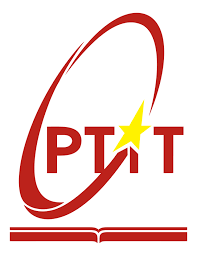


**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 1**



**BÀI THI HỌC KÌ**

**MÔN: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG THÔNG TIN**

***Đề tài: Book Store Online***

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn :**  **Sinh viên thực hiện:**  **Mã SV:**  **Nhóm môn học:** | *Thầy*Trần Đình Quế  Nguyễn Quang Liêm  B18DCCN325  10 |

**Hà Nội, 12/2021**

# **Phần I: Câu hỏi**

## **Q1. Các dạng kiến trúc phần mềm (Styles of software architecture)**

*An architectural pattern là một giải pháp chung, có thể tái sử dụng cho một vấn đề thường xảy ra trong kiến trúc phần mềm trong một ngữ cảnh nhất định. Architectural patterns tương tự như software design patterns nhưng có phạm vi rộng hơn*

\*Gồm 10 loại cơ bản:

-**Mô hình lớp** (**Layered pattern)**:

+Mô hình này có thể được sử dụng để cấu trúc các chương trình mà có thể được phân tách thành các nhóm nhiệm vụ con, mỗi nhóm 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**)

**+**Usage:

* General desktop applications.
* E-commerce web applications.

-**Mô hình client-server** (**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 khách hàng.

+ Usage:

* Các ứng dụng trực tuyến như email, chia sẻ tài liệu và ngân hàng.

-**Mô hình master-slave:**

+Mô hình này bao gồm hai bên:**master** và **slave** (chính- phụ). Thành phần master phân phối công việc giữa các thành phần slave giống nhau và tính toán kết quả cuối cùng từ kết quả mà các thành phần slave trả lại.

+ Usage:

* Trong sao chép cơ sở dữ liệu, master database coi là nguồn có thẩm quyền và các cơ sở dữ liệu phụ được đồng bộ hóa với nó.
* Các thiết bị ngoại vi kết nối với bus trong hệ thống máy tính (ổ chính và ổ phụ).

-**Mô hình pipe-filter**:

+Mô hình này có thể được sử dụng để cấu trúc hệ thống tạo ra và xử lý một luồng dữ liệu. Mỗi bước xử lý được bao gồm trong một thành phần filter.Dữ liệu cần xử lý được đưa qua các pipe.Các pipe này có thể được sử dụng để đệm hoặc cho các mục đích đồng bộ hóa.

+ Usage:

* Trình biên dịch. Các bộ lọc liên tiếp thực hiện phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa và tạo mã.
* Quy trình làm việc trong bioinformatics.

-**Mô hình broker**:

+Mô hình này được sử dụng để cấu trúc hệ thống phân tán với các thành phần tách rời. Các thành phần này có thể tương tác với nhau bằng các lệnh gọi dịch vụ từ xa. Một thành phần broker chịu trách nhiệm điều phối giao tiếp giữa các thành phần

+Các máy chủ công bố các khả năng của chúng (dịch vụ và đặc điểm) cho broker.Khách hàng yêu cầu một dịch vụ từ broker và sau đó broker chuyển hướng khách hàng đến một dịch vụ phù hợp từ sổ đăng ký của nó.

+ Usage::

* Phần mềm môi giới tin nhắn như Apache ActiveMQ , Apache Kafka , RabbitMQ và JBoss Messaging .

-**Mô hình peer-to-peer**:

+Trong mô hình này, các thành phần riêng lẻ được gọi là peer.Các peer có thể hoạt động như 1 client yêu cầu dịch vụ từ các peer khác.Một peer có thể hoạt động như một máy khách hoặc một máy chủ hoặc cả hai, và nó có thể thay đổi vai trò của mình theo thời gian.

+ Usage:

* Mạng chia sẻ tệp như Gnutella và G2
* Các giao thức đa phương tiện như P2PTV và PDTP .
* Các sản phẩm dựa trên tiền điện tử như Bitcoin và Blockchain

-**Mô hình Event-bus:**

+Mô hình này chủ yếu đề cập đến các sự kiện và có 4 thành phần chính:event source,event listener,channel và event bus

Sources xuất bản tin nhắn đến các kênh cụ thể trên một event bus.Listener đăng ký các kênh cụ thể. Listener được thông báo về các tin nhắn được xuất bản lên kênh mà họ đã đăng ký trước đó

+ Usage:

* Phát triển Android
* Dịch vụ thông báo

-**Mô hình Model-View-Controller:**

+Mô hình này, còn được gọi là mô hình MVC, chia một ứng dụng tương tác thành 3 phần:

* **Model** – chứa chức năng và dữ liệu cốt lõi
* **View** – hiển thị thông tin cho người dùng (nhiều hơn một view có thể được xác định)
* **Controller** – xử lý đầu vào từ người dùng

+Điều này được thực hiện để tách các đại diện nội bộ của thông tin khỏi cách thông tin được trình bày và chấp nhận từ người dùng. Nó tách các thành phần và cho phép sử dụng lại code hiệu quả.

+ Usage:

* Kiến trúc cho các ứng dụng World Wide Web trong các ngôn ngữ lập trình chính.
* Các khung công tác web như Django và Rails .

-**Mô hình blackboard**:

+Mô hình này hữu ích cho các vấn đề mà không có chiến lược giải pháp xác định nào được biết đến. Mẫu bảng đen gồm 3 thành phần chính:

* blackboard– một bộ nhớ chung có cấu trúc chứa các đối tượng từ không gian giải pháp
* Knowledge source– các mô-đun chuyên biệt với đại diện của riêng chúng
* Control component – lựa chọn, cấu hình và thực thi các mô-đun.

+Tất cả các thành phần có quyền truy cập vào bảng đen. Các thành phần có thể tạo ra các đối tượng dữ liệu mới được thêm vào bảng đen. Các thành phần tìm kiếm các loại dữ liệu cụ thể trên bảng đen và có thể tìm các loại dữ liệu này theo mẫu khớp với knowledge source hiện có.

+ Usage:

* Nhận dạng giọng nói
* Nhận dạng và theo dõi phương tiện
* Xác định cấu trúc protein
* Giải thích tín hiệu sonar.

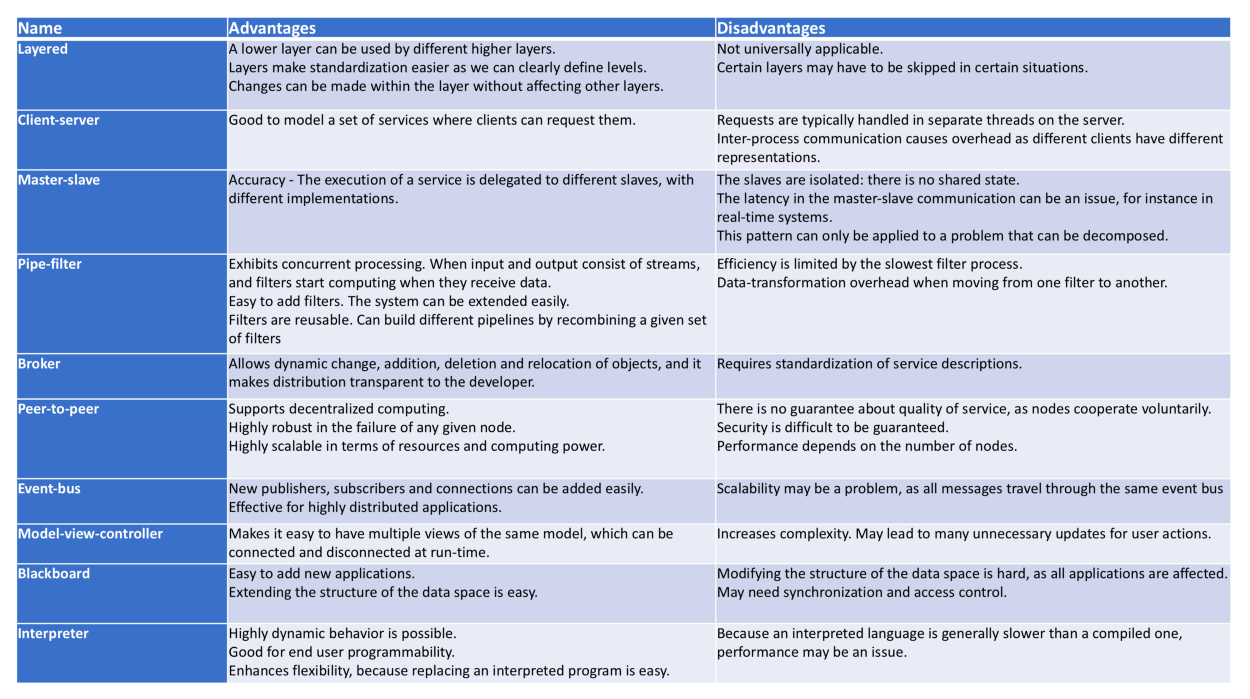
-**Mô hình Interpreter**

+Mô hình này được sử dụng để thiết kế một thành phần thông giữa các chương trình được viết bằng một ngôn ngữ chuyên dụng.Nó chủ yếu chỉ định cách đánh giá các dòng chương trình, được gọi là các câu hoặc biểu thức được viết bằng một ngôn ngữ cụ thể. Ý tưởng cơ bản là có một lớp cho mỗi ký hiệu của ngôn ngữ.

+ Usage:

* Các ngôn ngữ truy vấn cơ sở dữ liệu như SQL.
* Các ngôn ngữ được sử dụng để mô tả các giao thức giao tiếp.

**\*Comparison of Architectural Patterns**



**\*Các kiểu kiến ​​trúc phổ biến được sử dụng trong khi thiết kế phần mềm như sau:**

-*Kiến trúc tập trung cơ sở dữ liệu*:

+dữ liệu được lưu trữ ở file hoặc cơ sở dữ liệu ở trung tâm kiến trúc

+Dữ liệu lưu trữ được truy cập liên tục bởi việc cập nhật, xóa, thêm, sửa đổi từ kho dữ liệu.

+Kiến trúc tập trung cơ sở dữ liệu có tính toàn vẹn

+Quá trình truyền dữ liệu giữa các client sử dụng cơ chế bảng đen

+Các quy trình được thực thi độc lập ở mỗi client

-*Kiến trúc luồng dữ liệu*

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

+Pipe và filter pattern là 1 tập hợp các thành phần được gọi là các filter

+Các filter được kết nối với nhau qua các pipe và truyền dữ liệu từ thành phần này sang thành phần tiếp theo

+Luồng dữ liệu biến đổi thành một dòng biến đổi duy nhất, sau đó nó được gọi là batch sequential

-*Kiến trúc call-and-return*

+Kiểu kiến trúc này cho phép đạt được 1 cấu trúc chương trình dễ sửa đổi

**\*Các kiểu khác:**

-*Kiến trúc chương trình chính hoặc chương trình con*

+Chương trình được chia thành các phần nhỏ tùy theo thứ bậc

+Chương trình chính gọi nhiều thành phần chương trình trong hệ thống phân cấp cái mà các thành phần chương trình được chia thành chương trình con.

-*Kiến trúc thủ tục gọi từ xa*

+Chương trình chính hoặc các thành phần của chương trình con được phân phối trong mạng lưới nhiều máy tính

+Mục đích chính là để tăng hiệu suất

*-Kiến trúc hướng đối tượng*

+Kiến trúc này là phiên bản mới nhất của kiến ​​trúc call-and-return.

+Nó bao gồm gói dữ liệu và phương thức.

