

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



**BÀI TIỂU LUẬN**

**Môn:** Phân tích và thiết kế hệ thống thông tin

**Nhóm môn học:** 09

**Giảng viên:** Trần Đình Quế

**Họ tên sinh viên:** Phạm Xuân Quyền

**Mã sinh viên:** B18DCCN501

# Part 1: Question

## Q1: Styles of software architecture.

- Architectural pattern là một giải pháp chung, có thể tái sử dụng trong một vấn đề phổ biến trong kiến trúc phần mềm trong một ngữ cảnh nhất định.

- Một số softwar architectures phổ biến

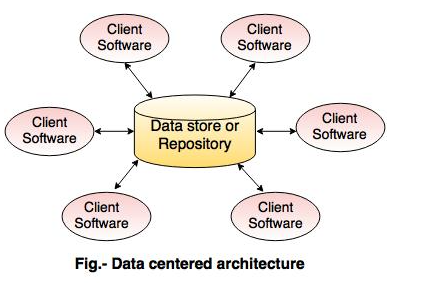
a. Data-centered architecture.

- Dữ liệu được lưu ở trung tâm của kiến trúc

- Dữ liệu lưu trữ được truy cập liên tục bởi các thành phần khác như cập nhật, xóa, thêm, sửa đổi từ kho dữ liệu.

- Tăng tính toàn vẹn

- Các tiến trình được thực thi độc lập bởi các thành phần clients.



b. Data-flow architecture.

- Kiến trúc này được áp dụng khi dữ liệu đầu vào được chuyển đổi thành một loạt các thành phần thao tác thành dữ liệu đầu ra.

- Pipe and filter pattern là một tập hợp các thành phần được gọi là filters.

- Filters được kết nối thông qua các pipe và truyền dữ liệu từ thành phần này sang thành phần tiếp theo.

c. Call and return architectures.

- Phong cách kiến ​​trúc này cho phép đạt được một cấu trúc chương trình dễ sửa đổi.

- Sau đây là các sub styles trong kiểu cấu trúc này:

+ Main program or subprogram architecture

* Chương trình được chia thành các phần nhỏ hơn theo thứ bậc
* Chương trình chính gọi nhiều thành phần chương trình trong hệ thống phân cấp mà các thành phần chương trình được chia thành chương trình con.

+  Remote procedure call architecture

* Chương trình chính hoặc các thành phần của chương trình con được phân phối trong mạng của nhiều máy tính
* Mục đích chính là để tăng hiệu suất.

d. Layered pattern

- Mẫu này có thể được sử dụng để cấu trúc các chương trình có thể được phân tách thành các nhóm nhiệm vụ con, mỗi nhiệm vụ ở một mức trừu tượng cụ thể. Mỗi lớp cung cấp các dịch vụ cho lớp cao hơn tiếp theo.

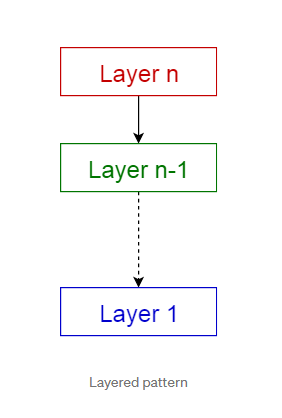
- 4 lớp thường thấy nhất của hệ thống thông tin chung như sau.

+ Presentation layer (also known as UI layer)

+ Application layer (also known as service layer)

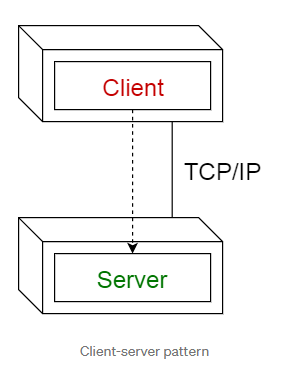
+ Business logic layer (also known as domain layer)

+ Data access layer (also known as persistence layer)



e. Client-server pattern.

- Mô hình này bao gồm hai bên; một máy chủ và nhiều máy khách. Thành phần máy chủ sẽ cung cấp dịch vụ cho nhiều thành phần máy khách. Khách hàng yêu cầu dịch vụ từ máy chủ và máy chủ cung cấp các dịch vụ liên quan cho các máy khách đó. Hơn nữa, máy chủ tiếp tục lắng nghe các yêu cầu của máy khách.



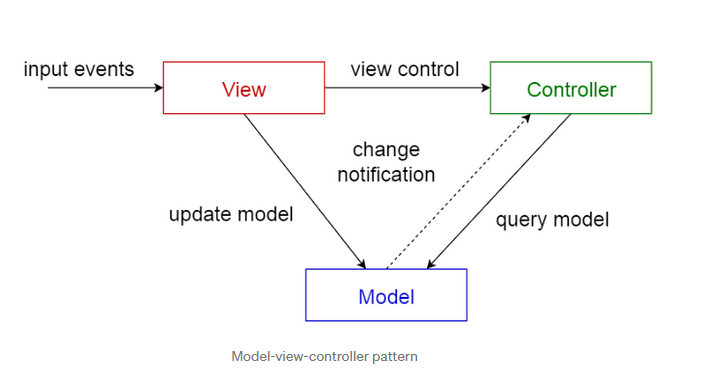
f. Model-view-controller pattern

- Mô hình này, còn được gọi là mẫu MVC, chia một ứng dụng tương tác thành 3 phần

+ model - contains the core functionality and data

+ view - displays the information to the user (more than one view may be defined)

+ controller - handles the input from the user



## Q2: Steps for analysis

- Là một quá trình thu thập và diễn giải các dữ kiện, xác định các vấn đề và phân rã một hệ thống thành các thành phần của nó.

