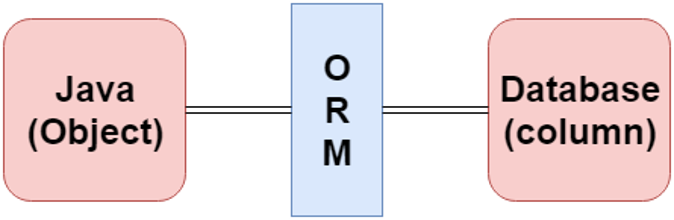
# Java Persistence API

## Object Relational Mapping (ORM)

### JPA Object Relational Mapping

ORM giúp làm việc với database dễ dàng hơn trong những ứng dụng hướng đối tượng, cho phép ta tập trung vào business logic thay vì các tiểu tiết lưu trữ dữ liệu nằm ở bên dưới.

* **Object** - thuộc thế giới hướng đối tượng
* **Relational** - thuộc thế giới database (relation chính là table)
* **Mapping** - ánh xạ giữa 2 thế thới: hướng đối tượng và database(table, column,…)



### Những lợi ích của ORM

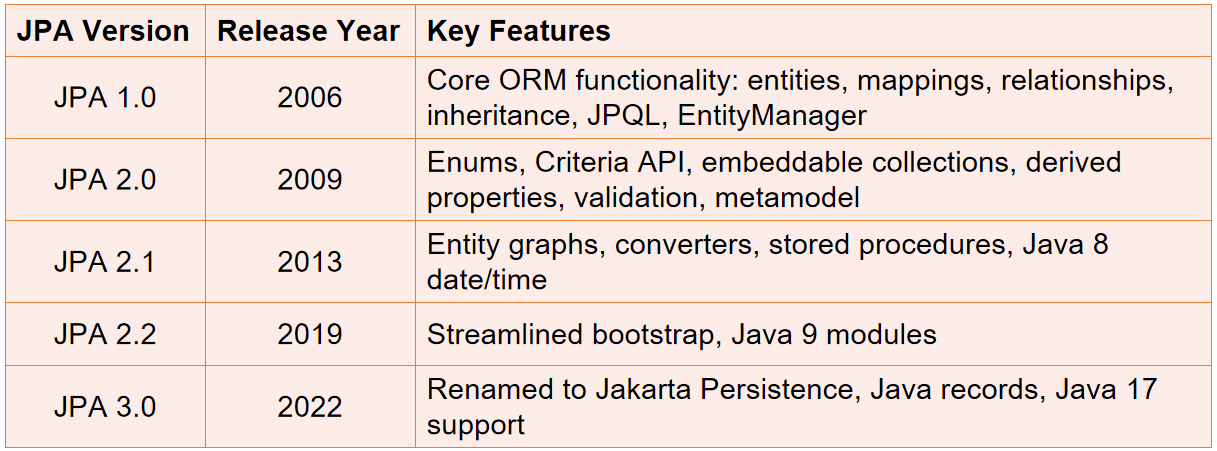
* **Tăng năng suất**: Giúp ta viết ít code hơn bới vì ORM đã giúp ta xử lý các **low-level data access tasks**.
* **Cải thiện khả năng bảo trì**: Code của ta sẽ clean hơn và dễ hiểu hơn bởi vì nó tập trung vào business logic không phải database details.
* **Nâng cao hiệu năng**: Những ORM frameworks có thể tối ưu hóa data access và caching, có khả năng cải thiện hiệu suất.
* **Không phụ thuộc vào database**: Có thể dễ dàng chuyển đổi những database mà không cần viết lại code.

### Những ORM Frameworks phổ biến cho Java

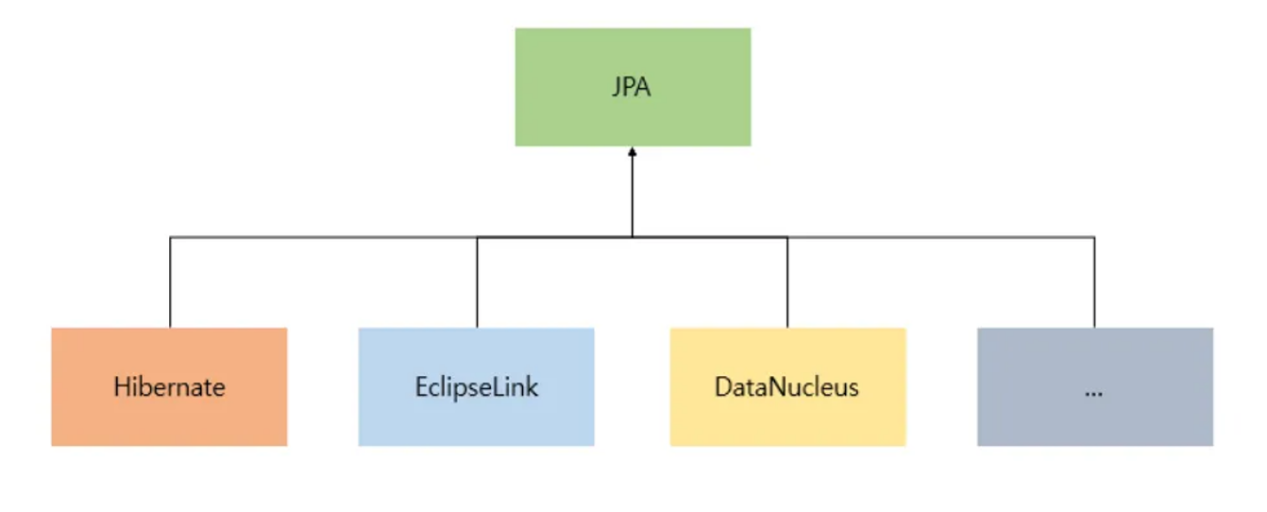
* **Hibernate**: Là **JPA implementation** được sử dụng rộng rãi nhất, mang đến một tập hợp các tính năng giàu giá trị và những tùy chọn tùy chỉnh rộng rãi.
* **EclipseLink**: Là một **JPA implementation** khác mà tập trung vào sự tuân thủ các tiêu chuẩn và khả năng di chuyển.
* **Spring Data JPA**: Một lớp trừu tượng nằm trên cùng những JPA providers, đơn giản hóa data access và cắt giảm những **boilerplate code**.

## Java Persistence API (JPA)

### JPA Versions and Key Features



### JPA Implementation Options



### JPA là gì?

* JPA là viết tắt của **Java Persistence API**. Nó là một **Java programming interface**.
* JPA cung cấp một tập các interface, annotations, … làm đơn giản hóa quá trình tương tác với database.