*-Kiến trúc phân lớp*

+Các lớp khác nhau được xác định trong kiến ​​trúc. Nó bao gồm lớp bên ngoài và bên trong.

+Các thành phần của lớp ngoài quản lý các hoạt động của giao diện người dùng.

+Các thành phần thực thi giao diện hệ điều hành ở lớp bên trong.

+Các lớp bên trong là lớp ứng dụng, lớp tiện ích và lớp lõi.

+Trong nhiều trường hợp, có thể có nhiều hơn một mẫu phù hợp và phong cách kiến ​​trúc thay thế có thể được thiết kế và đánh giá.

## **Q2. Các bước phân tích (Steps for analysis)**

**Phân tích hệ thống** 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.

Phân tích **xác định những gì hệ thống phải làm.**

Mục tiêu của giai đoạn là xác định các thông tin và chức năng xử lý của hệ thống, cụ thể như sau:

* Xác định *yêu cầu của HTTT* gồm: các chức năng chính – phụ; nghiệp vụ cần phải xử lý đảm bảo tính chính xác, tuân thủ đúng các văn bản luật và quy định hiện hành; đảm bảo tốc độ xử lý và khả năng nâng cấp trong tương lai.
* Phân tích và đặc tả *mô hình phân cấp chức năng tổng thể* thông qua sơ đồ BFD (Business Flow Diagram), từ mô hình BFD sẽ tiếp tục được xây dựng thành mô hình luồng dữ liệu DFD (Data Flow Diagram) thông qua quá trình phân rã chức năng theo các mức 0, 1, 2 ở từng ô xử lý.
* Phân tích *bảng dữ liệu.* Cần đưa vào hệ thống những bảng dữ liệu (data table) gồm các trường dữ liệu (data field) nào? Xác định khóa chính (primary key), khóa ngoại (foreign key) cũng như mối quan hệ giữa các bảng dữ liệu (relationship) và ràng buộc (constraint) dữ liệu cần thiết.

Ở giai đoạn này, các chuyên gia sẽ đặc tả sơ bộ các bảng dữ liệu trên giấy để có cái nhìn khách quan. Qua đó, xác định các giải pháp tốt nhất cho hệ thống đảm bảo đúng các yêu cầu đã khảo sát trước khi thực hiện trên các phần mềm chuyên dụng.

Các nhà phân tích sử dụng các công cụ khác nhau để hiểu và mô tả hệ thống thông tin. Một trong những cách là sử dụng phân tích có cấu trúc.

### **1. Phân tích có cấu trúc (Structured analysis) là gì?**

Phân tích có cấu trúc là một phương pháp phát triển cho phép người phân tích hiểu hệ thống và các hoạt động của nó một cách hợp lý. Đây là một cách tiếp cận có hệ thống, sử dụng các công cụ đồ họa để phân tích và tinh chỉnh các mục tiêu của hệ thống hiện có và phát triển một đặc tả hệ thống mới mà người dùng có thể dễ dàng hiểu được.

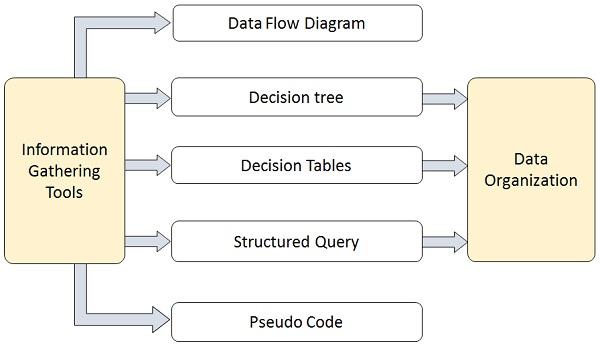
Nó có các đặc điểm sau:

* Nó là đồ họa chỉ định cách trình bày của ứng dụng.
* Nó phân chia các quá trình để đưa ra một bức tranh rõ ràng về luồng hệ thống.
* Nó mang tính logic hơn là tính vật lý, tức là các phần tử của hệ thống không phụ thuộc vào nhà cung cấp hoặc phần cứng.
* Đây là một cách tiếp cận hoạt động từ tổng quan cấp cao đến chi tiết cấp thấp hơn.

### **2. Công cụ phân tích có cấu trúc (Structured analysis tool)**

Trong quá trình Phân tích có cấu trúc, nhiều công cụ và kỹ thuật khác nhau được sử dụng để phát triển hệ thống:

* Sơ đồ luồng dữ liệu (Data Flow Diagrams –DFD)
* Từ điển dữ liệu (Data Dictionary)
* Cây quyết định (Decision Trees)
* Bảng Quyết định (Decision Tables)
* Tiếng Anh có cấu trúc (Structured English)
* Mã giả (pseudocode)



### **3. Sơ đồ luồng dữ liệu (Data Flow Diagrams –DFD) hoặc Biểu đồ bong bóng (Burble Chart)**

Nó là một kỹ thuật được phát triển bởi Larry Constantine để thể hiện các yêu cầu của hệ thống dưới dạng đồ họa.

* Nó cho thấy luồng dữ liệu giữa các chức năng khác nhau của hệ thống và chỉ định cách hệ thống hiện tại được thực hiện.
* Đây là giai đoạn ban đầu của giai đoạn thiết kế phân chia theo chức năng các đặc tả yêu cầu xuống mức chi tiết thấp nhất.
* Bản chất đồ họa của nó làm cho nó trở thành một công cụ giao tiếp tốt giữa người dùng và nhà phân tích hoặc nhà phân tích và nhà thiết kế hệ thống.
* Nó cung cấp một cái nhìn tổng quan về dữ liệu mà một hệ thống xử lý, những chuyển đổi nào được thực hiện, những dữ liệu nào được lưu trữ, những kết quả nào được tạo ra và chúng lưu chuyển đến đâu.

**3.1. Các yếu tố cơ bản của DFD**

DFD dễ hiểu và khá hiệu quả khi thiết kế cần thiết không rõ ràng và người dùng muốn có một ngôn ngữ ký hiệu để giao tiếp. Tuy nhiên, nó đòi hỏi một số lượng lớn các lần lặp để có được giải pháp chính xác và đầy đủ nhất.

Bảng sau đây cho thấy các ký hiệu được sử dụng trong thiết kế DFD và ý nghĩa của chúng

|  |  |  |
| --- | --- | --- |
| **Tên kí tự** | **Kí tự** | **Ý Nghĩa** |
| Hình vuông |  | Nguồn hoặc đích đến của dữ liệu |
| Múi tên |  | Luồng dữ liệu |
| Hình tròn |  | Quá trình chuyển đổi dữ liệu |
| Hình chữ nhật mở |  | Kho dữ liệu |

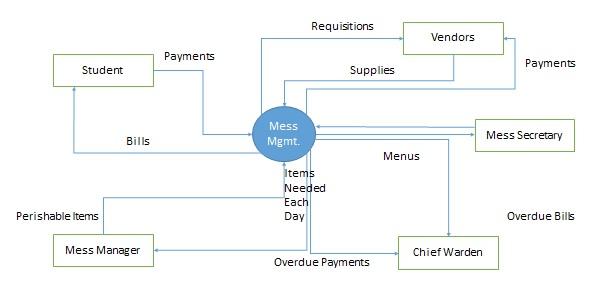
**3.2. Các loại DFD**

|  |  |
| --- | --- |
| **DFD logic (Logic DFD)** | **DFD vật lí (Physical DFD)** |
| Nó phụ thuộc vào việc cài đặt. Nó cho thấy những chức năng nào được thực hiện. | Nó là độc lập với việc cài đặt. Nó chỉ tập trung vào luồng dữ liệu giữa các quy trình. |
| Nó cung cấp thông tin chi tiết về phần cứng, phần mềm, tệp và con người ở mức độ thấp. | Nó giải thích các sự kiện của hệ thống và dữ liệu theo yêu cầu của mỗi sự kiện. |
| Nó mô tả cách hệ thống hiện tại hoạt động và cách một hệ thống sẽ được cài đặt. | Nó chỉ ra cách thức hoạt động của nghiệp vụ; không phải làm thế nào hệ thống có thể được cài đặt. |

**3.3. Sơ đồ ngữ cảnh (Context Diagram)**

Một sơ đồ ngữ cảnh giúp hiểu toàn bộ hệ thống bằng một DFD cung cấp cho bạn cái nhìn tổng quan về một hệ thống. Nó bắt đầu với việc đề cập đến các quy trình chính với một số chi tiết nhỏ và sau đó chuyển sang cung cấp thêm chi tiết về các quy trình với cách tiếp cận từ trên xuống.

Sơ đồ ngữ cảnh của quản lý nhà ăn (mess management) được hiển thị bên dưới.



### **4. Từ điển dữ liệu (Data Dictionary)**

Từ điển dữ liệu là một kho lưu trữ có cấu trúc các phần tử dữ liệu trong hệ thống. Nó lưu trữ mô tả của tất cả các phần tử dữ liệu DFD gồm chi tiết và định nghĩa về luồng dữ liệu, kho dữ liệu, dữ liệu được lưu trữ trong kho dữ liệu và các quy trình.

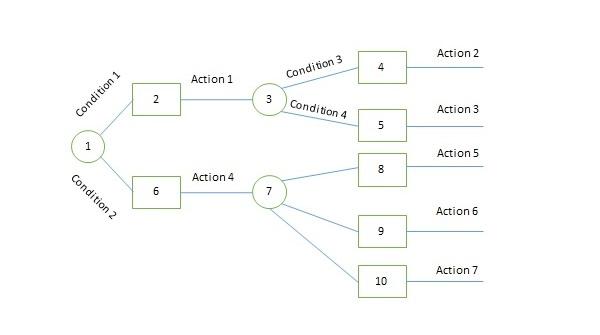
Từ điển dữ liệu cải thiện giao tiếp giữa nhà phân tích và người dùng. Nó đóng một vai trò quan trọng trong việc xây dựng cơ sở dữ liệu. Hầu hết các DBMS đều có từ điển dữ liệu như một tính năng tiêu chuẩn. Ví dụ, hãy tham khảo bảng sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No.** | **Tên dữ liệu** | **Mô tả** | **Số kí tự** |
| 1 | ISBN | Số ISBN | 10 |
| 2 | TITLE | Tên sách | 60 |
| 3 | SUB | Chủ đề sách | 80 |
| 4 | ANAME | Tên tác giả | 15 |

### **5. Cây quyết định (Decision Trees)**

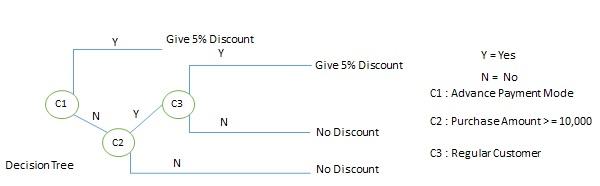
Cây quyết định là một phương pháp để xác định các mối quan hệ phức tạp bằng cách mô tả các quyết định và tránh các vấn đề trong giao tiếp. Cây quyết định là một sơ đồ thể hiện các hành động và điều kiện thay thế trong khuôn khổ cây ngang. Do đó, nó mô tả các điều kiện cần xem xét đầu tiên, thứ hai, v.v.

Cây quyết định mô tả mối quan hệ của từng điều kiện và các hành động được phép của chúng. Một nút hình vuông biểu thị một hành động và một nút tròn biểu thị một điều kiện. Nó buộc các nhà phân tích phải xem xét trình tự của các quyết định và xác định quyết định thực tế cần phải đưa ra.