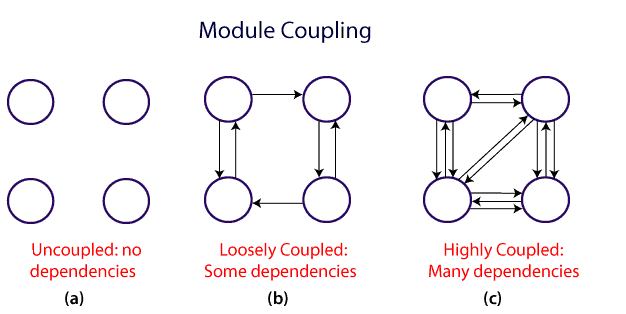
- Phân tích hệ thống được thực hiện với mục đích nghiên cứu một hệ thống hoặc các bộ phận của nó để xác định các mục tiêu của nó. Đây là một kỹ thuật giải quyết vấn đề nhằm cải thiện hệ thống và đảm bảo rằng tất cả các thành phần của hệ thống hoạt động hiệu quả để hoàn thành mục đích của chúng.

## Q3: Steps for design (cohesion, coupling….give example, Relationship, code…)

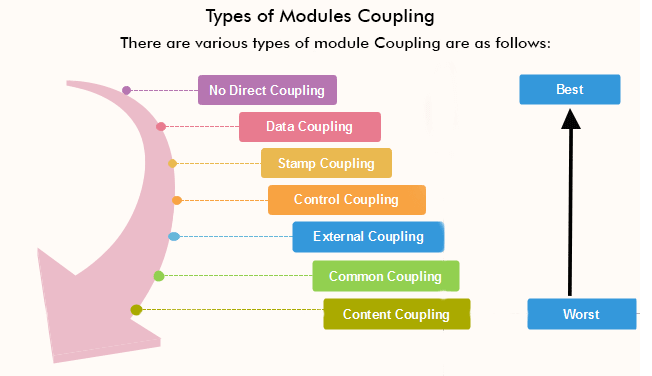
- Là một quá trình lập kế hoạch một hệ thống kinh doanh mới hoặc thay thế một hệ thống hiện có bằng cách xác định các thành phần hoặc mô-đun của nó để thỏa mãn các yêu cầu cụ thể. Trước khi lập kế hoạch, bạn cần tìm hiểu kỹ hệ thống cũ và xác định cách sử dụng máy tính tốt nhất để hoạt động hiệu quả.

- Module coupling

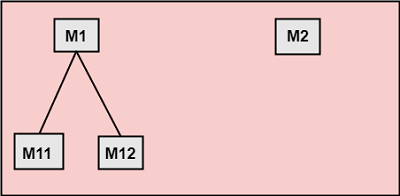
+ Trong kỹ thuật phần mềm, khớp nối là mức độ phụ thuộc lẫn nhau giữa các mô-đun phần mềm. Hai mô-đun được kết hợp chặt chẽ với nhau rất phụ thuộc vào nhau. Tuy nhiên, hai mô-đun được ghép nối lỏng lẻo không phụ thuộc vào nhau



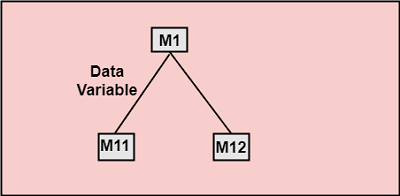
+ Types of module coupling



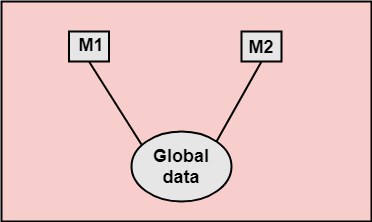
* No Direct Coupling: There is no direct coupling between M1 and M2.



* Data Coupling: When data of one module is passed to another module, this is called data coupling.



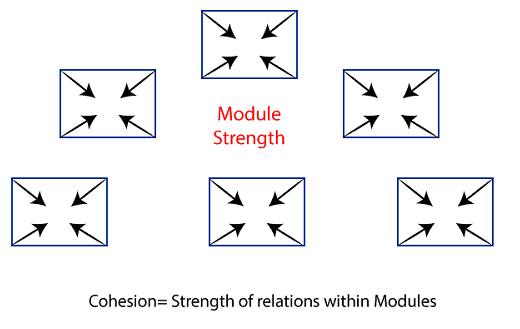
* Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.
* Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.
* External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.
* Common Coupling: Two modules are commonly coupled if they share information through some global data items.



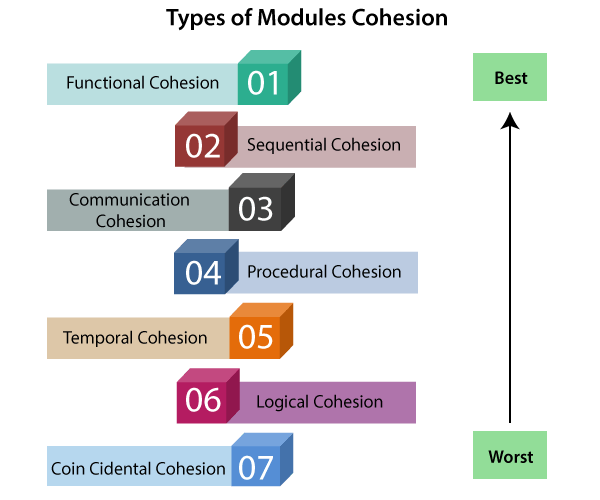
* Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

- Module Cohesion

+ Trong lập trình máy tính, sự gắn kết xác định mức độ mà các phần tử của một mô-đun thuộc về nhau. Do đó, sự gắn kết đo lường sức mạnh của mối quan hệ giữa các phần chức năng trong một mô-đun nhất định. Ví dụ, trong các hệ thống có tính gắn kết cao, chức năng có liên quan chặt chẽ với nhau.