### Tính năng chính của JPA

* **ORM**: JPA cho phép ta ánh xạ Java objects (entities) với những tables trong CSDL quan hệ.
* **Annotations**: Ta có thể sử dụng những annotations để định nghĩa các entity, các mối quan hệ,…
* **EntityManager**: Interface này cung cấp những phương thức CRUD (create, read, update, delete) trên một entity.
* **JPQL**: Là một ngôn ngữ truy vấn hướng đối tượng (tương tự như SQL nhưng thiên về hướng đối tượng) cho phép ta truy vẫn những entitis và các mối quan hệ của chúng.
* **Transaction Management**: JPA hỗ trợ quản lý transaction, đảm bảo tính toàn vẹn và tính nhất quán về dữ liệu.

### Những lợi ích của việc sử dụng JPA

* **Simplified Data Access**: JPA giúp ta làm việc với dữ liệu quan hệ dễ dàng hơn trong những ứng dụng Java bằng cách cung cấp một higher-level abstraction.
* **Code Portability**: Code sử dụng JPA có thể dễ dàng chuyển sang các JPA providers khác nhau.
* **Reduced Boilerplate Code**: JPA annotations và the EntityManager interface làm giảm một lượng code cần cho truy cập data.
* **Improved Data Integrity**: JPA's transaction management và validation features giúp đảm bảo tính nguyên vẹn dữ liệu.
* **Enhanced Performance**: Caching và những kỹ thuật tối ưu hóa khác có thể cải thiện hiệu năng của các ứng dụng sử dụng JPA.

### Entity Annotations trong JPA

* **@Entity**: Annotation này đánh dấu một Java class như là một entity, có nghĩa là nó đại diện cho một table trong database.
* **@Table**: Annotation này cho phép ta chỉ rõ tên của table mà một entity ánh xạ đến.
* **@Transient**: Annotation này đánh dấu một thuộc tính mà không nên được persist xuống database.
* **@NamedQueries** và **@NamedQuery**: 2 annotation này cho phép ta định nghĩa named queries mà có thể được sử dụng để retrieve entities.

### Mapping Annotations trong JPA

* **@Id**: Đánh dấu một thuộc tính tương ứng với khóa chính của bảng trong database.
* **@Column**: Định nghĩa ánh xạ giữa một thuộc tính của entity với một cột trong table. Ta có thể sử dụng nó để chỉ định tên cột, kiểu dữ liệu của cột,…
* **@Basic**: Chỉ định một thuộc tính là một basic type và nên được persist.
* **@Enumerated**: Ánh xạ một kiểu Enum đến một cột trong database.

### JPA Entity Manager

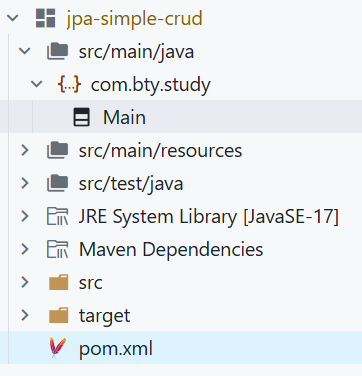
* **Manages entity lifecycle**: Persist, find, merge, remove
* **Controls persistence context**: Cache, flush, detach
* **Executes queries**: JPQL, Criteria API
* **Handles transactions**: Begin, commit, rollback
* **Provides access to entity metadata**

### Key Operations của Entity Manager

* **Persisting Entities**: em.persist(entity) lưu enity xuống database.
* **Finding Entities**: em.find(entityClass,primaryKey) lấy một entity dựa vào khóa chính của nó.
* **Merging Entities**: em.merge(entity) hợp nhất detached entity instance vào current persistence context.
* **Removing Entities**: em.remove(entity) xóa persistent entity instance.
* **Querying Data**: em.createQuery(jpqlString) tạo JPQL query để lấy entities dựa vào tiêu chuẩn được chỉ định.