Hạn chế chính của cây quyết định là nó thiếu thông tin trong định dạng của nó để mô tả những kết hợp điều kiện nào khác mà bạn có thể thực hiện để kiểm tra. Nó là một đại diện duy nhất của các mối quan hệ giữa các điều kiện và hành động.

Ví dụ, hãy tham khảo cây quyết định sau:



### **6. Bảng Quyết định (Decision Tables)**

Bảng quyết định là một phương pháp mô tả mối quan hệ logic phức tạp một cách chính xác và dễ hiểu.

* Nó hữu ích trong các tình huống mà các hành động phụ thuộc vào sự xuất hiện của một hoặc một số kết hợp các điều kiện độc lập.
* Nó là một ma trận chứa hàng hoặc cột để xác định một vấn đề và các hành động.

**6.1. Các thành phần của Bảng Quyết định**

Stub điều kiện (Condition Stub) - Nó nằm ở góc phần tư phía trên bên trái, liệt kê tất cả các điều kiện cần được kiểm tra.

Stub hành động (Action Stub) - Nó nằm ở góc phần tư phía dưới bên trái, phác thảo tất cả các hành động cần thực hiện để đáp ứng điều kiện như vậy.

Mục nhập điều kiện (Condition Entry)- Nó nằm ở góc phần tư phía trên bên phải, cung cấp câu trả lời cho các câu hỏi được hỏi trong góc phần tư Stub điều kiện.

Mục nhập hành động (Action stub) - Nó nằm ở góc phần tư phía dưới bên phải cho biết hành động thích hợp là kết quả của các câu trả lời cho các điều kiện trong góc phần tư mục nhập điều kiện.

Các mục trong bảng quyết định được đưa ra bởi Quy tắc quyết định (Decision Rules) xác định mối quan hệ giữa các tổ hợp điều kiện và các quá trình hành động.

Trong phần quy tắc

Y cho thấy sự tồn tại của một điều kiện.

N đại diện cho điều kiện không được thỏa mãn.

Một khoảng trống (A blank) - hành động sẽ được bỏ qua.

X - hành động sẽ được thực hiện.

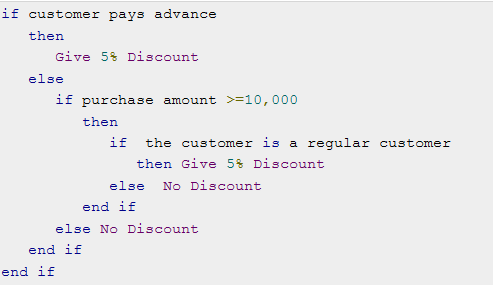
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Điều kiện** | **Quy tắc 1** | **Quy tắc 2** | **Quy tắc 3** | **Quy tắc 4** |
| Advance payment made | Y | N | N | N |
| Purchase amount >= 10000 | - | Y | Y | N |
| Regular Customer | - | Y | N | - |
| **Hành động** |  |  |  |  |
| Give 5% discount | X | X | - | - |
| Give no discount | - | - | X | X |

### **7. Tiếng Anh có cấu trúc (Structured English)**

Tiếng Anh cấu trúc có nguồn gốc từ ngôn ngữ lập trình có cấu trúc giúp mô tả quy trình dễ hiểu và chính xác hơn. Nó dựa trên logic thủ tục sử dụng các câu xây dựng và mệnh lệnh được thiết kế để thực hiện thao tác cho hành động.

* Nó được sử dụng tốt nhất khi các chuỗi và vòng lặp trong một chương trình phải được xem xét và vấn đề cần các chuỗi hành động với các quyết định.
* Nó không có quy tắc cú pháp nghiêm ngặt. Nó thể hiện tất cả logic về cấu trúc quyết định tuần tự và sự lặp lại.

Ví dụ, hãy xem chuỗi hành động sau:



### **8. Mã giả (Psudocode)**

Mã giả không tuân theo bất kỳ ngôn ngữ lập trình nào và diễn đạt logic bằng tiếng Anh đơn giản.

* Nó có thể chỉ định logic lập trình vật lý mà không cần mã hóa thực tế trong và sau khi thiết kế vật lý.
* Nó được sử dụng kết hợp với lập trình có cấu trúc.
* Nó thay thế các sơ đồ của một chương trình

### **9. Hướng dẫn lựa chọn công cụ thích hợp**

Sử dụng các nguyên tắc sau để chọn công cụ thích hợp nhất phù hợp với yêu cầu của bạn

* Sử dụng DFD ở phân tích cấp cao hoặc cấp thấp để cung cấp tài liệu hệ thống tốt.
* Sử dụng từ điển dữ liệu để đơn giản hóa cấu trúc nhằm đáp ứng yêu cầu dữ liệu của hệ thống.
* Sử dụng tiếng Anh có cấu trúc nếu có nhiều vòng lặp và hành động phức tạp.
* Sử dụng các bảng quyết định khi có một số lượng lớn các điều kiện cần kiểm tra và logic phức tạp.
* Sử dụng cây quyết định khi trình tự các điều kiện là quan trọng và nếu có ít điều kiện cần được kiểm tra.

## **Q3. Các bước để thiết kế (Steps for design)**

Thiết kế hệ thống là giai đoạn thu hẹp khoảng cách giữa miền vấn đề và hệ thống hiện có theo cách có thể quản lý được. Giai đoạn này tập trung vào miền giải pháp, tức là "làm thế nào để cài đặt?"

Thông qua thông tin được thu thập từ quá trình khảo sát và phân tích, các chuyên gia sẽ chuyển hóa vào phần mềm, công cụ chuyên dụng để đặc tả thiết kế hệ thống chi tiết. Giai đoạn này được chia làm hai bước sau:

*Bước 1: Thiết kế tổng thể*

Trên cơ sở các bảng dữ liệu đã phân tích và đặc tả trên giấy sẽ được thiết kế dưới dạng mô hình mức ý niệm bằng phần mềm chuyên dụng như Sybase PowerDesigner, CA ERwin Data Modeler. Bằng mô hình mức ý niệm sẽ cho các chuyên gia có **cái nhìn tổng quát nhất về mối quan hệ giữa các đối tượng** trước khi chuyển đổi thành mô hình mức vật lý.

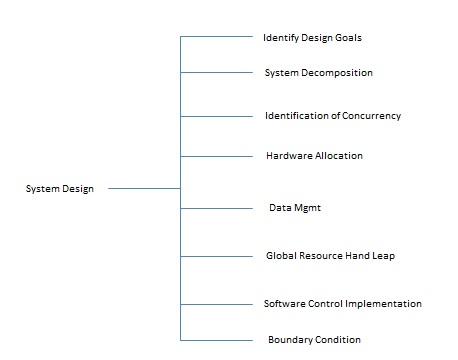
*Bước 2: Thiết kế chi tiết*

* Thiết kế cơ sở dữ liệu (Database): Với mô hình mức vật lý hoàn chỉnh ở giai đoạn thiết kế đại thể sẽ được kết sinh mã thành file sql.
* Thiết kế truy vấn, thủ tục, hàm: thu thập, xử lý thông tin nhập và đưa ra thông tin chuẩn xác theo đúng nghiệp vụ.
* Thiết kế giao diện chương trình đảm bảo phù hợp với môi trường, văn hóa và yêu cầu của doanh nghiệp thực hiện dự án.
* Thiết kế chức năng chương trình đảm bảo tính logic trong quá trình nhập liệu và xử lý cho người dùng.
* Thiết kế báo cáo. Dựa trên các yêu cầu của mỗi doanh nghiệp và quy định hiện hành sẽ thiết kế các mẫu báo cáo phù hợp hoặc cho phép doanh nghiệp tư tạo mẫu báo cáo ngay trên hệ thống.
* Thiết kế các kiểm soát bằng hình thức đưa ra các thông báo, cảnh báo hoặc lỗi cụ thể tạo tiện lợi và kiểm soát chặt chẽ quá trình nhập liệu với mục tiêu tăng độ chính xác cho dữ liệu.

Tóm lại, *thiết kế là việc áp dụng các công cụ, phương pháp, thủ tục để tạo ra mô hình hệ thống cần sử dụng*. Sản phẩm cuối cùng của giai đoạn thiết kế là đặc tả hệ thống ở dạng nó tồn tại thực tế, sao cho nhà lập trình và kỹ sư phần cứng có thể dễ dàng chuyển thành chương trình và cấu trúc hệ thống

Đây là giai đoạn mà tài liệu SRS được chuyển đổi thành một định dạng có thể được thực hiện và quyết định cách hệ thống sẽ hoạt động.

Trong giai đoạn này, hoạt động phức tạp của phát triển hệ thống được chia thành nhiều hoạt động phụ nhỏ hơn, chúng phối hợp với nhau để đạt được mục tiêu chính của phát triển hệ thống.



### **1. Đầu vào cho thiết kế hệ thống**

Thiết kế hệ thống có các yếu tố đầu vào sau:

* Báo cáo công việc
* Kế hoạch xác định yêu cầu
* Phân tích tình hình hiện tại
* Các yêu cầu hệ thống được đề xuất bao gồm mô hình dữ liệu khái niệm, DFD đã sửa đổi và Siêu dữ liệu (Metadata).

### **2. Đầu ra cho thiết kế hệ thống**

Thiết kế hệ thống cho các kết quả sau:

* Những thay đổi về cơ sở hạ tầng và tổ chức đối với hệ thống được đề xuất.
* Một lược đồ dữ liệu, thường là một lược đồ quan hệ.
* Siêu dữ liệu để xác định bảng / tệp và cột / dữ liệu-mục.
* Sơ đồ phân cấp chức năng hoặc bản đồ trang web mô tả bằng đồ thị cấu trúc chương trình.
* Mã thật hoặc mã giả cho mỗi mô-đun trong chương trình.
* Một bản mẫu cho hệ thống được đề xuất.

### **3. Các loại thiết kế hệ thống**

**3.1. Thiết kế logic**

Thiết kế logic liên quan đến một biểu diễn trừu tượng của luồng dữ liệu, đầu vào và đầu ra của hệ thống. Nó mô tả các đầu vào (nguồn), đầu ra (đích), cơ sở dữ liệu (kho dữ liệu), thủ tục (luồng dữ liệu) tất cả ở định dạng đáp ứng yêu cầu của người dùng.

Trong khi chuẩn bị thiết kế logic của một hệ thống, nhà phân tích hệ thống xác định nhu cầu của người dùng ở mức độ chi tiết mà hầu như xác định luồng thông tin vào và ra khỏi hệ thống và các nguồn dữ liệu cần thiết. Sơ đồ luồng dữ liệu DFD, sơ đồ E-R được sử dụng.

**3.2 Thiết kế vật lí**

Thiết kế vật lý liên quan đến các quá trình đầu vào và đầu ra thực tế của hệ thống. Nó tập trung vào cách dữ liệu được nhập vào hệ thống, được xác minh, xử lý và hiển thị dưới dạng đầu ra.

Nó tạo ra hệ thống làm việc bằng cách xác định đặc tả thiết kế chỉ định chính xác những gì hệ thống làm. Nó liên quan đến thiết kế giao diện người dùng, thiết kế quy trình và thiết kế dữ liệu.

Nó bao gồm các bước sau:

* Chỉ định phương tiện đầu vào / đầu ra, thiết kế cơ sở dữ liệu và chỉ định các thủ tục sao lưu.
* Lập kế hoạch triển khai hệ thống.
* Lập kế hoạch kiểm tra và cài đặt cũng như chỉ định bất kỳ phần cứng và phần mềm mới nào.
* Cập nhật chi phí, lợi ích, ngày chuyển đổi và các ràng buộc hệ thống.