+ Types of modules cohesion



* Functional Cohesion: Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
* Sequential Cohesion: A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
* Communicational Cohesion: A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
* Procedural Cohesion: A module is said to be procedural cohesion if the set of purposes of the module are all parts of a procedure in which a particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
* Temporal Cohesion: When a module includes functions that are associated with the fact that all the methods must be executed at the same time, the module is said to exhibit temporal cohesion.
* Logical Cohesion: A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
* Coincidental Cohesion: A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

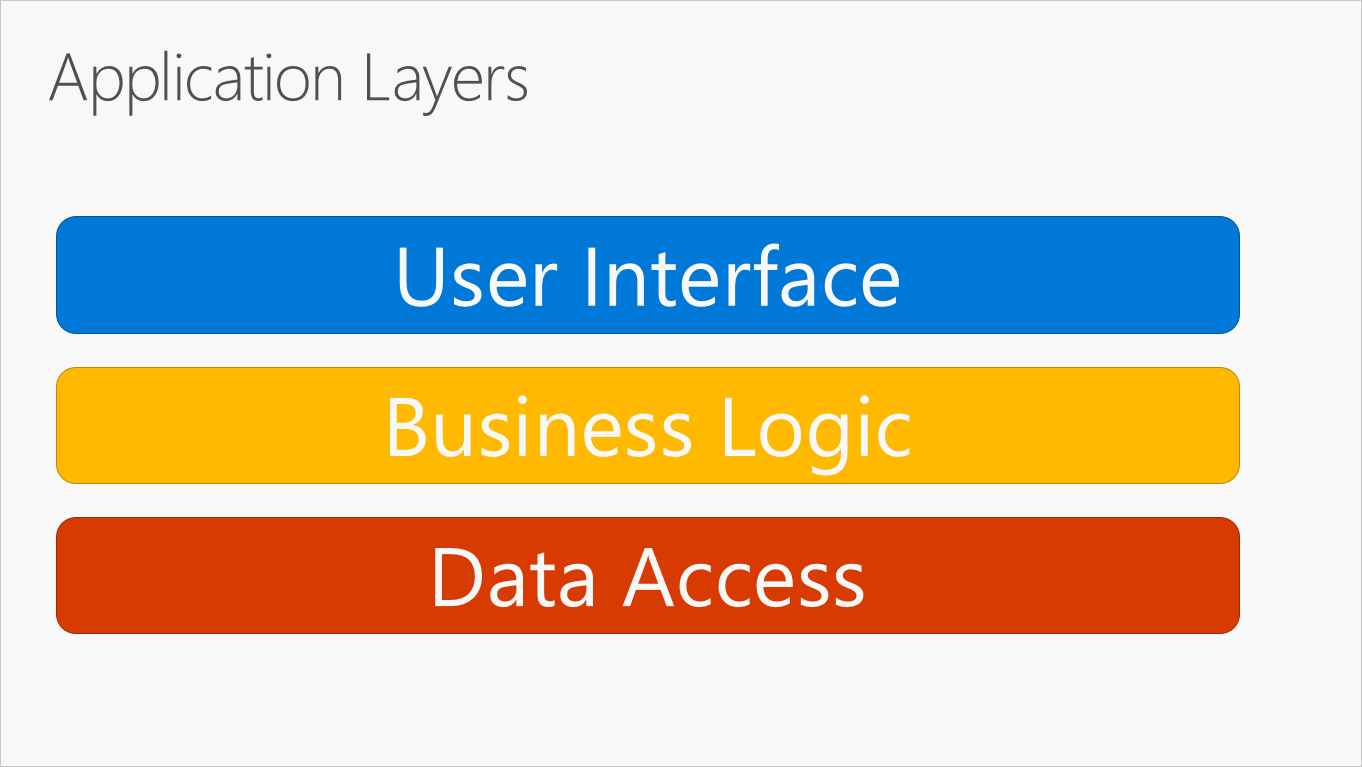
## Q4: 3 Layer architecture style & MVC

a. 3 layer architecture style.

- Khi các ứng dụng ngày càng phức tạp, một cách để quản lý sự phức tạp đó là chia nhỏ ứng dụng theo trách nhiệm hoặc mối quan tâm của nó. Cách tiếp cận này tuân theo nguyên tắc tách biệt các mối quan tâm và có thể giúp tổ chức cơ sở mã đang phát triển để các nhà phát triển có thể dễ dàng tìm thấy nơi một số chức năng được triển khai. Tuy nhiên, kiến ​​trúc phân lớp cung cấp một số lợi thế ngoài việc tổ chức mã.

- Bằng cách tổ chức mã thành các lớp, chức năng cấp thấp phổ biến có thể được sử dụng lại trong toàn bộ ứng dụng

- Các lớp (và đóng gói) giúp thay thế chức năng trong ứng dụng dễ dàng hơn nhiều. Ví dụ: ban đầu, một ứng dụng có thể sử dụng cơ sở dữ liệu SQL Server của riêng nó để duy trì tính ổn định, nhưng sau đó có thể chọn sử dụng chiến lược bền vững dựa trên đám mây hoặc chiến lược đằng sau API web. Nếu ứng dụng đã đóng gói đúng cách triển khai tính liên tục của nó trong một lớp logic, thì lớp SQL Server cụ thể đó có thể được thay thế bằng một lớp mới triển khai cùng một giao diện công khai.



- These layers are frequently abbreviated as UI, BLL (Business Logic Layer), and DAL (Data Access Layer). Using this architecture, users make requests through the UI layer, which interacts only with the BLL. The BLL, in turn, can call the DAL for data access requests. The UI layer shouldn't make any requests to the DAL directly, nor should it interact with persistence directly through other means. Likewise, the BLL should only interact with persistence by going through the DAL. In this way, each layer has its own well-known responsibility.