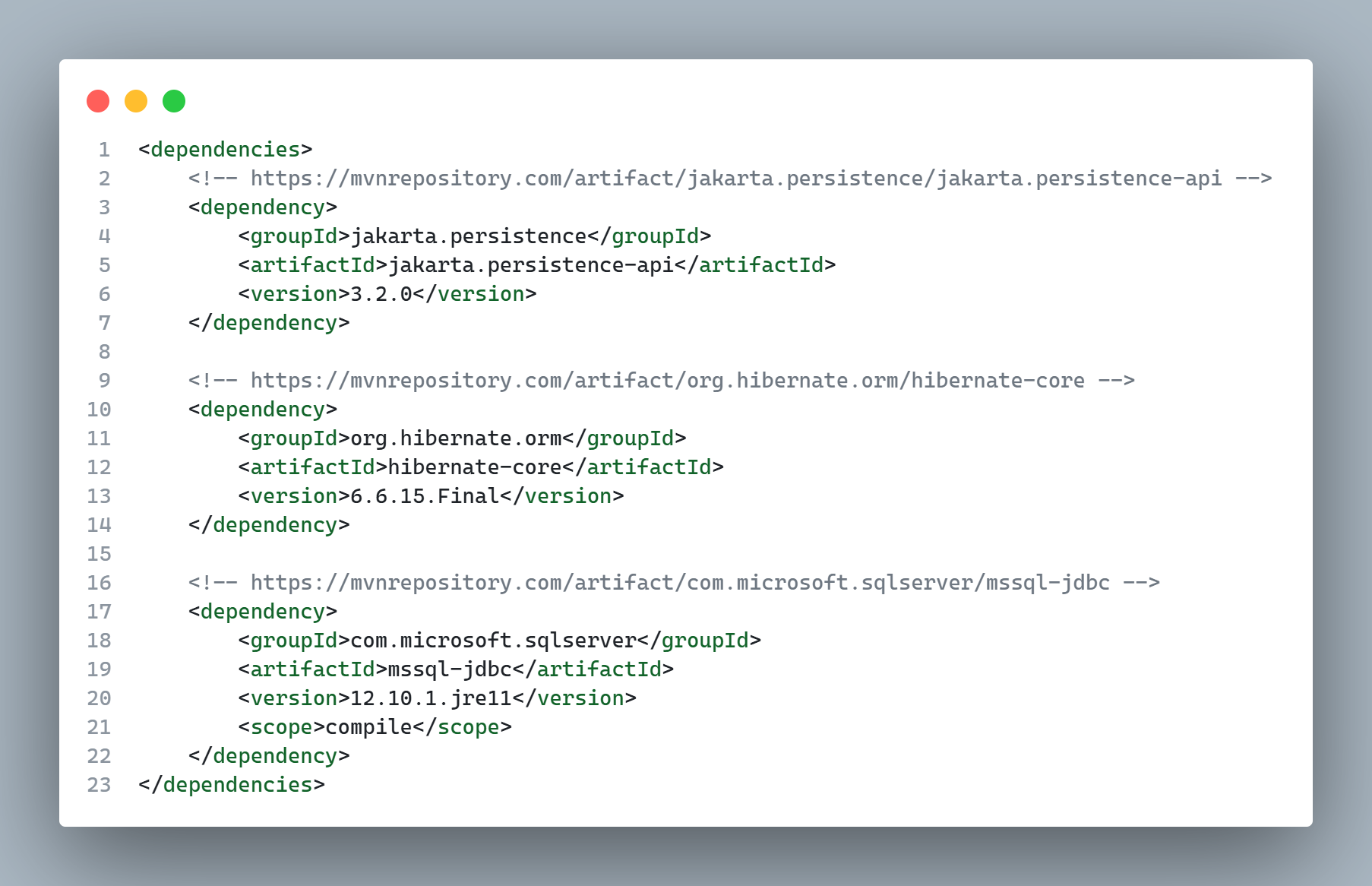
## Demo Simple CRUD với JPA

Tạo Maven project - **jpa-simple-crud**

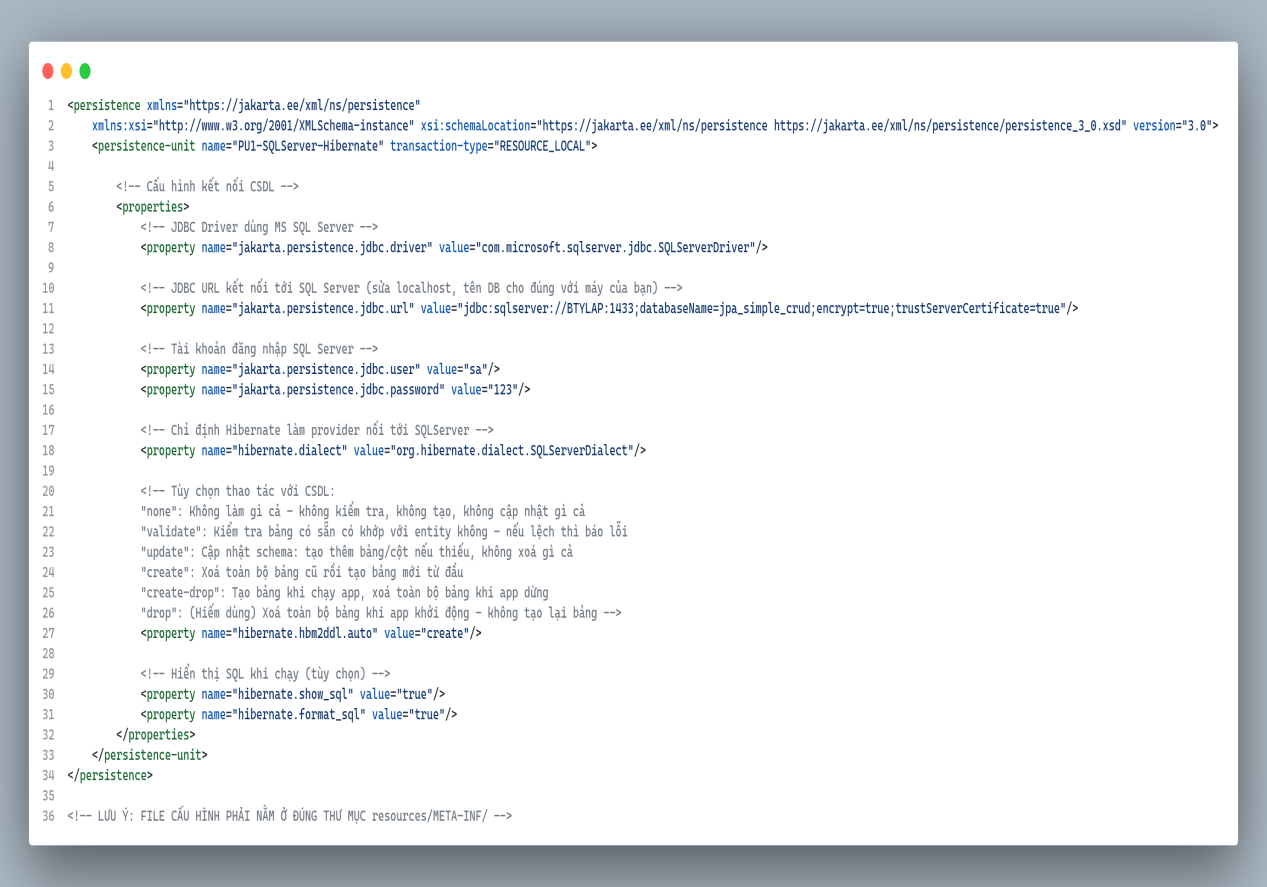


Thêm các dependencies vào file **pom.xml**:

* JPA
* Hibernate
* SQL Server driver



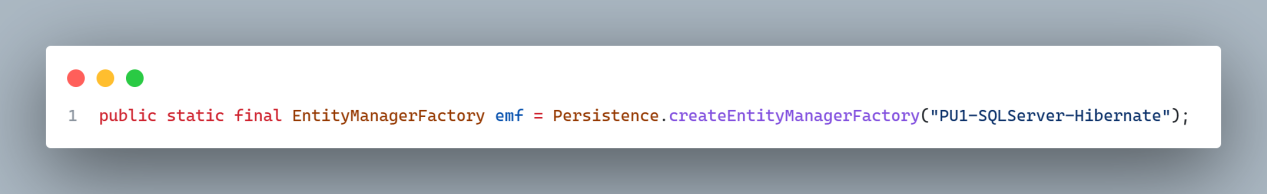
Tạo file ***persistence.xml*** trong **resources/META-INF/** với nội dung:



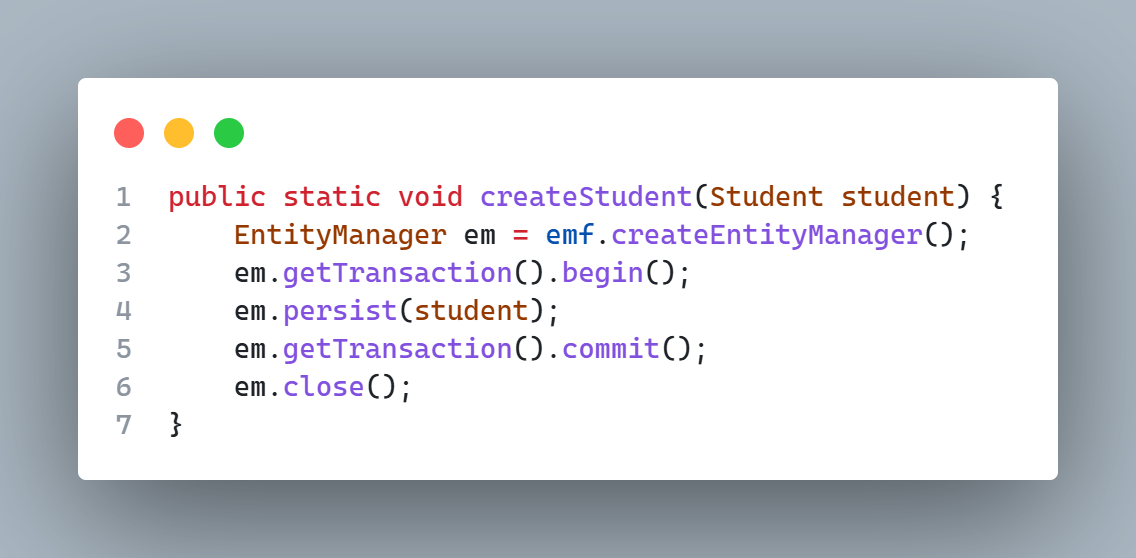
Tạo lớp **Student** là entity để map xuống table trong database:



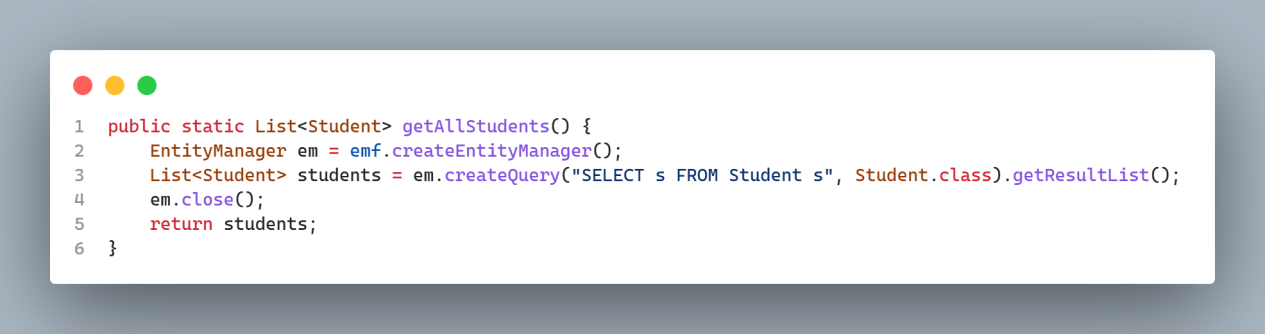
Tạo đối tượng **EntityManagerFactory emf** kết nối đến CSDL:



Tạo hàm **createStudent** - thêm một sinh viên vào CSDL:



Tạo hàm **getAllStudents** - lấy tất cả sinh viên từ CSDL:



Tạo hàm **main** thực hiện thêm sinh viên và lấy tất cả sinh viên:



**Kết quả**:

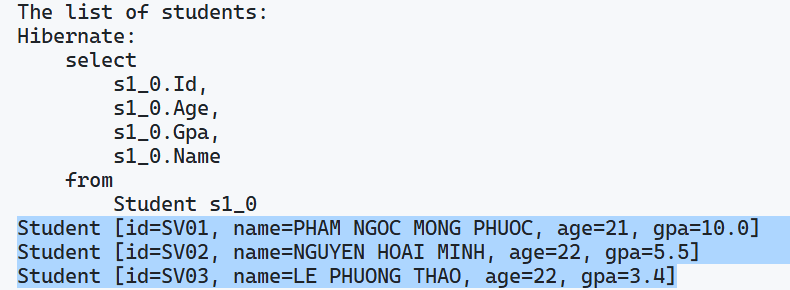
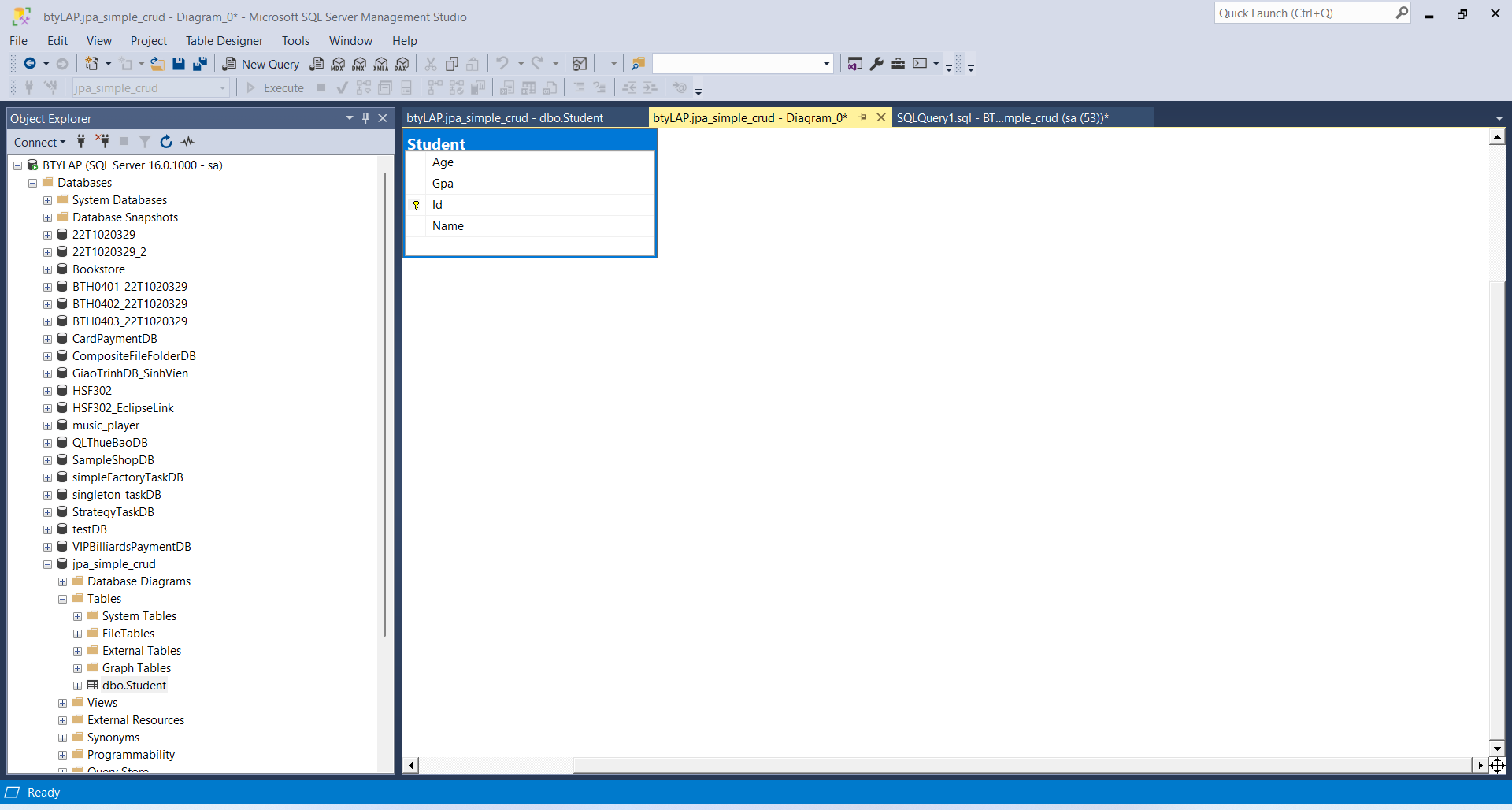
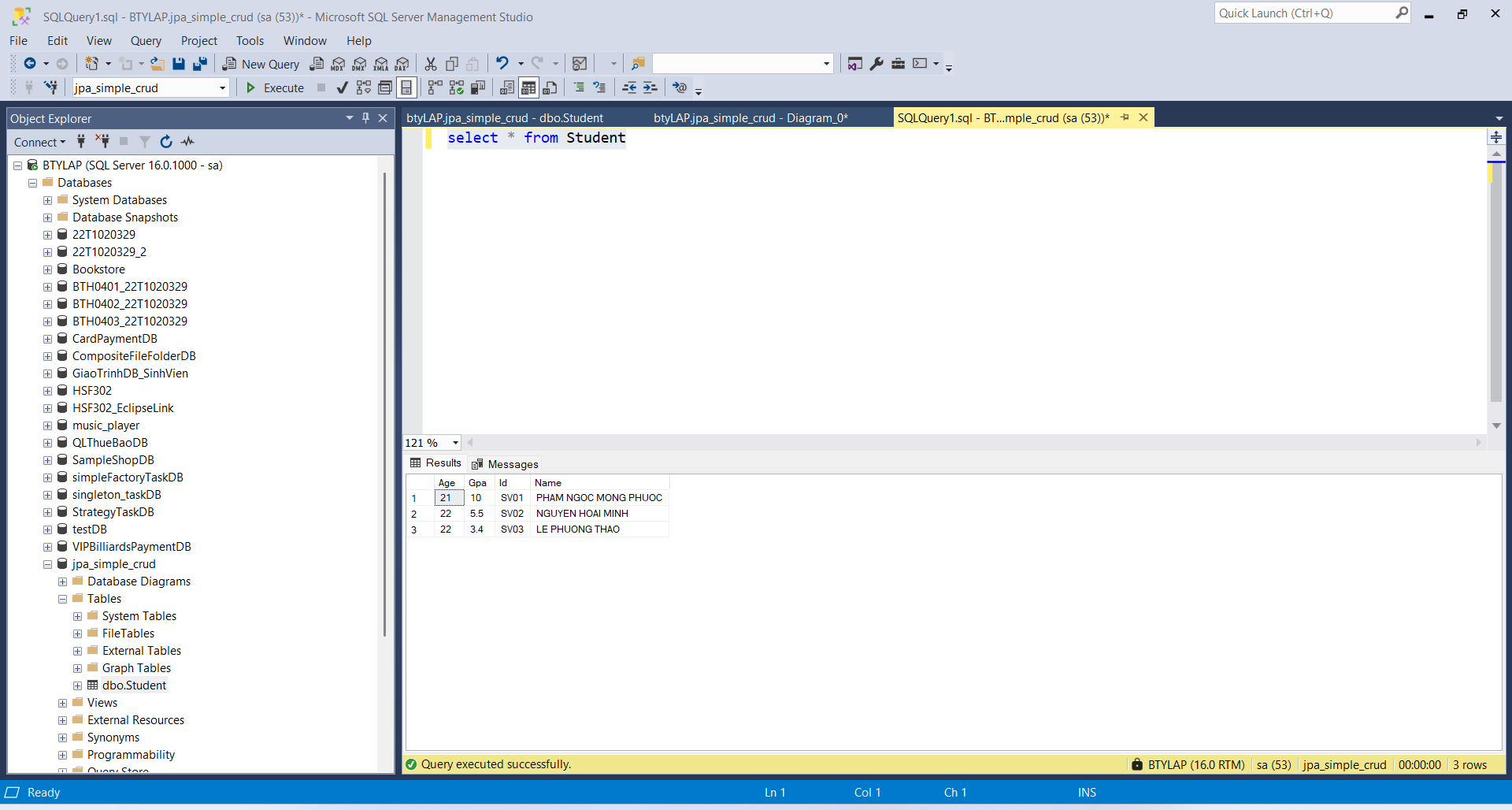


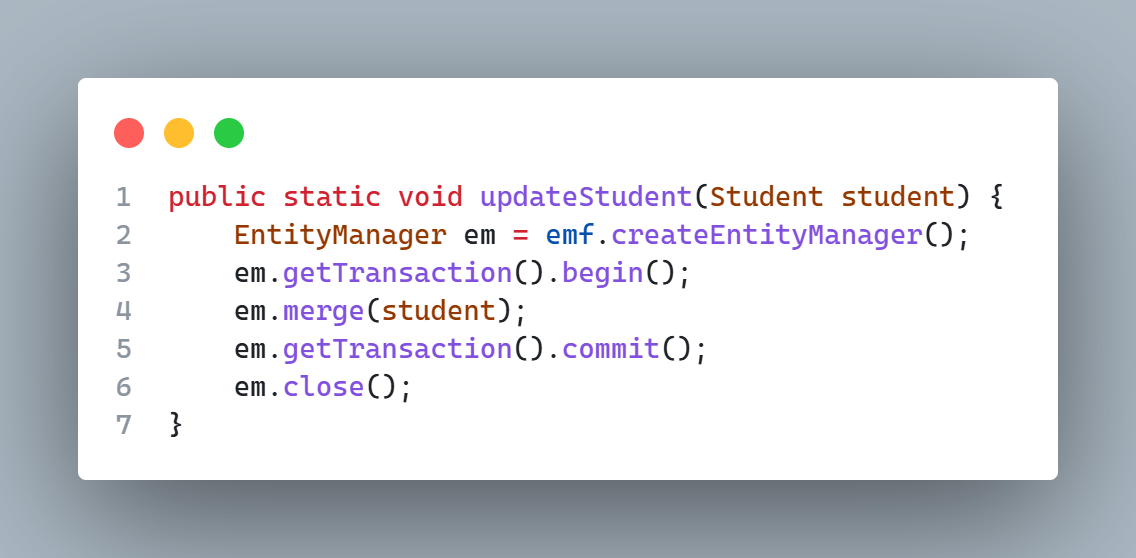
Table **Student** được tự động tạo ra:



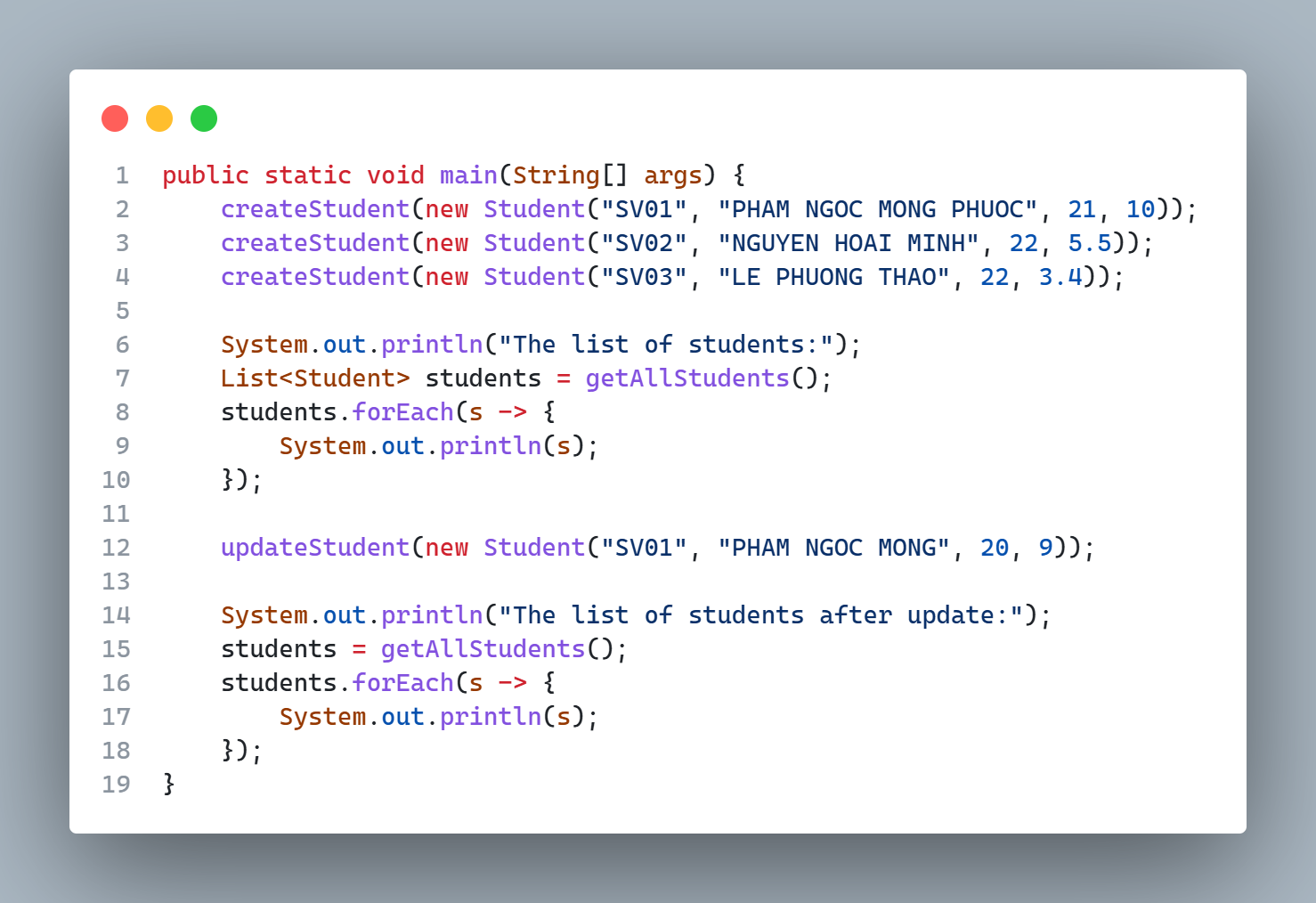
Dữ liệu đã được thêm vào:



Tạo hàm **updateStudent** - cập nhật một sinh viên:

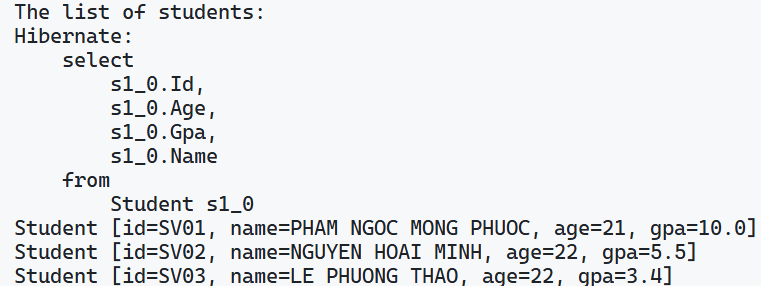


Gọi hàm **updateStudent** để cập nhật một student trong hàm main():