**3.3 Thiết kế kiến ​​trúc**

Nó còn được gọi là thiết kế cấp cao tập trung vào việc thiết kế kiến ​​trúc hệ thống. Nó mô tả cấu trúc và hành vi của hệ thống. Nó xác định cấu trúc và mối quan hệ giữa các mô-đun khác nhau của quá trình phát triển hệ thống.

**3.4. Thiết kế chi tiết**

Nó tuân theo thiết kế Kiến trúc và tập trung vào sự phát triển của từng mô-đun.

### **4. Mô hình hóa dữ liệu khái niệm**

Nó là đại diện của dữ liệu tổ chức bao gồm tất cả các thực thể và mối quan hệ chính. Các nhà phân tích hệ thống phát triển một mô hình dữ liệu khái niệm cho hệ thống hiện tại hỗ trợ phạm vi và yêu cầu cho hệ thống được đề xuất.

Mục đích chính của mô hình hóa dữ liệu khái niệm là nắm bắt được càng nhiều ý nghĩa của dữ liệu càng tốt. Hầu hết các tổ chức ngày nay sử dụng mô hình dữ liệu khái niệm bằng cách sử dụng mô hình E-R sử dụng ký hiệu đặc biệt để thể hiện càng nhiều ý nghĩa về dữ liệu càng tốt.

**4.1. Mô hình mối quan hệ thực thể (Entity Relationship Model)**

Nó là một kỹ thuật được sử dụng trong thiết kế cơ sở dữ liệu giúp mô tả mối quan hệ giữa các thực thể khác nhau của một tổ chức. Các thuật ngữ được sử dụng trong mô hình E-R.

Thực thể (ENTITY) - Nó chỉ định các đối tượng thực riêng biệt trong một ứng dụng. Ví dụ: nhà cung cấp, mặt hàng, sinh viên, khóa học, giáo viên, v.v.

Quan hệ (RELATIONSHIP) - Chúng là những mối quan hệ phụ thuộc có ý nghĩa giữa các thực thể. Ví dụ, các mặt hàng cung cấp của nhà cung cấp, giáo viên dạy các khóa học, sau đó việc cung cấp và dạy là mối quan hệ.

Thuộc tính (ATTRIBUTES) - Nó chỉ định các thuộc tính của các mối quan hệ. Ví dụ: mã nhà cung cấp, tên sinh viên. Các ký hiệu được sử dụng trong mô hình E-R và ý nghĩa tương ứng của chúng

|  |  |
| --- | --- |
| **Kí tự** | **Ý nghĩa** |
|  | Thực thể |
|  | Thực thể yếu |
|  | Quan hệ |
|  | Quan hệ xác định |
|  | Thuộc tính |
|  | Thuộc tính khóa |
|  | Thuộc tính có nhiều giá trị  (Ex: Phone) |
|  | Thuộc tính đa hợp (Ex: name) |
|  | Thuộc tính kế thừa  (Ex: Từ dob -> age) |
|  | E2 tham gia đầy đủ trong R |
|  | Số lượng tham gia E1:E2 = 1:N |

Ba loại mối quan hệ có thể tồn tại giữa hai tập dữ liệu: một-một, một-nhiều và nhiều-nhiều.

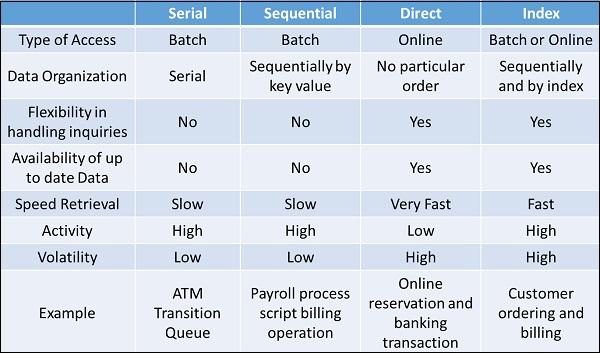
### **5. Tổ chức tệp**

Nó mô tả cách các bản ghi được lưu trữ trong một tệp.

Có bốn phương pháp tổ chức tệp

* Serial - Các bản ghi được lưu trữ theo thứ tự thời gian (theo thứ tự khi chúng được nhập hoặc xuất hiện). Ví dụ - Ghi chi phí điện thoại, giao dịch ATM, Hàng đợi điện thoại.
* Tuần tự (Sequential) - Bản ghi được lưu trữ theo thứ tự dựa trên trường khóa chứa giá trị xác định duy nhất một bản ghi. Ví dụ - Danh bạ điện thoại.
* Trực tiếp (Direct) - Mỗi bản ghi được lưu trữ dựa trên một địa chỉ thực hoặc vị trí trên thiết bị. Địa chỉ được tính từ giá trị được lưu trữ trong trường khóa của bản ghi. Quy trình ngẫu nhiên hoặc thuật toán băm thực hiện chuyển đổi.
* Được đánh chỉ mục (Indexed) - Các bản ghi có thể được xử lý cả tuần tự và không tuần tự bằng cách sử dụng các chỉ mục.

So sánh



### **6. Truy cập tệp**

Người ta có thể truy cập tệp bằng cách sử dụng Truy cập tuần tự hoặc Truy cập ngẫu nhiên. Các phương thức Truy cập tệp cho phép các chương trình máy tính đọc hoặc ghi các bản ghi trong một tệp.

**6.1. Truy cập tuần tự**

Mọi bản ghi trên tệp được xử lý bắt đầu từ bản ghi đầu tiên cho đến khi đạt đến Kết thúc Tệp (EOF). Nó hiệu quả khi một số lượng lớn các bản ghi trong tệp cần được truy cập vào bất kỳ thời điểm nào. Dữ liệu được lưu trữ trên băng (truy cập tuần tự) chỉ có thể được truy cập tuần tự.

**6.2. Truy cập Trực tiếp (Ngẫu nhiên)**

Các bản ghi được định vị bằng cách biết vị trí hoặc địa chỉ thực của chúng trên thiết bị hơn là vị trí của chúng so với các bản ghi khác. Dữ liệu được lưu trữ trên thiết bị CD (truy cập trực tiếp) có thể được truy cập tuần tự hoặc ngẫu nhiên.

**6.3. Các loại tệp được sử dụng trong hệ thống tổ chức**

Sau đây là các loại tệp được sử dụng trong hệ thống tổ chức:

* Tệp chính (Master file) - Nó chứa thông tin hiện tại của một hệ thống. Ví dụ: tệp khách hàng, tệp sinh viên, danh bạ điện thoại.
* Tệp bảng (Table file) - Đây là một loại tệp chính thay đổi không thường xuyên và được lưu trữ ở định dạng bảng. Ví dụ, lưu trữ Zipcode.
* Tệp giao dịch (Transaction file) - Nó chứa thông tin hàng ngày được tạo ra từ các hoạt động kinh doanh. Nó được sử dụng để cập nhật hoặc xử lý tệp chính. Ví dụ: Địa chỉ của nhân viên.
* Tệp tạm thời (Temporary file) - Nó được tạo và sử dụng bất cứ khi nào hệ thống cần.
* Tệp phản chiếu (Mirror file) - Chúng là bản sao chính xác của các tệp khác. Giúp giảm thiểu rủi ro thời gian chết trong trường hợp bản gốc không sử dụng được. Chúng phải được sửa đổi mỗi khi thay đổi tệp gốc.
* Tệp nhật ký (Log file) - Chúng chứa các bản sao của hồ sơ chính và giao dịch để ghi lại mọi thay đổi được thực hiện đối với tệp chính. Nó tạo điều kiện thuận lợi cho việc kiểm tra và cung cấp cơ chế phục hồi trong trường hợp hệ thống bị lỗi.
* Lưu trữ tệp (Archive file) - Sao lưu tệp có chứa phiên bản lịch sử của các tệp khác.

### **7. Kiểm soát tài liệu**

Tài liệu là một quá trình ghi lại thông tin cho bất kỳ mục đích tham khảo hoặc hoạt động nào. Nó giúp ích người dùng, người quản lý và nhân viên CNTT - những người mà cần nó. Điều quan trọng là tài liệu đã chuẩn bị phải được cập nhật thường xuyên để dễ dàng theo dõi tiến trình của hệ thống.

Sau khi cài đặt hệ thống, nếu hệ thống hoạt động không bình thường, thì tài liệu hướng dẫn giúp người quản trị hiểu được luồng dữ liệu trong hệ thống để sửa chữa các sai sót và đưa hệ thống hoạt động.

Lập trình viên hoặc nhà phân tích hệ thống thường tạo tài liệu chương trình và hệ thống. Các nhà phân tích hệ thống thường chịu trách nhiệm chuẩn bị tài liệu để giúp người dùng tìm hiểu hệ thống. Trong các công ty lớn, một nhóm hỗ trợ kỹ thuật bao gồm những người viết kỹ thuật có thể hỗ trợ việc chuẩn bị tài liệu hướng dẫn sử dụng và tài liệu đào tạo.

Ưu điểm:

* Nó có thể giảm thời gian ngừng hoạt động của hệ thống, cắt giảm chi phí và tăng tốc các nhiệm vụ bảo trì.
* Nó cung cấp mô tả rõ ràng về dòng chính thức của hệ thống hiện tại và giúp hiểu loại dữ liệu đầu vào và cách đầu ra có thể được tạo ra.
* Nó cung cấp cách thức giao tiếp hiệu quả giữa người dùng kỹ thuật và không kỹ thuật về hệ thống.
* Nó tạo điều kiện thuận lợi cho việc đào tạo người dùng mới để họ có thể dễ dàng hiểu được quy trình của hệ thống.
* Nó giúp người sử dụng giải quyết các vấn đề như khắc phục sự cố và giúp người quản lý đưa ra các quyết định cuối cùng tốt hơn của hệ thống tổ chức.
* Nó cung cấp khả năng kiểm soát tốt hơn hoạt động bên trong hoặc bên ngoài của hệ thống.

Khi nói đến Thiết kế hệ thống, có bốn tài liệu chính sau đây:

* Tài liệu chương trình
* Tài liệu hệ thống
* Tài liệu hoạt động
* Tài liệu người dùng

**7.1. Tài liệu Chương trình**

Nó mô tả các đầu vào, đầu ra và logic xử lý cho tất cả các mô-đun chương trình.

Quá trình tài liệu chương trình bắt đầu trong giai đoạn phân tích hệ thống và tiếp tục trong quá trình cài đặt.

Tài liệu này hướng dẫn các lập trình viên, những người xây dựng các mô-đun được hỗ trợ tốt bởi các nhận xét và mô tả bên trong và bên ngoài có thể được hiểu và duy trì một cách dễ dàng.

**7.2. Tài liệu Hoạt động**

Tài liệu vận hành chứa tất cả thông tin cần thiết để xử lý và phân phối đầu ra trực tuyến và in. Tài liệu hoạt động phải rõ ràng, ngắn gọn và có sẵn trực tuyến nếu có thể.

Nó bao gồm các thông tin sau:

* Chương trình, nhà phân tích hệ thống, người lập trình và nhận dạng hệ thống.
* Lập lịch thông tin cho đầu ra được in, chẳng hạn như báo cáo, tần suất thực hiện và thời hạn.
* Tệp đầu vào, nguồn, tệp đầu ra và đích của chúng.
* E-mail và danh sách phân phối báo cáo.
* Yêu cầu các biểu mẫu đặc biệt, bao gồm cả biểu mẫu trực tuyến.
* Các thông báo lỗi và thông tin cho các nhà khai thác và các thủ tục khởi động lại.
* Hướng dẫn đặc biệt, chẳng hạn như yêu cầu bảo mật.