+ User interface

* UI Components : gồm các thành phần tạo nên giao diện của ứng dụng (GUI). Chúng chịu trách nhiệm thu nhận và hiển thị dữ liệu cho người dùng… Ví dụ : textbox, button, combobox, …
* UI Process Components : là thành phần chịu trách nhiệm quản lý các quá trình chuyển đổi giữa các UI… Ví dụ : Sắp xếp quá trình kiểm tra thông tin khách hàng:

+ Bussiness layers

* Service Interface : là thành phần giao diện lập trình mà lớp này cung cấp cho lớp Presentation sử dụng.
* Bussiness Workflows : chịu trách nhiệm xác định và điều phối các quy trình nghiệp vụ gồm nhiều bước và kéo dài. Những quy trình này phải được sắp xếp và thực hiện theo một thứ tự chính xác.
* Ví dụ : Thực hiện mua một đơn hàng trên tiki qua nhiều bước : kiểm tra gói hàng còn không?, tính tổng chi phí, cho phép giao dịch và sắp xếp việc giao hàng.
* Bussiness Components : chịu trách nhiệm kiểm tra các quy tắc nghiệp vụ, ràng buộc logic và thực hiện các công việc . Các thành phần này cũng thực hiện các dịch vụ mà Service Interface cung cấp và Business Workflows sẽ sử dụng nó.
* Ví dụ : Tiếp tục ví dụ ở trên. Bạn sẽ cần một Bussiness Component để kiểm tra gói hàng có khả dụng không ? hay một component để tính tổng chi phí,…
* Bussiness Entities : thường được sử dụng như Data Transfer Objects ( DTO ) . Bạn có thể sử dụng để truyền dữ liệu giữa các lớp (Presentation và Data Layer). Chúng thường là cấu trúc dữ liệu ( DataSets, XML,… ) hay các lớp đối tượng đã được tùy chỉnh.
* Ví dụ : tạo 1 class Student lưu trữ các dữ liệu về tên, ngày sinh, ID, lớp.

+ Data layer

* Data Access Logic Components : chịu trách nhiệm chính lưu trữ và truy xuất dữ liệu từ các nguồn dữ liệu ([Data Sources](https://topdev.vn/blog/dung-gi-de-luu-tru-data-thay-the-local-storage/)) như XML, file system,… Hơn nữa còn tạo thuận lợi cho việc dễ cấu hình và bảo trì.
* Service Agents : giúp bạn gọi và tương tác với các dịch vụ từ bên ngoài một cách dễ dàng và đơn giản.

b. MVC

- Model, view, and controller (MVC) is a well-known three-layer development architecture used for web applications developments.

+ Model- Model classes are used to implement the logic of data domains. These classes are used to retrieve, insert or update the data into the database associated with our application.

+ View- Views are used to prepare the interface of our application. By using that interface users interact with our application.

+ Controller- Controller classes are used to respond to the user’s requests. Controller classes perform the users requested actions. These classes work with model classes and select the appropriate view that should be displayed to the user according to user requests.

- Lợi ích MVC

+ MVC giúp chúng tôi triển khai cách tiếp cận phát triển theo hướng kiểm thử, trong đó chúng ta triển khai các trường hợp kiểm thử tự động trước khi viết mã. Các trường hợp kiểm thử đơn vị này giúp chúng ta xác định trước và xác minh các yêu cầu của mã mới trước khi viết nó.

+ Kiến trúc MVC giúp chúng ta kiểm soát độ phức tạp của ứng dụng bằng cách chia nó thành ba thành phần

+ MVC không sử dụng biểu mẫu dựa trên máy chủ, đó là lý do tại sao nó là lý tưởng cho những nhà phát triển muốn toàn quyền kiểm soát hành vi ứng dụng của họ.

+ Phương pháp tiếp cận phát triển theo hướng thử nghiệm được hỗ trợ bởi kiến ​​trúc MVC.

+ MVC sử dụng front controller pattern. front controller pattern xử lý nhiều yêu cầu đến bằng giao diện duy nhất (bộ điều khiển). Bộ điều khiển phía trước cung cấp khả năng kiểm soát tập trung. Chúng ta chỉ cần cấu hình một bộ điều khiển trong máy chủ web thay vì nhiều bộ điều khiển.

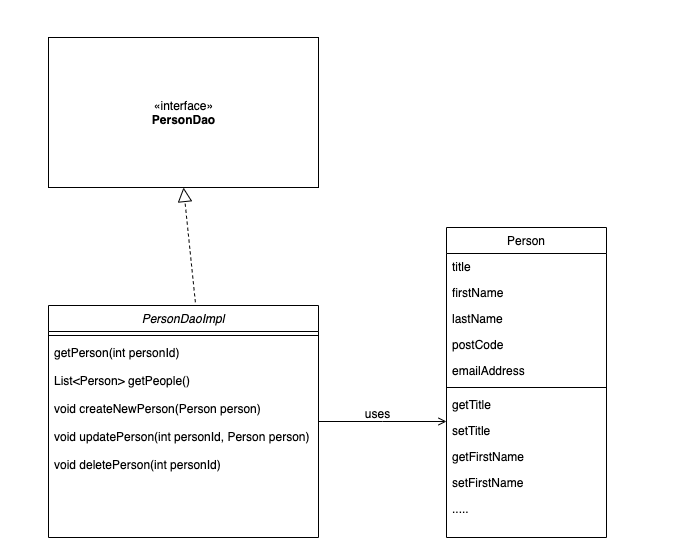
+ front controller pattern cung cấp hỗ trợ thông tin liên lạc định tuyến phong phú để thiết kế ứng dụng web của chúng ta.

## Q5: DAO? Why? class diagram & code for DAO. Review Appendix B1 & Explain the diagram

- DAO is an acronym that stands for “Data Access Object”.

- As the name implies, it is an ‘object’ that encapsulates data access operations. In object-oriented programming; it is an object that allows or abstracts away the data access element from/to an external data system.

- A class diagram of the DAO design pattern:



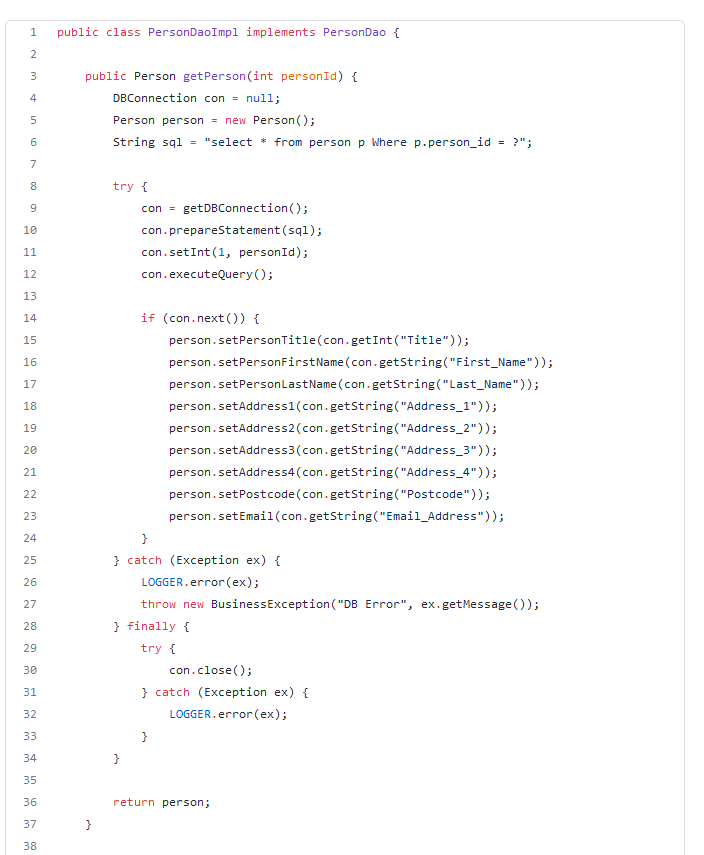
- As you can see from the above class diagram, there are three participants namely, the PersonDaointerface, the implementing class ( PersonDaoImpl ), and finally a Value Object (VO) / Entity Object (EO) / Data Transfer Object (DTO) — Person that encapsulates the data in an object.

- The DAO interface

+ A PersonDao Java interface that defines the list of operations the implementing DAO would implement.



- The Implementing Class



+ The picture above represents an implement function from PersonDAO

## Q6: Database design

- The database design process

+ Requirements analysis, or identifying the purpose of your database

+ Organizing data into tables

+ Specifying primary keys and analyzing relationships

+ Normalizing to standardize the tables

a. Requirements analysis, or identifying the purpose of your database

- Here are some ways to gather information before creating the database:

+ Interview the people who will use it

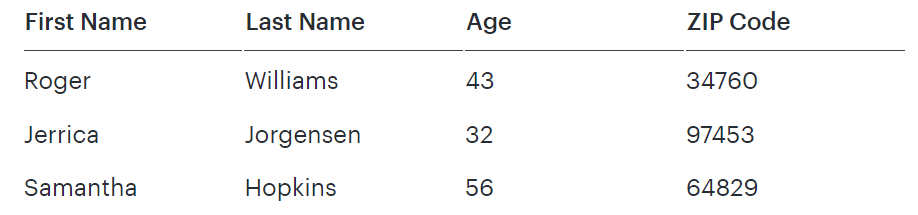
+ Analyze business forms, such as invoices, timesheets, surveys

+ Comb through any existing data systems (including physical and digital files)

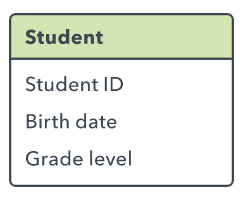
- For example Customers (Name, Address, City, State, Zip, Email address)

b. Organizing data into tables

- Within a database, related data are grouped into tables, each of which consists of rows (also called tuples) and columns, like a spreadsheet.



- For the purposes of creating a visual overview of the database, known as an entity-relationship diagram, you won’t include the actual tables. Instead, each table becomes a box in the diagram.

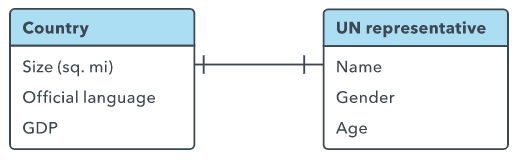


c. Specifying primary keys and analyzing relationships

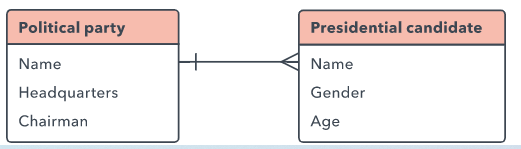
- Finally, you should decide which attribute or attributes will serve as the primary key for each table if any. A primary key (PK) is a unique identifier for a given entity, meaning that you could pick out an exact customer even if you only knew that value.

- Relationships between entities.

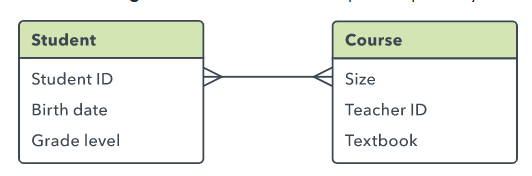
+ One-to-one



+ One-to-many



+ Many-to-many



d. Database normalization

- Once you have a preliminary design for your database, you can apply normalization rules to make sure the tables are structured correctly. Think of these rules as the industry standards.

+ 1NF (First Normal Form): Một bảng (quan hệ) được gọi là ở dạng chuẩn 1NF nếu và chỉ nếu toàn bộ các miền giá trị của các cột có mặt trong bảng (quan hệ) đều chỉ chứa các giá trị nguyên tử (nguyên tố)

+ 2NF: Một quan hệ ở dạng chuẩn 2NF nếu quan hệ đó là 1NF và các thuộc tính không khoá phải phụ thuộc hàm đầy đủ vào khoá chính

+ 3NF: Một quan hệ ở dạng chuẩn 3NF nếu quan hệ đó là 2NF và các thuộc tính không khoá phải phụ thuộc trực tiếp vào khoá chính

- Một số lưu ý khác trong database design

+ Multidimensional data: Some users may want to access multiple dimensions of a single type of data, particularly in OLAP databases. For instance, they may want to know the sales by customer, state, and month. In this situation, it’s best to create a central fact table that other customer, state, and month tables can refer to.

+ Data integrity rules: You should also configure your database to validate the data according to the appropriate rules. Many database management systems, such as Microsoft Access, enforce some of these rules automatically.

+ UML: The Unified Modeling Language (UML) is another visual way of expressing complex systems created in an object-oriented language. Several of the concepts mentioned in this guide are known in UML under different names. For instance, an entity is known as a class in UML.

+ Database management systems: Oracle DB, MySQL, Microsoft SQL Server,…

## Q7: Interface design (principles…)

- Một số guidelines khi thiết kế giao diện

+ Các nút được trên màn hình có thể dự đoán được, người dùng sử dụng các nút đó unconsiously.