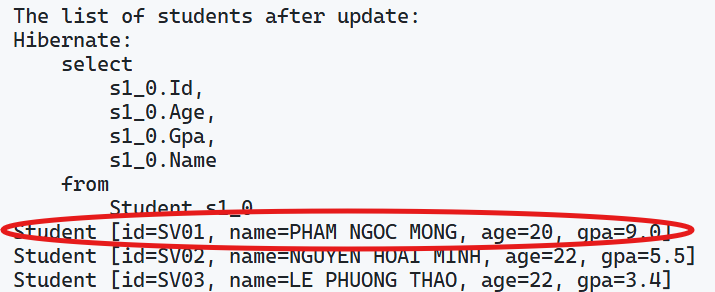


**Kết quả**:

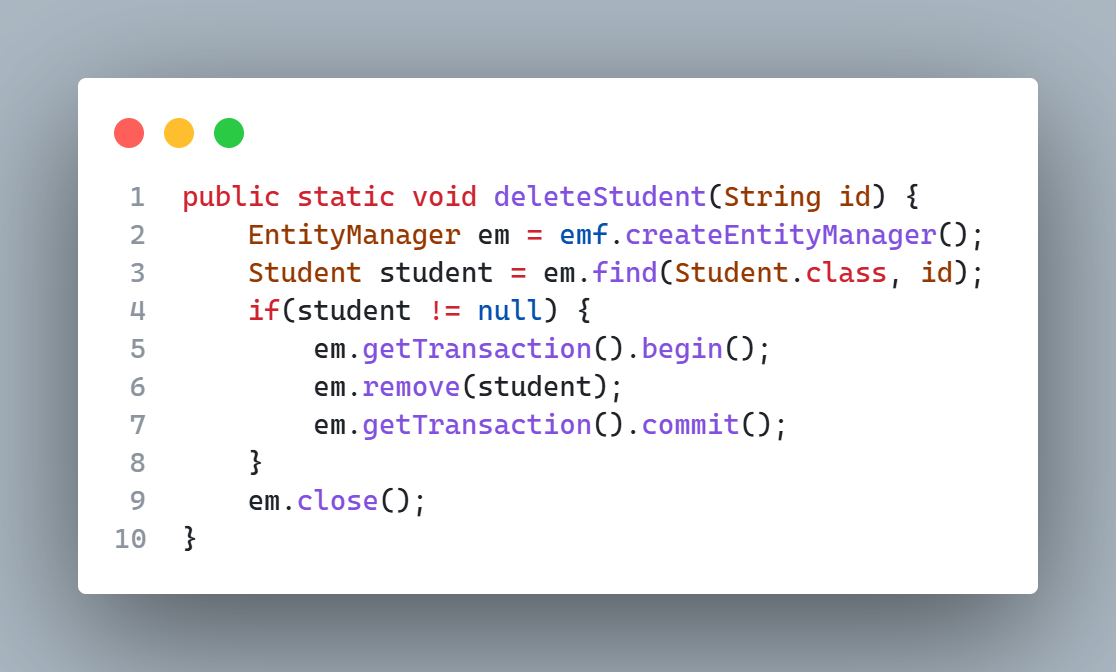
- Trước khi update:



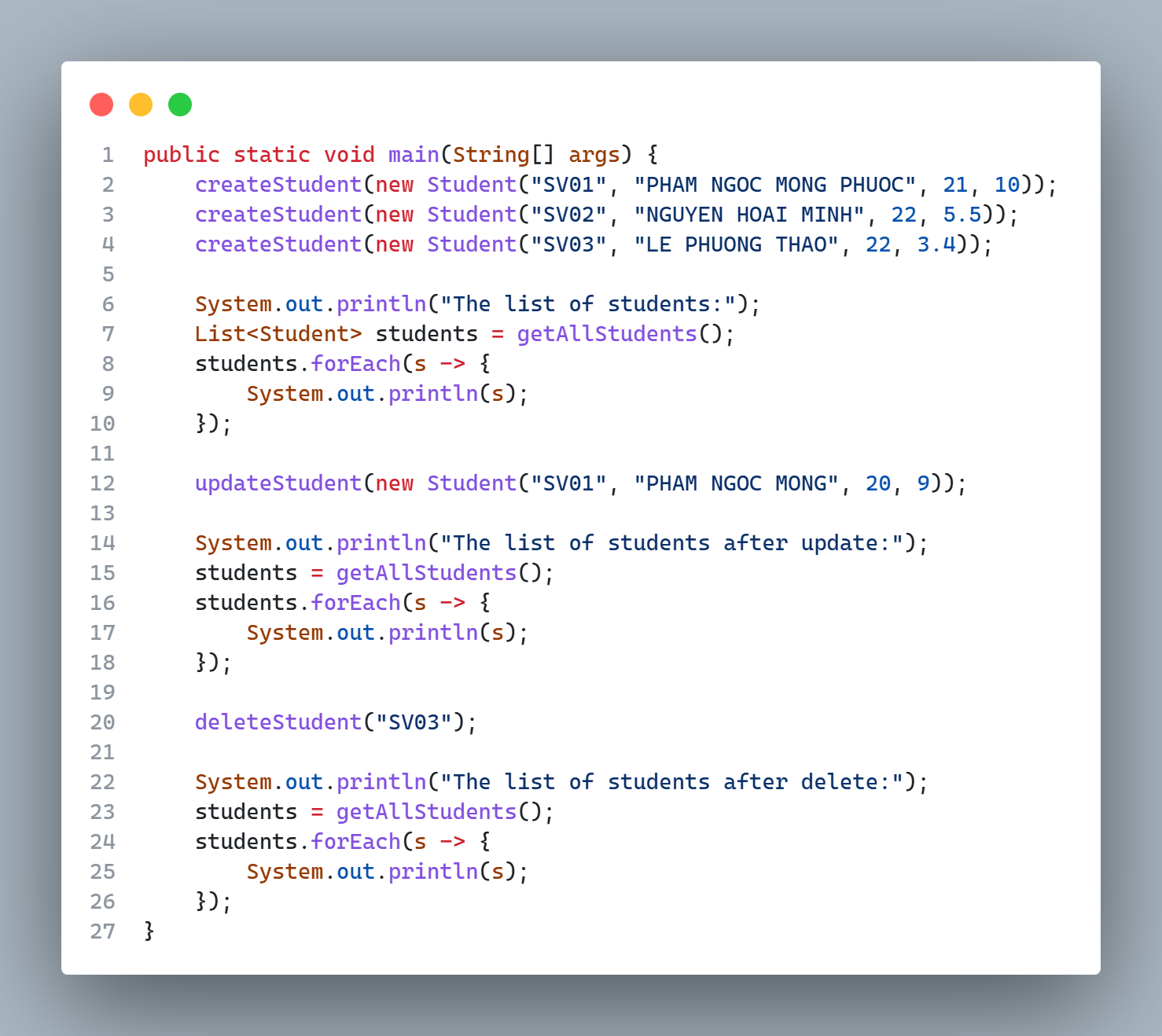
- Sau khi update:



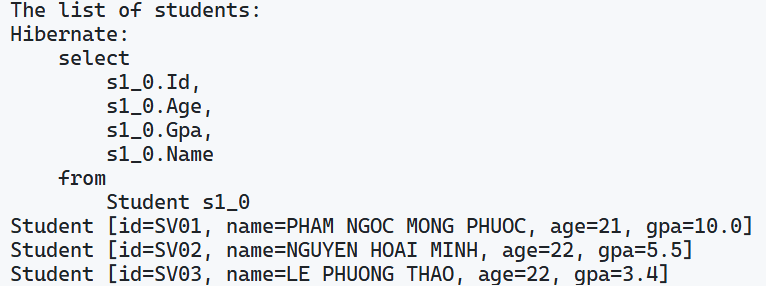
Tạo hàm **deleteStudent** - xóa một sinh viên trong CSDL:

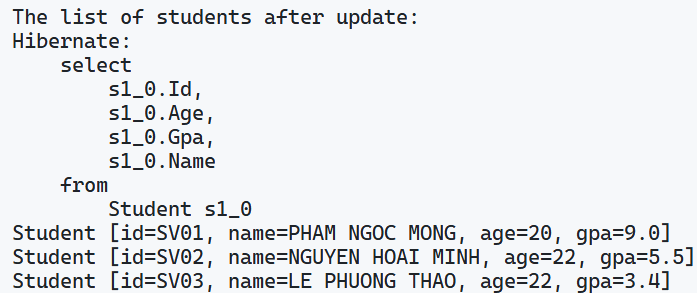


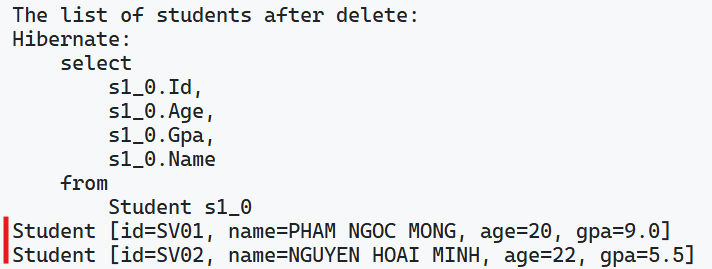
Gọi hàm **deleteStudent** để xóa một sinh viên trong hàm main():



**Kết quả**:







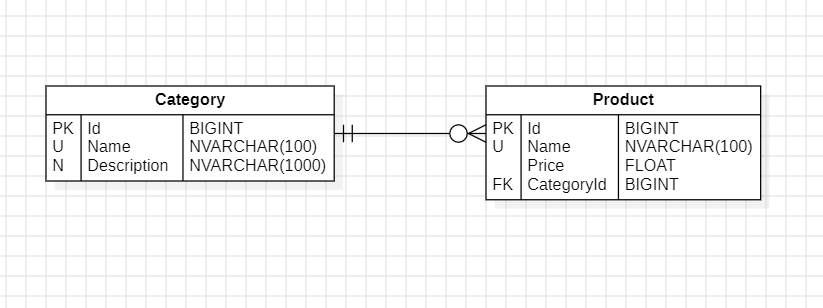
## Mapping trong JPA

Relationships Annotations trong JPA:

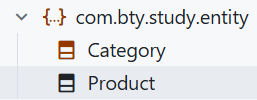
* **@ManyToOne**: Định nghĩa mối quan hệ **n-1** giữa 2 entity.
* **@OneToMany**: Định nghĩa mối quan hệ **1-n** giữa 2 entity.
* **@OneToOne**: Định nghĩa mối quan hệ **1-1** giữa 2 entity.
* **@ManyToMany**: Định nghĩa mối quan hệ **n-n** giữa 2 entity.

## Demo JPA (One To Many)

Ta thiết kế 2 bảng **Category** và **Product** với cấu trúc như sau:



Tạo 2 entity class: **Category** và **Product**



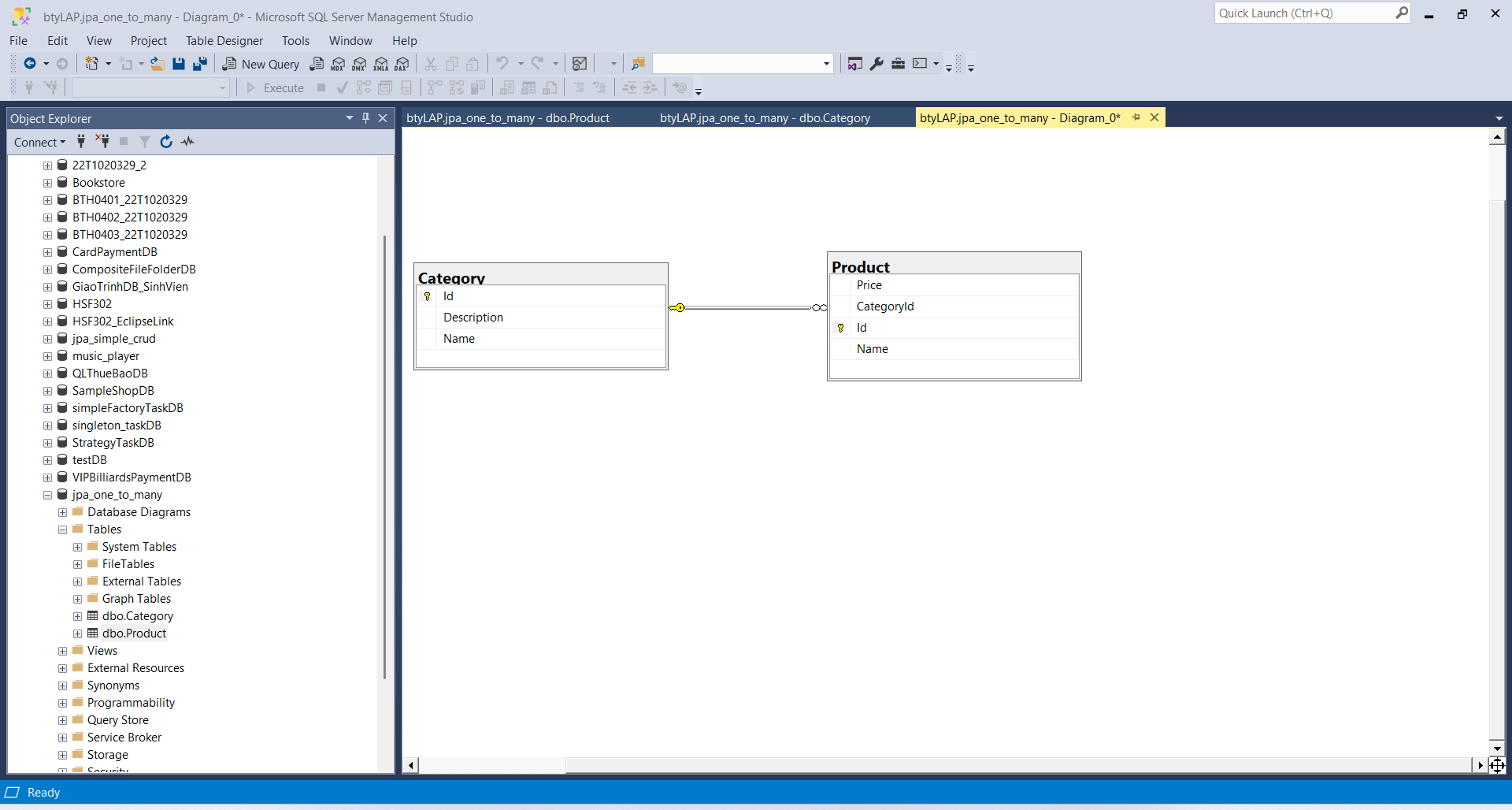
Sử dụng **@ManyToOne** để thiết lập mối quan hệ trong class **Product**:



Sử dụng **@OneToMany** để thiết lập mối quan hệ trong class **Category**:



**Kết quả**:



- 2 bảng được tự động tạo ra từ 2 class ở trên

- Bảng Category có quan hệ 1 - n với bảng Product

## Tổng kết