**7.4. Tài liệu người dùng**

Nó bao gồm các hướng dẫn và thông tin cho người dùng sẽ tương tác với hệ thống. Ví dụ: hướng dẫn sử dụng, hướng dẫn trợ giúp và hướng dẫn. Tài liệu hướng dẫn sử dụng có giá trị trong việc đào tạo người dùng và cho mục đích tham khảo. Nó phải rõ ràng, dễ hiểu và dễ tiếp cận cho người dùng ở mọi cấp độ.

Người dùng, chủ sở hữu hệ thống, nhà phân tích và lập trình viên, tất cả đều nỗ lực tổng hợp để phát triển hướng dẫn cho người dùng.

Tài liệu người dùng nên bao gồm:

* Tổng quan về hệ thống mô tả rõ ràng tất cả các tính năng, khả năng và hạn chế chính của hệ thống.
* Mô tả nội dung tài liệu nguồn, chuẩn bị, xử lý và mẫu.
* Tổng quan về menu và các tùy chọn màn hình nhập dữ liệu, nội dung và hướng dẫn xử lý.
* Ví dụ về các báo cáo được tạo thường xuyên hoặc có sẵn theo yêu cầu của người dùng, bao gồm cả các mẫu.
* Thông tin về dấu vết kiểm tra và bảo mật.
* Giải trình về trách nhiệm đối với các yêu cầu đầu vào, đầu ra hoặc xử lý cụ thể.
* Thủ tục yêu cầu thay đổi và báo cáo sự cố.
* Ví dụ về các trường hợp ngoại lệ và lỗi.
* Câu hỏi thường gặp (FAQ).
* Giải thích về cách nhận trợ giúp và các thủ tục cập nhật hướng dẫn sử dụng.

**7.5. Tài liệu Hệ thống**

Tài liệu hệ thống đóng vai trò là đặc điểm kỹ thuật của IS và cách thực hiện các mục tiêu của IS. Người dùng, người quản lý và chủ sở hữu IS không bao giờ cần tài liệu hệ thống tham chiếu. Tài liệu hệ thống cung cấp cơ sở để hiểu các khía cạnh kỹ thuật của IS khi các sửa đổi được thực hiện.

* Nó mô tả từng chương trình trong IS và toàn bộ IS.
* Nó mô tả các chức năng của hệ thống, cách chúng được triển khai, mục đích của từng chương trình trong toàn bộ IS liên quan đến thứ tự thực hiện, thông tin được chuyển đến và từ các chương trình cũng như quy trình tổng thể của hệ thống.
* Nó bao gồm các mục từ điển dữ liệu, sơ đồ luồng dữ liệu, mô hình đối tượng, bố cục màn hình, tài liệu nguồn và yêu cầu hệ thống đã khởi tạo dự án.
* Hầu hết các tài liệu hệ thống được chuẩn bị trong giai đoạn phân tích hệ thống và thiết kế hệ thống.
* Trong quá trình cài đặt hệ thống, nhà phân tích phải xem xét tài liệu hệ thống để xác minh rằng tài liệu đó là đầy đủ, chính xác, cập nhật và bao gồm mọi thay đổi được thực hiện trong quá trình cài đặt.

**Q4. Kiến trúc phân tầng và MVC (Layer architecture style & MVC)**

**MVC**

-Mô hình model ,view , controller (MVC) là một kiến ​​trúc phát triển ba lớp nổi tiếng được sử dụng cho các phát triển ứng dụng web. Sau đây là cách sử dụng, lợi thế và thực tiễn của MVC trong quá trình phát triển ứng dụng web. Model view controller (MVC) là một mô hình kiến ​​trúc thường được sử dụng trong các ứng dụng dựa trên web. Nó cung cấp ba lớp chính: model, view, và controller. Nhiều nhà phát triển sử dụng MVC như một mẫu thiết kế tiêu chuẩn. Nó là một framework hoàn chỉnh. MVC cung cấp ba loại lớp:

Model- Các lớp model được sử dụng để thực hiện logic của các miền dữ liệu.Các lớp này được sử dụng để truy xuất, chèn hoặc cập nhật dữ liệu vào cơ sở dữ liệu được liên kết với ứng dụng

View- View xđược sử dụng để chuẩn bị giao diện của ứng dụng của chúng tôi. Bằng cách sử dụng giao diện đó, người dùng tương tác với ứng dụng

Controller- Controller được sử dụng để phản hồi các yêu cầu của người dùng. Các lớp điều khiển thực hiện các hành động mà người dùng yêu cầu. Các lớp này làm việc với các lớp mô hình và chọn dạng xem thích hợp sẽ được hiển thị cho người dùng theo yêu cầu của người dùng.

Kiến trúc mẫu MVC về cơ bản là một kiến ​​trúc ba lớp. Nó phân tách các đặc điểm của ứng dụng. Lớp đầu tiên của nó liên quan đến logic đầu vào của người dùng, lớp thứ hai liên quan đến logic nghiệp vụ và lớp thứ ba được sử dụng để thực hiện logic giao diện người dùng. MVC cung cấp khớp nối rất lỏng lẻo giữa ba lớp này. Mẫu MVC được sử dụng để chỉ định vị trí của mỗi logic trong ứng dụng .

Các mẫu MVC cung cấp cơ sở của sự phát triển song song. Có nghĩa là mỗi lớp của ứng dụng độc lập với nhau, tức là ba developer có thể làm việc trên một ứng dụng đồng thời.Một nhà phát triển sẽ làm việc trên logic đầu vào của người dùng (controller logic), , nhà phát triển khác sẽ làm việc trên logic giao diện người dùng (view), và nhà phát triển thứ ba sẽ làm việc trên logic nghiệp vụ

(model).

-Khi nào sử dụng kiến ​​trúc mẫu MVC:

+Kiến trúc mẫu MVC cung cấp cho chúng ta ý tưởng về sự tách biệt mối quan tâm, nó giúp chúng ta thực hiện sự tách biệt mối quan tâm giữa các lớp model,view,control trong các ứng dụng.

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

+Nếu đang phát triển một ứng dụng có mức nâng rất cao ở phía máy chủ và ít giao tiếp ở phía máy khách thì chúng ta không nên sử dụng kiến ​​trúc mẫu MVC, thay vào đó chúng ta nên sử dụng thiết lập đơn giản như mô hình biểu mẫu dựa trên web. Sau đây là một số đặc điểm sẽ giúp chúng ta có nên sử dụng kiến ​​trúc MVC trong ứng dụng của mình hay không:

i.Ứng dụng cần giao tiếp không đồng bộ ở back-end

ii. Ứng dụng có một chức năng dẫn đến việc không tải lại toàn bộ trang, chẳng hạn như bình luận về một bài đăng trong khi sử dụng Facebook hoặc cuộn vô tận, v.v.

iii. Thao tác dữ liệu chủ yếu ở phía máy khách (trình duyệt) chứ không phải phía máy chủ.

iv. Cùng một loại dữ liệu đang được phân phối theo những cách khác nhau trên một trang duy nhất (điều hướng).

v. Khi ứng dụng có nhiều kết nối không đáng kể được sử dụng để sửa đổi dữ liệu (button, switches).

-Lợi ích của kiến trúc MVC:

1. Kiến trúc MVC giúp chúng ta kiểm soát độ phức tạp của ứng dụng bằng cách chia nó thành ba thành phần tức là model,view,control
2. MVC không sử dụng server-base form, 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.
3. Việc tiếp cận Test driven development được hỗ trợ bởi kiến ​​trúc MVC.
4. MVC sử dụng front control pattern. Front control pattern xử lý nhiều yêu cầu đến bằng giao diện duy nhất (bộ điều khiển). Front control cung cấp khả năng kiểm soát tập trung. Chúng ta chỉ cần cấu hình một controller trong máy chủ web thay vì nhiều bộ điều khiển.
5. Front control cung cấp hỗ trợ thông tin liên lạc định tuyến phong phú để thiết kế ứng dụng web.

-Các tính năng của MVC framework:

Khi chúng ta chia logic của ứng dụng thành ba nhiệm vụ (logic đầu vào, logic nghiệp vụ, logic giao diện), việc kiểm tra các thành phần này sẽ trở nên rất dễ dàng.Khả năng kiểm tra rất nhanh và linh hoạt, vì chúng tôi có thể sử dụng bất kỳ unit testing framework nào tương thích với MVC framework.Nó là framework có thể mở rộng và kết nối.Chúng ta có thể thiết kế các thành phần của ứng dụng theo cách mà chúng có thể dễ dàng thay thế hoặc có thể sửa đổi một cách dễ dàng.Chúng tôi có thể kết nối công cụ xem của riêng mình, chiến lược định tuyến URL, tuần tự hóa ràng buộc phương pháp hành động.Thay vì phụ thuộc vào lớp để tạo đối tượng, chúng ta sử dụng công nghệ dependency injection (DI)cho phép chúng ta đưa đối tượng vào các lớp. Một kỹ thuật khác inversion of control (IOC) được sử dụng để thể hiện sự phụ thuộc giữa các đối tượng, nó chỉ định rằng đối tượng nào cần đối tượng nào khác.MVC cung cấp thành phần ánh xạ URL giúp chúng tôi xây dựng bằng cách sử dụng các URL dễ hiểu và có thể tìm kiếm được. Thay vì sử dụng phần mở rộng tên tệp, MVC hỗ trợ các mẫu đặt tên URL rất hữu ích cho việc search engine optimization (SEO) và representational state transfer (REST).1 vài MVC framework như ASP.NET MVC framework cung cấp cho chúng ta một số tính năng được tích hợp sẵn như xác thực biểu mẫu, quản lý phiên, logic nghiệp vụ giao dịch, bảo mật ứng dụng web, ánh xạ quan hệ đối tượng, bản địa hóa, thành viên và vai trò và ủy quyền URL, v.v. Phổ biến nhất các framework có sẵn ngày nay là backbone.js, ember.js; angle.js và knockout.js.

I.backbone.js:backbone.js framework hữu ích khi ứng dụng của chúng ta cần sự linh hoạt, chúng ta có các yêu cầu không chắc chắn. Ngoài ra, chúng tôi muốn điều chỉnh sự thay đổi trong quá trình phát triển ứng dụng.

II. Ember.js- Khi chúng ta muốn rằng ứng dụng của mình phải tương tác với JSON API hơn là chúng ta nên sử dụng ember.js framework trong ứng dụng của mình.

III. Angular.js- Nếu chúng ta muốn ứng dụng của mình có độ tin cậy và ổn định hơn, chúng ta muốn thử nghiệm rộng rãi cho ứng dụng của mình thì chúng ta nên sử dụng angular.js framework.

IV. Knockout.js- nếu chúng ta muốn tạo một giao diện động phức tạp của ứng dụng thì knockout.js framework sẽ rất hữu ích cho chúng ta.

Mỗi framework đều có những ưu và nhược điểm riêng. Các nhà phát triển có thể sử dụng bất kỳ khuôn khổ nào theo yêu cầu của họ, phù hợp với ứng dụng web của họ.

-Các công cụ và công nghệ sử dụng với MVC:

Có nhiều công cụ và công nghệ có thể được sử dụng để phát triển ứng dụng web với sự trợ giúp của kiến ​​trúc MVC. Tùy thuộc vào sự quan tâm của các nhà phát triển, họ có thể sử dụng bất kỳ công cụ và công nghệ nào để phát triển ứng dụng web. Dưới đây là một số công cụ và công nghệ có thể được sử dụng để phát triển ứng dụng web bằng kiến ​​trúc MVC:

Công cụ

Visual Studio: Visual studio không chỉ là một công cụ mà còn là một môi trường phát triển hoàn chỉnh cung cấp cho chúng tôi cơ sở để tạo ra các loại ứng dụng khác nhau. Khi chúng tôi muốn phát triển ứng dụng bằng ASP.NET MVC framework thì visual studio rất hữu ích cho chúng ta.