+ Các nút phải dõ ràng, các icon, nhãn cần được rõ ràng giúp người dùng dễ dàng phát hiện.

+ Các giao diện nên đơn giản

+ Giảm thiểu số lượng hành động để thực hiện nhiệm vụ nhưng tập trung vào một chức năng chính trên mỗi trang

+ Đặt điều khiển gần các đối tượng mà người dùng muốn điều khiển. Ví dụ, một nút để gửi biểu mẫu phải ở gần biểu mẫu.

+ Sử dụng các mẫu thiết kế giao diện người dùng thích hợp để giúp hướng dẫn người dùng và giảm bớt gánh nặng (ví dụ: điền trước biểu mẫu).

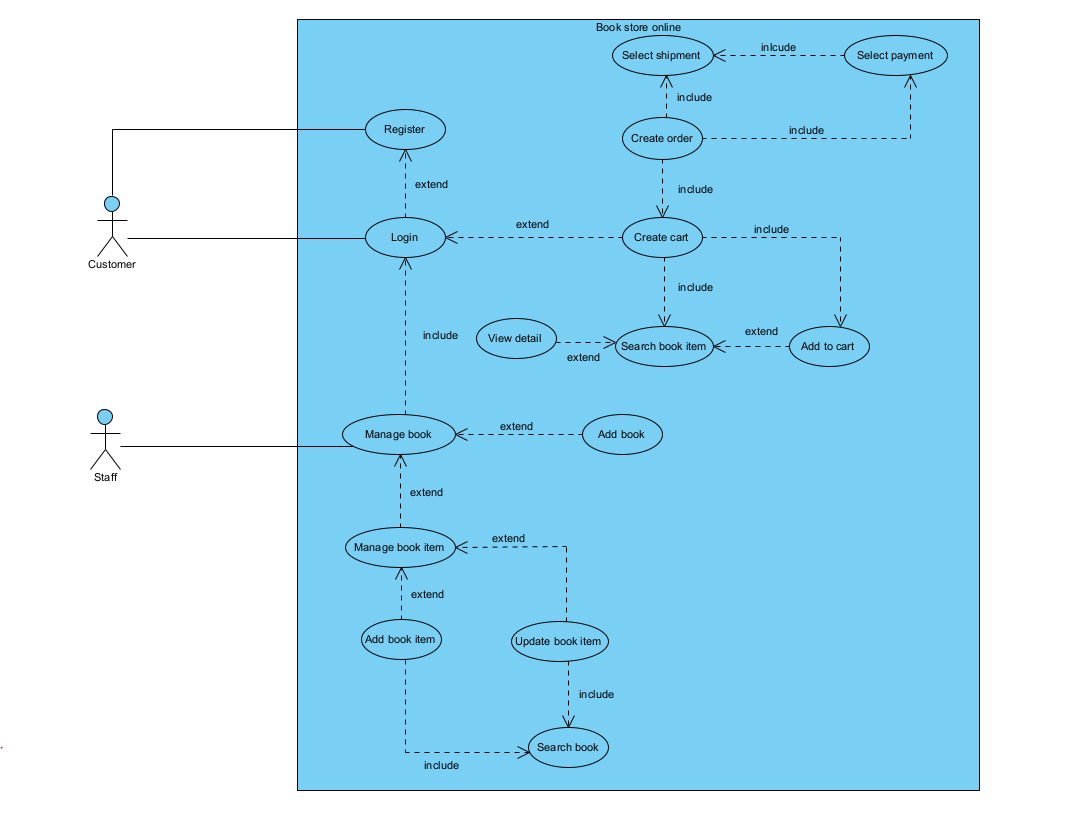
+ Beware of using dark patterns, which include hard-to-see prefilled opt-in/opt-out checkboxes and sneaking items into users’ carts.

+ Duy trì tính nhất quán của thương hiệu

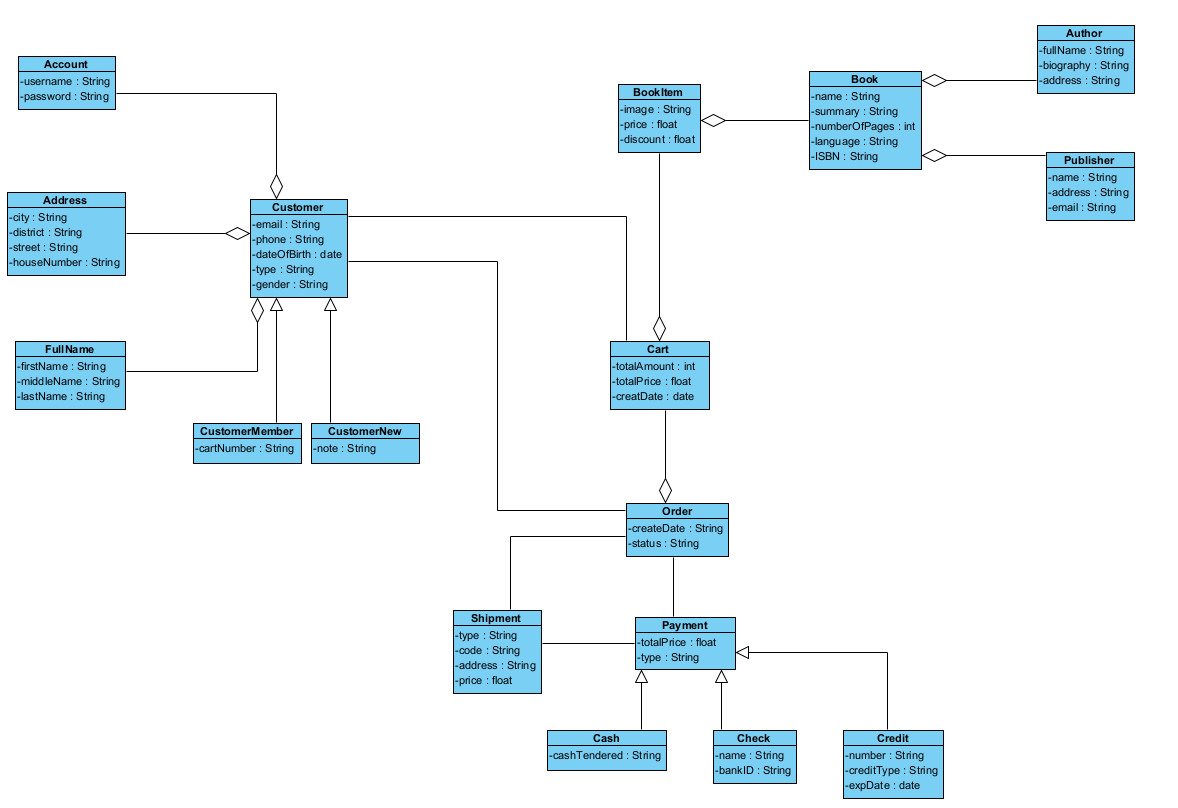
+ Always provide next steps which users can deduce *naturally*, whatever their context

