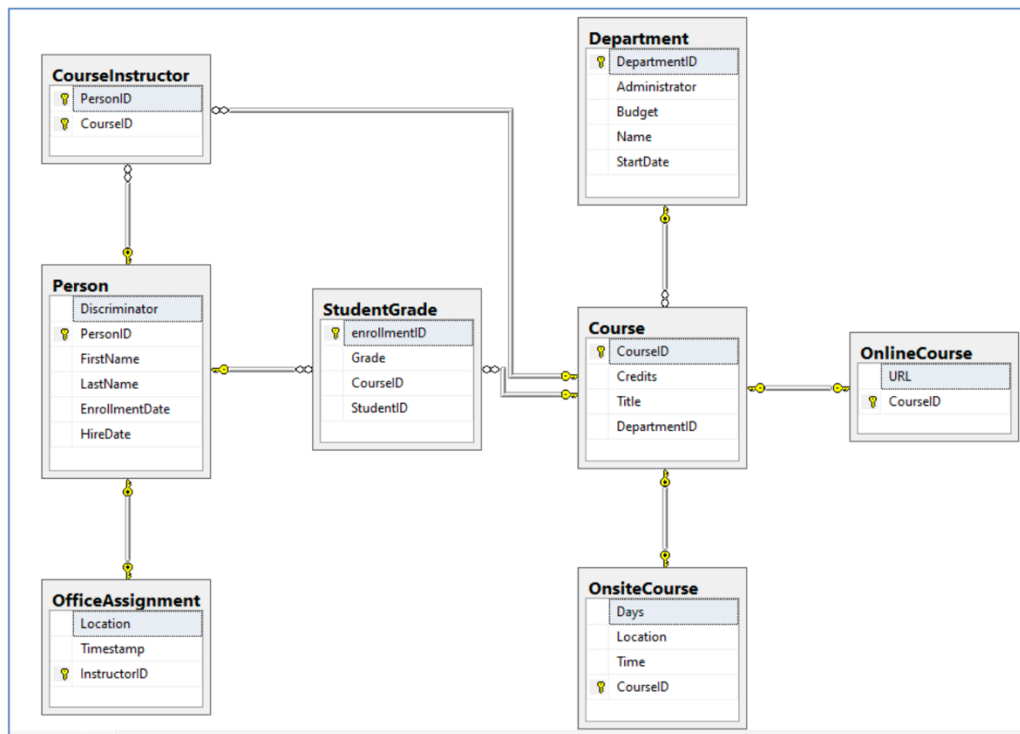
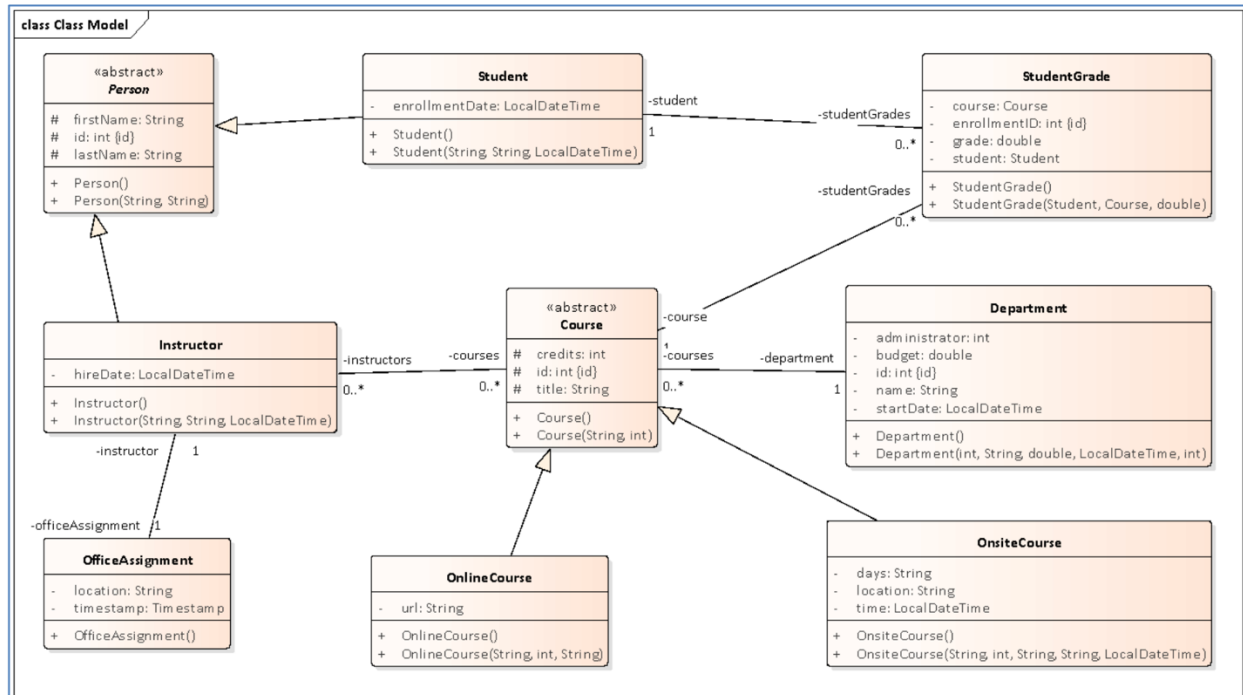