1. MYSQL Server- máy chủ quản lý cơ sở dữ liệu quan hệ để duy trì cơ sở dữ liệu .
2. SQL Server- một công cụ cơ sở dữ liệu để duy trì cơ sở dữ liệu giống như máy chủ MYSQL.
3. MYSQL Workbench- một công cụ thiết kế cơ sở dữ liệu.
4. Net Beans- IDE (Môi trường phát triển tích hợp) cung cấp môi trường hoàn chỉnh để phát triển các ứng dụng khác nhau.
5. Máy chủ Glassfish: Máy chủ ứng dụng Java EE.

Công nghệ

* 1. HTML, CSS, JQUERY, AJAX để thiết kế
  2. Các trang máy chủ Servlet và Java (JSP) được sử dụng với Net bean
  3. Công nghệ EJB (Enterprise Java bean)
  4. JSTL (máy chủ Java trang thư viện thẻ tiêu chuẩn)
  5. JPA (API bền vững của Java)
  6. JDBC (kết nối cơ sở dữ liệu Java)
  7. ASP.NET MVC được sử dụng với Visual studio

Có nhiều công cụ và công nghệ khác có thể được sử dụng với kiến trúc MVC, nhưng chúng tôi đã liệt kê một số công cụ và công nghệ mà chúng tôi sẽ sử dụng để xây dựng ứng dụng web của mình bằng kiến trúc MVC.

- Kiến trúc MVC có ảnh hưởng lớn hơn trong thế giới ứng dụng dựa trên web. Nó rất hữu ích cho các nhà phát triển. Các nhà phát triển có thể dễ dàng tạo các ứng dụng web của họ bằng cách sử dụng kiến ​​trúc MVC. Nó làm giảm độ phức tạp của ứng dụng và chia ứng dụng thành ba thành phần chính tức là mô hình, khung nhìn và bộ điều khiển. Nó cung cấp toàn quyền kiểm soát ứng dụng web cho các nhà phát triển. Sử dụng kiến ​​trúc MVC kiểm tra các thành phần khác nhau của ứng dụng trở nên rất dễ dàng. Kiến trúc MVC cung cấp mức độ trừu tượng cao hơn cho các ứng dụng web của họ.

-Kết luận:

Kiến trúc mẫu MVC và các công nghệ tích hợp giống như JSP, Servlet và EJB trên nền tảng của J2EE đã đơn giản hóa quá trình phát triển của các ứng dụng web. Chúng ta có thể tạo ứng dụng web có thể mở rộng, minh bạch và di động với sự trợ giúp của kiến ​​trúc MVC. Kiến trúc MVC cung cấp khái niệm phát triển song song vì nó chia logic của ứng dụng thành ba lớp, vì vậy mỗi nhà phát triển khác nhau có thể làm việc đồng thời trên ba lớp này của cùng một ứng dụng web.

Lớp đầu tiên của kiến ​​trúc MVC là model được sử dụng để tương tác với cơ sở dữ liệu. Về cơ bản, nó là lớp logic của ứng dụng được sử dụng để chèn, truy xuất và cập nhật dữ liệu trong cơ sở dữ liệu. Lớp thứ hai là view liên quan đến giao diện người dùng, qua đó người dùng tương tác với ứng dụng của chúng tôi, cách ứng dụng của chúng tôi sẽ được hiển thị cho người dùng, về cơ bản nó được sử dụng để phát triển giao diện người dùng. Thứ ba sau của kiến ​​trúc MVC là controller được sử dụng để lấy đầu vào từ người dùng. Nó kiểm soát việc cập nhật trạng thái đối tượng mô hình và dữ liệu hiển thị cho người dùng theo đầu vào của người dùng. Người dùng có thể yêu cầu ứng dụng của chúng tôi và có thể nhận được phản hồi thông qua lớp bộ điều khiển.

**3 layer architech style:**

-Ứng dụng kiến trúc "N-Layer" truyền thống

Graphical user interface, application

Description automatically generated

+Các lớp này thường được viết tắt là UI, BLL (Business Logic Layer) và DAL (Data Access Layer). Sử dụng kiến ​​trúc này, người dùng đưa ra các yêu cầu thông qua lớp giao diện người dùng, lớp này chỉ tương tác với BLL. Đến lượt nó, BLL có thể gọi DAL cho các yêu cầu truy cập dữ liệu. Lớp giao diện người dùng không nên trực tiếp đưa ra bất kỳ yêu cầu nào tới DAL, cũng như không nên tương tác trực tiếp với sự bền bỉ thông qua các phương tiện khác. Tương tự như vậy, BLL chỉ nên tương tác với sự bền bỉ bằng cách thông qua DAL. Theo cách này, mỗi lớp có trách nhiệm của riêng nó.

+Một nhược điểm của cách tiếp cận phân lớp truyền thống này là các phụ thuộc thời gian biên dịch chạy từ trên xuống dưới. Nghĩa là, lớp giao diện người dùng phụ thuộc vào BLL, lớp này phụ thuộc vào DAL. Điều này có nghĩa là BLL, thường giữ logic quan trọng nhất trong ứng dụng, phụ thuộc vào chi tiết triển khai truy cập dữ liệu (và thường là sự tồn tại của cơ sở dữ liệu). Kiểm tra logic nghiệp vụ trong một kiến ​​trúc như vậy thường rất khó, đòi hỏi phải có cơ sở dữ liệu kiểm tra. Nguyên tắc đảo ngược phụ thuộc có thể được sử dụng để giải quyết vấn đề này.

+Khi nhu cầu ứng dụng phát triển, các giải pháp triển khai phức tạp và mạnh mẽ hơn có thể được yêu cầu.

+Trong nội bộ, tổ chức dự án này thành nhiều dự án dựa trên trách nhiệm cải thiện khả năng bảo trì của ứng dụng.

+Đơn vị này có thể được mở rộng hoặc mở rộng để tận dụng khả năng mở rộng theo yêu cầu dựa trên đám mây.Mở rộng quy mô có nghĩa là thêm CPU, bộ nhớ, dung lượng đĩa hoặc các tài nguyên khác vào (các) máy chủ lưu trữ ứng dụng của bạn.Mở rộng quy mô có nghĩa là thêm các phiên bản bổ sung của các máy chủ đó, cho dù đây là máy chủ vật lý, máy ảo hay vùng chứa. Khi ứng dụng của bạn được lưu trữ trên nhiều phiên bản, bộ cân bằng tải được sử dụng để chỉ định yêu cầu cho các phiên bản ứng dụng riêng lẻ.

+Cách tiếp cận đơn giản nhất để mở rộng ứng dụng web trong Azure là định cấu hình mở rộng quy mô theo cách thủ công trong Gói dịch vụ ứng dụng của ứng dụng.

-Kiến trúc clean:

+Các ứng dụng tuân theo Nguyên tắc đảo ngược phụ thuộc cũng như nguyên tắc Domain-Driven Design (DDD) có xu hướng đi đến một kiến ​​trúc tương tự. Công trình kiến ​​trúc này đã có nhiều tên tuổi trong những năm qua. Một trong những cái tên đầu tiên là Kiến trúc Lục giác, sau đó là Cổng và Bộ điều hợp. Gần đây hơn, nó được gọi là Kiến trúc onion hoặc Kiến trúc Clean .

+Kiến trúc clean đặt logic nghiệp vụ và mô hình ứng dụng ở trung tâm của ứng dụng. Thay vì có logic nghiệp vụ phụ thuộc vào quyền truy cập dữ liệu hoặc các mối quan tâm về cơ sở hạ tầng khác, sự phụ thuộc này được đảo ngược: cơ sở hạ tầng và chi tiết triển khai phụ thuộc vào Lõi ứng dụng. Chức năng này đạt được bằng cách xác định các giao diện hoặc trừu tượng trong Lõi ứng dụng, sau đó được triển khai bởi các kiểu được xác định trong lớp Cơ sở hạ tầng. Một cách phổ biến để hình dung kiến ​​trúc này là sử dụng một loạt các vòng tròn đồng tâm, tương tự như một củ hành tây.

A picture containing text, electronics

Description automatically generated+Trong sơ đồ này, các phần phụ thuộc chảy về phía vòng tròn trong cùng. Lõi ứng dụng lấy tên từ vị trí của nó ở lõi của sơ đồ này. Và bạn có thể thấy trên sơ đồ rằng Lõi ứng dụng không có phụ thuộc vào các lớp ứng dụng khác. Các thực thể và giao diện của ứng dụng nằm ở trung tâm. Ngay bên ngoài, nhưng vẫn nằm trong Lõi ứng dụng, là các dịch vụ miền, thường triển khai các giao diện được xác định trong vòng tròn bên trong. Bên ngoài Lõi ứng dụng, cả giao diện người dùng và các lớp Cơ sở hạ tầng phụ thuộc vào Lõi ứng dụng, nhưng không phụ thuộc vào nhau (nhất thiết).

Diagram, timeline

Description automatically generated+Lưu ý rằng các mũi tên liền mạch thể hiện sự phụ thuộc vào thời gian biên dịch, trong khi mũi tên đứt nét biểu thị sự phụ thuộc chỉ thời gian chạy. Với kiến trúc sạch, lớp giao diện người dùng hoạt động với các giao diện được xác định trong Lõi ứng dụng tại thời điểm biên dịch và lý tưởng là không nên biết về các kiểu triển khai được xác định trong lớp Cơ sở hạ tầng. Tuy nhiên, tại thời điểm chạy, các kiểu triển khai này là bắt buộc để ứng dụng thực thi, vì vậy chúng cần phải hiện diện và được kết nối với các giao diện Application Core thông qua chèn phụ thuộc.

Diagram

Description automatically generatedSơ đồ kiến trúc cốt lõi của ASP.NET sau Kiến trúc clean.

+Vì Lõi ứng dụng không phụ thuộc vào Cơ sở hạ tầng, nên rất dễ dàng để viết các bài kiểm tra đơn vị tự động cho lớp này.

Diagram

Description automatically generated

Chart

Description automatically generated with medium confidence+Vì lớp giao diện người dùng không có bất kỳ sự phụ thuộc trực tiếp nào vào các loại được xác định trong dự án Cơ sở hạ tầng, nên tương tự như vậy, rất dễ dàng hoán đổi các triển khai, để tạo điều kiện cho thử nghiệm hoặc để đáp ứng các yêu cầu ứng dụng thay đổi. Việc sử dụng và hỗ trợ chèn phụ thuộc được tích hợp trong ASP.NET Core làm cho kiến ​​trúc này trở thành cách thích hợp nhất để cấu trúc các ứng dụng nguyên khối không tầm thường.

+Đối với các ứng dụng nguyên khối, các dự án Lõi ứng dụng, Cơ sở hạ tầng và Giao diện người dùng đều được chạy như một ứng dụng duy nhất.

+Mã tổ chức trong Kiến trúc Clean:

Trong một giải pháp Kiến trúc sạch, mỗi dự án đều có trách nhiệm rõ ràng. Như vậy, các loại nhất định thuộc về mỗi dự án và bạn sẽ thường xuyên tìm thấy các thư mục tương ứng với các loại này trong dự án thích hợp.