# Part 2: Project

## 1. Draw a use case diagram with a detailed level

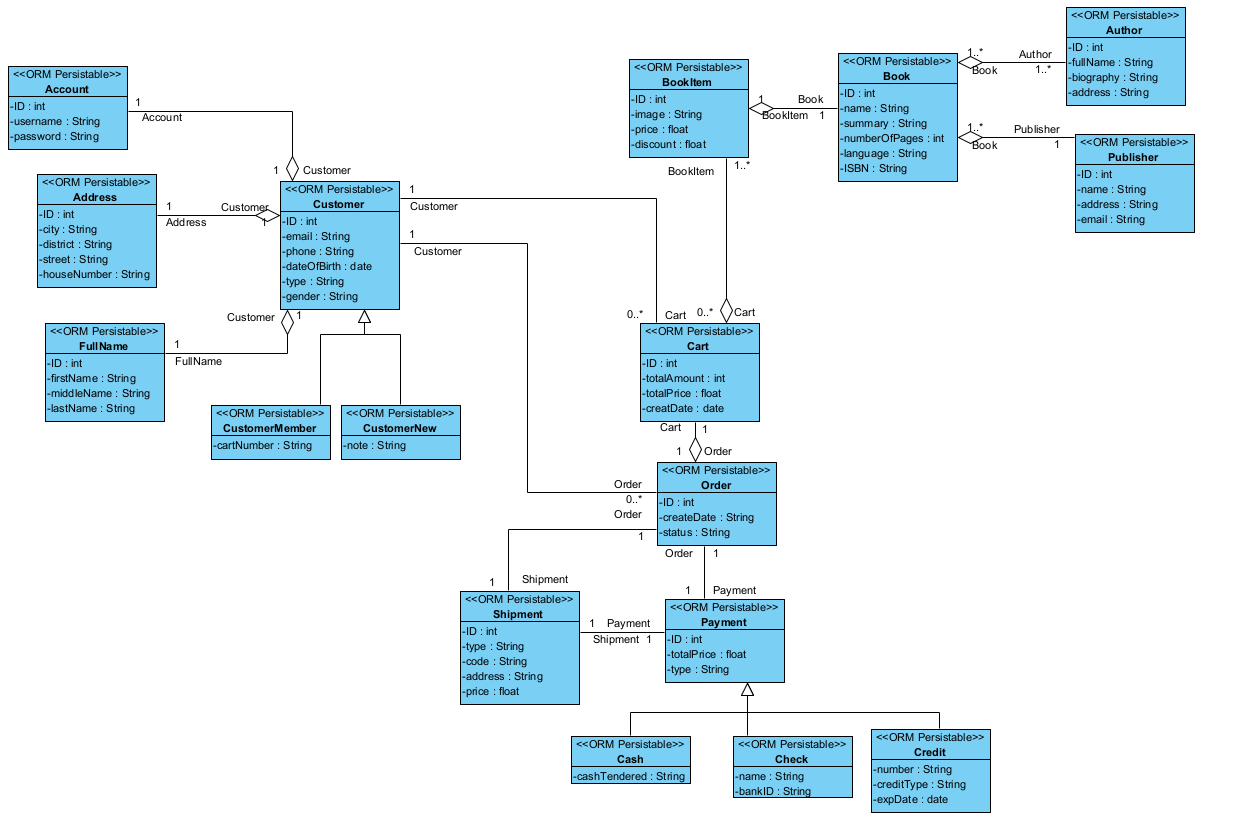


## 2. Construct a class diagram in analysis

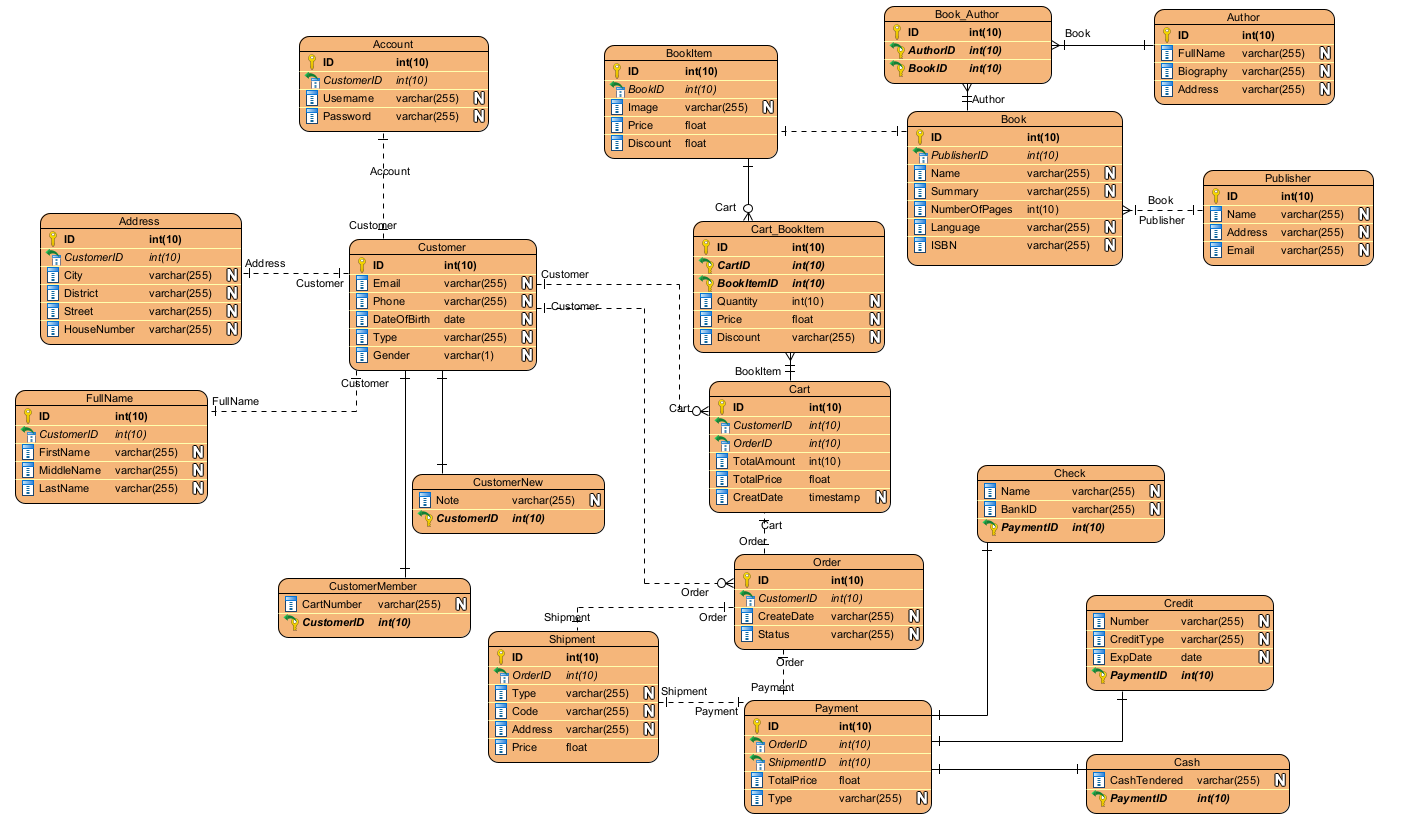


## 3. Construct a class diagram for the data model, generate data model and