## Bài tập 3

### Mối quan hệ kế thừa



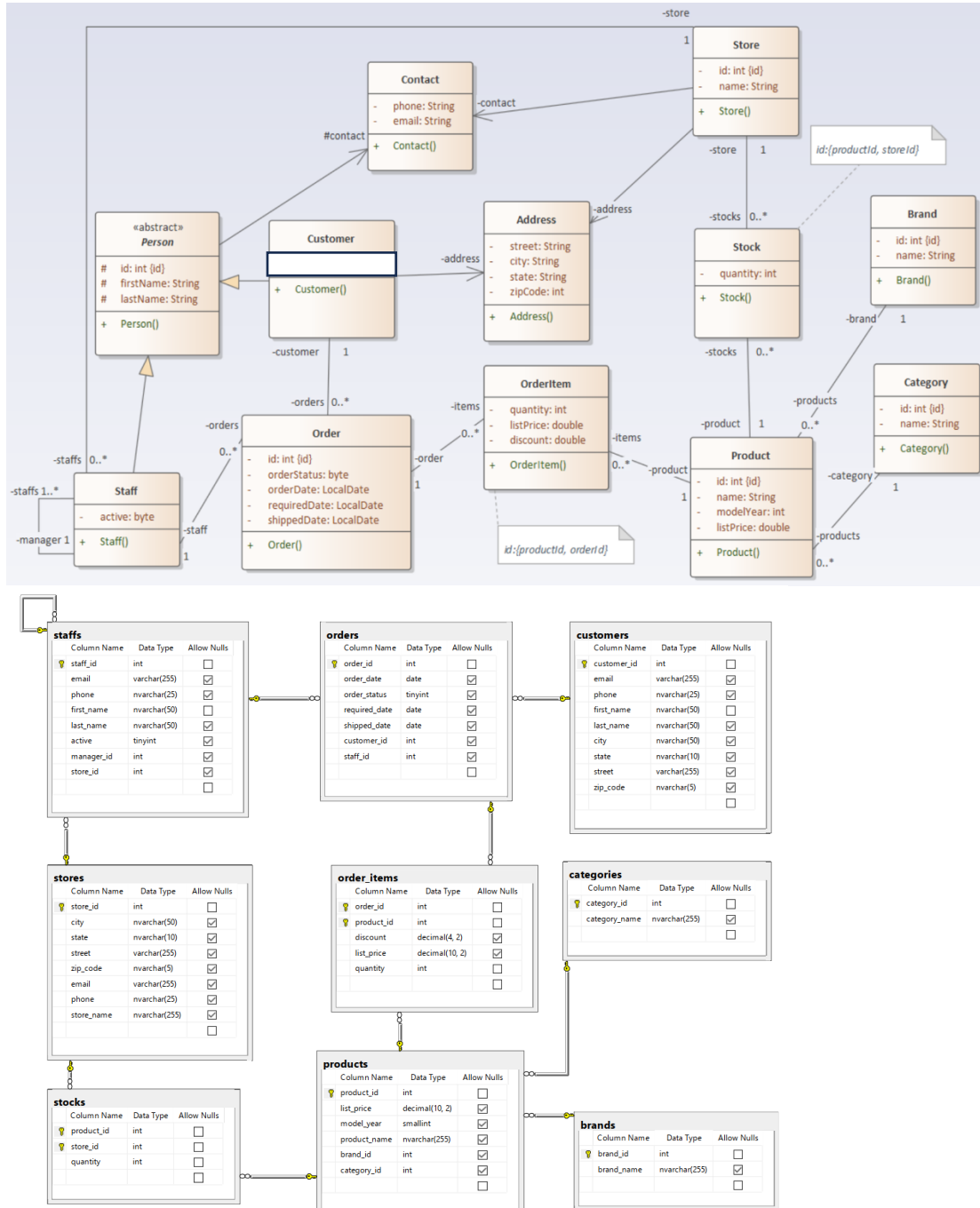
## Bài tập 4



- Ánh xạ từ mô hình lớp sang mô hình CSDL quan hệ trên
- Nhập dữ liệu mẫu
- Thực hiện các thao tác CRUD và tổng hợp dữ liệu bằng ngôn ngữ lập trình Java

## Bài tập 5

A) Viết các lớp persistence entity, sau cho ánh xạ ra được mô hình CSDL quan hệ tương ứng:



B) Load dữ liệu database (MS SQL Server)

<https://www.mediafire.com/file/lqjpf4kvrwh4yt/BikeStores.rar>

C) (Java Programming) Viết các câu truy vấn sau:

1. Thực hiện các thao tác Create, Update, Delete, Find by Id, Get All trên từng đối tượng.
2. Tìm danh sách sản phẩm có giá cao nhất.
3. Tìm danh sách sản phẩm chưa bán được lần nào.
4. Thống kê số khách hàng theo từng bang.  
+ `getNumberCustomerByState() : Map<String, Integer>`
5. Tính tổng tiền của đơn hàng khi biết mã số đơn hàng.
6. Đếm số đơn hàng của từng khách hàng.  
+ `getOrdersByCustomers() : Map<Customer, Integer>`
7. Tính tổng số lượng của từng sản phẩm đã bán ra.  
+ `getTotalProduct(): Map<Product, Integer>`
8. Tính tổng tiền của tất cả các hóa đơn trong một ngày nào đó.
9. Xóa tất cả các khách hàng chưa mua hàng.
10. Thống kê tổng tiền hóa đơn theo tháng / năm.