\*Lõi ứng dụng

Lõi ứng dụng nắm giữ mô hình kinh doanh, bao gồm các thực thể, dịch vụ và giao diện. Các giao diện này bao gồm các phần tóm tắt cho các hoạt động sẽ được thực hiện bằng Cơ sở hạ tầng, chẳng hạn như truy cập dữ liệu, truy cập hệ thống tệp, cuộc gọi mạng, v.v. Đôi khi các dịch vụ hoặc giao diện được xác định ở lớp này sẽ cần hoạt động với các loại không phải thực thể không có phụ thuộc vào giao diện người dùng hoặc Cơ sở hạ tầng. Chúng có thể được định nghĩa là simple Data Transfer Objects (DTOs).

\*Các loại cốt lõi ứng dụng

* Các thực thể (các lớp mô hình kinh doanh vẫn tồn tại)
* Tổng hợp (nhóm thực thể)
* Giao diện
* Dịch vụ miền
* Thông số kỹ thuật
* Các ngoại lệ tùy chỉnh và các điều khoản bảo vệ
* Sự kiện miền và trình xử lý

\*Cơ sở hạ tầng:

Dự án Cơ sở hạ tầng thường bao gồm triển khai truy cập dữ liệu. Trong một ứng dụng web ASP.NET Core điển hình, những triển khai này bao gồm Entity Framework (EF) DbContext, bất kỳ Migrationđối tượng EF Core nào đã được xác định và các lớp triển khai truy cập dữ liệu. Cách phổ biến nhất để mã triển khai truy cập dữ liệu trừu tượng là thông qua việc sử dụng mẫu thiết kế Kho lưu trữ .Ngoài việc triển khai quyền truy cập dữ liệu, dự án Cơ sở hạ tầng cần bao gồm việc triển khai các dịch vụ phải tương tác với các mối quan tâm về cơ sở hạ tầng. Các dịch vụ này phải triển khai các giao diện được xác định trong Lõi ứng dụng và do đó, Cơ sở hạ tầng phải có tham chiếu đến dự án Lõi ứng dụng.

\*Các loại cơ sở hạ tầng

* Loại lõi EF ( DbContext, Migration)
* Các kiểu triển khai quyền truy cập dữ liệu (Kho lưu trữ)
* Các dịch vụ dành riêng cho cơ sở hạ tầng (ví dụ, FileLoggerhoặc SmtpNotifier)

\*Lớp giao diện người dùng

Lớp giao diện người dùng trong ứng dụng ASP.NET Core MVC là điểm đầu vào cho ứng dụng. Dự án này nên tham chiếu đến dự án Application Core và các loại của nó phải tương tác chặt chẽ với cơ sở hạ tầng thông qua các giao diện được xác định trong Application Core. Không được phép thực hiện trực tiếp hoặc lệnh gọi tĩnh đến các loại lớp Cơ sở hạ tầng trong lớp giao diện người dùng.

\*Các loại lớp giao diện người dùng

* Bộ điều khiển
* Bộ lọc tùy chỉnh
* Phần mềm trung gian tùy chỉnh
* Lượt xem
* ViewModels
* Khởi động

Các Startup class hoặc Program.cs tập tin có trách nhiệm cấu hình ứng dụng, và cho dây lên các loại thực hiện để giao diện. Nơi thực hiện logic này được gọi là gốc cấu thành của ứng dụng và là nơi cho phép chèn phụ thuộc hoạt động bình thường tại thời điểm chạy.

-Monolithic application deployed as a container:

+Bạn có thể xây dựng một Dịch vụ hoặc Ứng dụng Web dựa trên triển khai đơn lẻ và nguyên khối và triển khai nó như một vùng chứa. Trong ứng dụng, nó có thể không phải là nguyên khối nhưng được tổ chức thành một số thư viện, thành phần hoặc lớp. Bên ngoài, nó là một vùng chứa duy nhất với một quy trình, ứng dụng web đơn hoặc dịch vụ duy nhất.

+Để quản lý mô hình này, bạn triển khai một vùng chứa duy nhất để đại diện cho ứng dụng. Để mở rộng quy mô, chỉ cần thêm các bản sao bổ sung có bộ cân bằng tải ở phía trước. Sự đơn giản đến từ việc quản lý một triển khai duy nhất trong một vùng chứa hoặc máy ảo.

+Bạn có thể bao gồm nhiều thành phần / thư viện hoặc các lớp bên trong mỗi vùng chứa.Tuy nhiên, tuân theo nguyên tắc vùng chứa “một vùng chứa thực hiện một việc và thực hiện nó trong một quá trình “, mô hình nguyên khối có thể là một xung đột.

+Nhược điểm của phương pháp này xuất hiện nếu / khi ứng dụng phát triển, đòi hỏi nó phải mở rộng quy mô. Nếu toàn bộ ứng dụng mở rộng quy mô, nó không thực sự là một vấn đề. Tuy nhiên, trong hầu hết các trường hợp, một vài phần của ứng dụng là điểm nghẹt thở yêu cầu mở rộng quy mô, trong khi các thành phần khác được sử dụng ít hơn.

+Ngoài vấn đề “scale everything”, các thay đổi đối với một thành phần duy nhất yêu cầu kiểm tra lại toàn bộ ứng dụng và triển khai lại hoàn chỉnh tất cả các phiên bản

+Cách tiếp cận nguyên khối là phổ biến và nhiều tổ chức đang phát triển với cách tiếp cận kiến ​​trúc này. Nhiều người đang có kết quả đủ tốt, trong khi những người khác đang đạt đến giới hạn. Nhiều người đã thiết kế ứng dụng của họ theo mô hình này, vì các công cụ và cơ sở hạ tầng quá khó để xây dựng kiến ​​trúc hướng dịch vụ (SOA) và họ không thấy cần thiết cho đến khi ứng dụng phát triển. Nếu bạn thấy mình đang đạt đến giới hạn của cách tiếp cận nguyên khối, thì việc chia nhỏ ứng dụng để cho phép nó tận dụng tốt hơn các vùng chứa và microservices có thể là bước hợp lý tiếp theo.

+Việc triển khai các ứng dụng nguyên khối trong Microsoft Azure có thể đạt được bằng cách sử dụng các máy ảo chuyên dụng cho từng trường hợp. Sử dụng Bộ quy mô máy ảo Azure , bạn có thể dễ dàng mở rộng các máy ảo. Azure App Services có thể chạy các ứng dụng nguyên khối và dễ dàng mở rộng các phiên bản mà không cần phải quản lý các máy ảo. Azure App Services cũng có thể chạy các phiên bản Docker container, đơn giản hóa việc triển khai. Sử dụng Docker, bạn có thể triển khai một máy ảo duy nhất làm máy chủ lưu trữ Docker và chạy nhiều phiên bản.

+Việc triển khai tới các máy chủ khác nhau có thể được quản lý bằng các kỹ thuật triển khai truyền thống. Các máy chủ Docker có thể được quản lý bằng các lệnh như docker chạy được thực hiện theo cách thủ công hoặc thông qua tự động hóa như đường ống Phân phối liên tục (CD).

+Có những lợi ích khi sử dụng vùng chứa để quản lý việc triển khai ứng dụng nguyên khối. Việc mở rộng các phiên bản của vùng chứa nhanh hơn và dễ dàng hơn nhiều so với việc triển khai các máy ảo bổ sung. Ngay cả khi sử dụng bộ quy mô máy ảo để mở rộng máy ảo, chúng cũng cần thời gian để tạo. Khi được triển khai dưới dạng phiên bản ứng dụng, cấu hình của ứng dụng được quản lý như một phần của máy ảo

+Triển khai các bản cập nhật dưới dạng hình ảnh Docker nhanh hơn nhiều và hiệu quả về mạng. Hình ảnh Docker thường bắt đầu trong vài giây, tốc độ phát hành nhanh hơn. Xé một phiên bản Docker dễ dàng như phát docker stoplệnh, thường hoàn thành trong vòng chưa đầy một giây.

+Vì các vùng chứa vốn dĩ là bất biến theo thiết kế, bạn không bao giờ cần phải lo lắng về các máy ảo bị hỏng, trong khi các tập lệnh cập nhật có thể quên tính đến một số cấu hình hoặc tệp cụ thể còn lại trên đĩa.

+Bạn có thể sử dụng vùng chứa Docker để triển khai đơn nguyên các ứng dụng web đơn giản hơn. Cách tiếp cận này cải thiện sự tích hợp liên tục và các đường ống triển khai liên tục, đồng thời giúp đạt được thành công từ việc triển khai đến sản xuất.

+Kiến trúc dựa trên microservices có nhiều lợi ích, nhưng những lợi ích đó đi kèm với cái giá phải trả là sự phức tạp ngày càng tăng. Trong một số trường hợp, chi phí lớn hơn lợi ích, do đó, một ứng dụng triển khai nguyên khối chạy trong một vùng chứa duy nhất hoặc chỉ trong một vài vùng chứa là một lựa chọn tốt hơn.

+Một ứng dụng nguyên khối có thể không dễ dàng phân tách thành các microservices được phân tách rõ ràng. Các microservices phải hoạt động độc lập với nhau để cung cấp một ứng dụng linh hoạt hơn. Nếu bạn không thể cung cấp các phần tính năng độc lập của ứng dụng, việc tách nó ra chỉ làm tăng thêm sự phức tạp.

+Một ứng dụng có thể chưa cần mở rộng các tính năng một cách độc lập. Nhiều ứng dụng, khi chúng cần mở rộng ra ngoài một phiên bản duy nhất, có thể làm như vậy thông qua quá trình nhân bản toàn bộ phiên bản đó tương đối đơn giản. Công việc bổ sung để tách ứng dụng thành các dịch vụ rời rạc mang lại lợi ích tối thiểu khi việc mở rộng các phiên bản đầy đủ của ứng dụng rất đơn giản và tiết kiệm chi phí.

+Trong giai đoạn đầu phát triển một ứng dụng, bạn có thể không có ý tưởng rõ ràng về ranh giới chức năng tự nhiên ở đâu. Khi bạn phát triển một sản phẩm khả thi tối thiểu, sự phân tách tự nhiên có thể chưa xuất hiện. Một số điều kiện này có thể là tạm thời. Bạn có thể bắt đầu bằng cách tạo một ứng dụng nguyên khối và sau đó tách một số tính năng để được phát triển và triển khai dưới dạng microservices. Các điều kiện khác có thể cần thiết đối với không gian có vấn đề của ứng dụng, có nghĩa là ứng dụng có thể không bao giờ được chia thành nhiều microservices.

+Việc tách một ứng dụng thành nhiều quy trình rời rạc cũng dẫn đến chi phí cao. Có nhiều phức tạp hơn trong việc tách các tính năng thành các quy trình khác nhau. Các giao thức truyền thông trở nên phức tạp hơn. Thay vì gọi phương thức, bạn phải sử dụng liên lạc không đồng bộ giữa các dịch vụ. Khi chuyển sang kiến trúc microservices, bạn cần thêm nhiều khối xây dựng được triển khai trong phiên bản microservices của ứng dụng eShopOnContainers: xử lý bus sự kiện, khả năng phục hồi thông báo và thử lại, tính nhất quán cuối cùng, v.v.