database MySQL

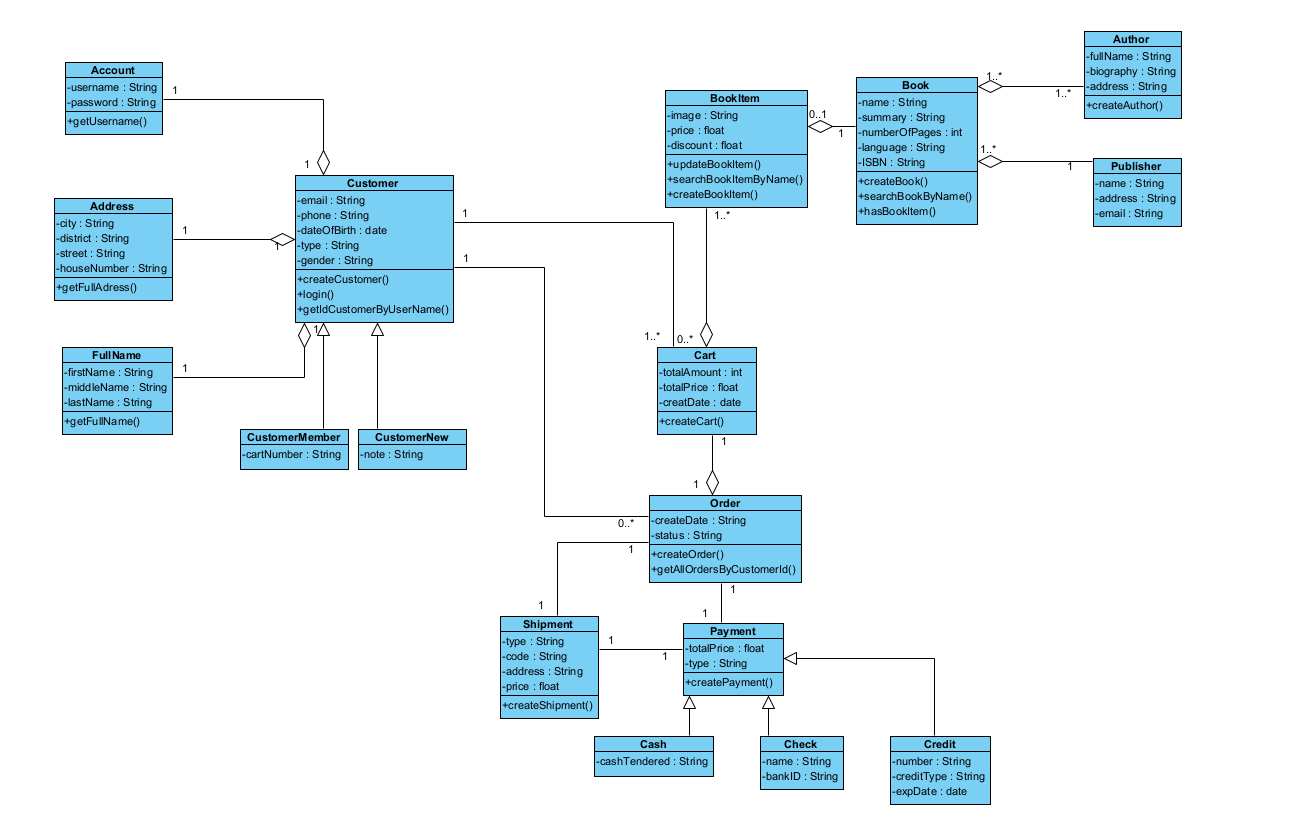
- Class diagram



- Data model



## 4. Construct a class diagram in design



## 5. Using the pattern DAO, redesign classes in your system

## 6. Construct package diagrams with 3-layer architecture with MVC for your systems