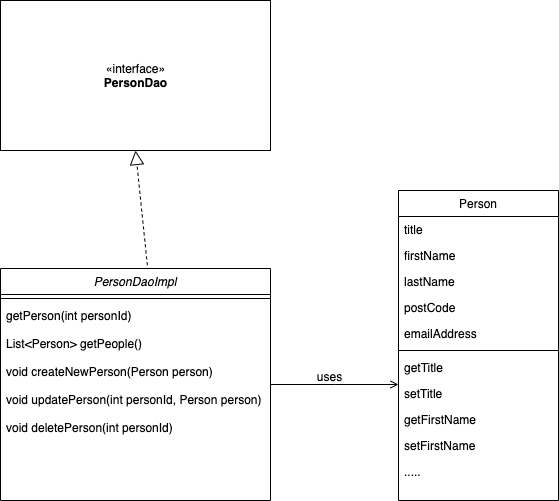
**Q5)** **DAO? Why? class diagram & code for DAO. Review Appendix B1 & Explain diagram.**

DAO là từ viết tắt của “**Data Access Object -**Đối tượng truy cập dữ liệu”.

DAO là một ‘đối tượng’ đóng gói các hoạt động truy cập dữ liệu. Trong lập trình hướng đối tượng; nó là một đối tượng cho phép hoặc trừu tượng hóa phần tử truy cập dữ liệu từ / đến một hệ thống dữ liệu bên ngoài.

Hệ thống dữ liệu này có thể là bất kỳ nguồn dữ liệu chung nào hoặc có thể là một loại cơ sở dữ liệu rất cụ thể. Khi mô hình này xuất hiện; người ta thường sử dụng điều này như một mẫu để trừu tượng hóa tất cả các quyền truy cập cơ sở dữ liệu. Cơ sở dữ liệu thường là RDBMS như MySQL, Oracle RDBMS,PostgreSQL, DB2,...

**Structure — Class Diagram**



có ba thành phần tham gia cụ thể là PersonDaoInterface, the implementing class (PersonDaoImpl)

Value Object (VO) / Entity Object (EO) / Data Transfer Object (DTO) — Person đóng gói dữ liệu trong một đối tượng.

VO, EO hoặc DTO thường được sử dụng thay thế cho nhau trong một số ngữ cảnh nhưng chỉ đơn giản là có các đối tượng hoạt động như một cơ chế để truyền dữ liệu từ hoặc đến cơ sở dữ liệu.

\* **The VO / EO / DTO**

****

****

Đóng gói dữ liệu đại diện cho một thực thể ‘Person’. Nó chỉ chứa một loạt các trường đại diện cho các thuộc tính Person với các getters / setters tương ứng để truy xuất và thiết lập dữ liệu trên đối tượng.

**\*The DAO Interface**

Giả định sẽ có một giao diện Java PersonDao xác định danh sách các hoạt động mà DAO triển khai sẽ thực hiện.



DAO thường ở dạng một lớp trên mỗi bảng. Như trong ví dụ trên; chúng tôi giả định rằng chúng tôi có một bảng Cơ sở dữ liệu được gọi là Person trong cơ sở dữ liệu. Lớp PersonDaoImpl triển khai sẽ đại diện cho bảng Person cụ thể đó.

**\*The Implementing Class**  
Mỗi phương thức trong DAO sẽ chứa mã thực hiện những gì phương thức hiển thị.











Đầu tiên thiết lập kết nối cơ sở dữ liệu và sau đó sử dụng câu truy vấn để gửi truy vấn SQL hoặc cập nhật SQL đến cơ sở dữ liệu.Code sử dụng thư viện JDBC để kết nối CSDL. JDBC là thư viện nội bộ của Java cho phép kết nối cơ sở dữ liệu với cơ sở dữ liệu quan hệ.

Tiếp theo, sử dụng try catch chủ yếu để xử lý lỗi (vì đang kết nối với cơ sở dữ liệu là hệ thống bên ngoài nên nó có thể bị lỗi).

**Q6)** **Database design**

**\*The database design process(Quy trình thiết kế cơ sở dữ liệu)**

Cơ sở dữ liệu có cấu trúc ổn:

* Tiết kiệm dung lượng ổ đĩa bằng cách loại bỏ dữ liệu dư thừa.
* Duy trì độ chính xác và tính toàn vẹn của dữ liệu.
* Cung cấp quyền truy cập vào dữ liệu theo những cách hữu ích.

Thiết kế một cơ sở dữ liệu hữu ích, hiệu quả là vấn đề tuân theo quy trình thích hợp, bao gồm các giai đoạn sau:

1. Phân tích yêu cầu hoặc xác định mục đích của cơ sở dữ liệu của mình

2. Cấu trúc dữ liệu thành bảng.

3. Chỉ định khóa chính và phân tích quan hệ.

4. Chuẩn hóa các bảng.

**\* Requirements analysis: identifying the purpose of the database(Phân tích yêu cầu: xác định mục đích của cơ sở dữ liệu)**

Hiểu được mục đích của cơ sở dữ liệu của bạn sẽ cho bạn các lựa chọn trong suốt quá trình thiết kế. Đảm bảo rằng bạn xem xét cơ sở dữ liệu từ mọi góc độ.

Dưới đây là một số cách để thu thập thông tin trước khi tạo cơ sở dữ liệu:

* Phỏng vấn những người sẽ sử dụng nó
* Phân tích các hình thức kinh doanh, chẳng hạn như hóa đơn, bảng chấm công, khảo sát
* Kết hợp thông qua bất kỳ hệ thống dữ liệu hiện có nào.

Bắt đầu bằng cách thu thập bất kỳ dữ liệu hiện có nào sẽ được đưa vào cơ sở dữ liệu. Sau đó, liệt kê các loại dữ liệu bạn muốn lưu trữ và các thực thể hoặc người, sự vật, vị trí và sự kiện mà những dữ liệu đó mô tả, như sau:

Customers

* Name
* Address
* City, State, Zip
* Email address

Products

* Name
* Price
* Quantity in stock
* Quantity on order

Orders

* Order ID
* Sales representative
* Date
* Product(s)
* Quantity
* Price
* Total

Thông tin này sau này sẽ trở thành một phần của từ điển dữ liệu, nó phác thảo các bảng và trường trong cơ sở dữ liệu. Đảm bảo chia nhỏ thông tin thành những phần hữu ích nhỏ nhất. Ví dụ: hãy xem xét tách địa chỉ đường phố khỏi quốc gia để sau này bạn có thể lọc các cá nhân theo quốc gia cư trú của họ. Ngoài ra, tránh đặt cùng một điểm dữ liệu trong nhiều bảng, điều này làm tăng thêm độ phức tạp không cần thiết.

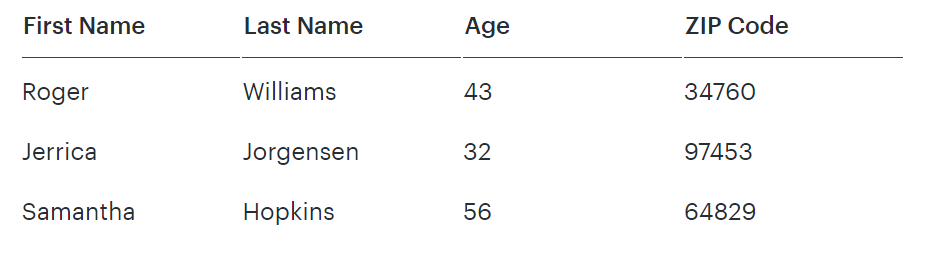
Khi biết cơ sở dữ liệu sẽ bao gồm những loại dữ liệu nào, dữ liệu đó đến từ đâu và nó sẽ được sử dụng như thế nào, bạn đã sẵn sàng để bắt đầu lập kế hoạch cho cơ sở dữ liệu thực tế.

**\*Database structure: the building blocks of a database(Cấu trúc cơ sở dữ liệu: các khối xây dựng của cơ sở dữ liệu)**

Trong cơ sở dữ liệu, dữ liệu liên quan được nhóm thành các bảng, mỗi bảng bao gồm các hàng (còn được gọi là bộ dữ liệu) và các cột, giống như một bảng tính.

Để chuyển đổi danh sách dữ liệu của bạn thành bảng, hãy bắt đầu bằng cách tạo bảng cho từng loại thực thể, chẳng hạn như sản phẩm, bán hàng, khách hàng và đơn đặt hàng. Đây là một ví dụ:

Mỗi hàng của bảng được gọi là một bản ghi. Hồ sơ bao gồm dữ liệu về một cái gì đó hoặc một người nào đó, chẳng hạn như một khách hàng cụ thể. Ngược lại, các cột (còn được gọi là trường hoặc thuộc tính) chứa một loại thông tin duy nhất xuất hiện trong mỗi bản ghi, chẳng hạn như địa chỉ của tất cả các khách hàng được liệt kê trong bảng.

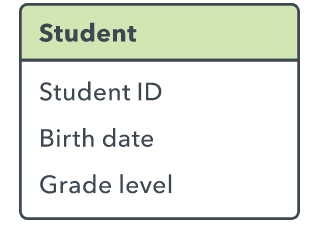


Để giữ cho dữ liệu nhất quán từ bản ghi này sang bản ghi tiếp theo, hãy gán kiểu dữ liệu thích hợp cho mỗi cột. Các kiểu dữ liệu phổ biến bao gồm:

* CHAR - a specific length of text
* VARCHAR - text of variable lengths
* TEXT - large amounts of text
* INT - positive or negative whole number
* FLOAT, DOUBLE - can also store floating point numbers
* BLOB-dữ liệu nhị phân.

Một số hệ thống quản lý cơ sở dữ liệu cũng cung cấp kiểu dữ liệu Autonumber, tự động tạo một số duy nhất trong mỗi hàng.

Với mục đích tạo sơ đồ mối quan hệ thực thể,mỗi bảng trở thành một box trong diagram. Tiêu đề của mỗi hộp phải cho biết dữ liệu trong bảng đó mô tả những gì, trong khi các thuộc tính được liệt kê bên dưới, như sau:



Cuối cùng, bạn nên quyết định thuộc tính nào hoặc các thuộc tính nào sẽ đóng vai trò là khóa chính cho mỗi bảng, nếu có. Khóa chính (PK) là số nhận dạng duy nhất cho một thực thể nhất định, có nghĩa là bạn có thể chọn ra một khách hàng chính xác ngay cả khi bạn chỉ biết giá trị đó.

Các thuộc tính được chọn làm khóa chính phải là duy nhất, không thay đổi và luôn hiện diện (không bao giờ NULL hoặc trống). Vì lý do này, số thứ tự và tên người dùng có thể tạo nên khóa chính , trong khi số điện thoại hoặc địa chỉ đường phố thì không. Bạn cũng có thể sử dụng nhiều trường kết hợp làm khóa chính (đây được gọi là khóa tổng hợp).

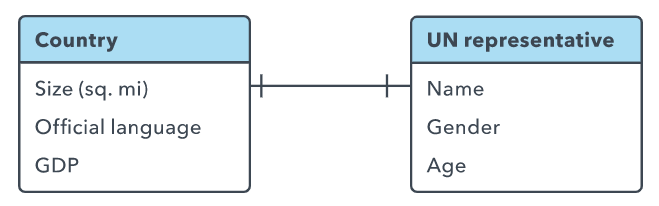
Nên ước tính kích thước của cơ sở dữ liệu để đảm bảo rằng bạn có thể nhận được mức hiệu suất và không gian lưu trữ mà nó sẽ yêu cầu

**Creating relationships between entities(Tạo mối quan hệ giữa các thực thể).**

Mỗi thực thể có thể có mối quan hệ với mọi thực thể khác, nhưng những mối quan hệ đó thường là một trong ba loại:

\*One-to-one relationships:

Khi chỉ có một bản sao của Đối tượng A cho mỗi bản sao của Đối tượng B, chúng được cho là có mối quan hệ một đối một (thường được viết là 1: 1). Bạn có thể chỉ ra loại mối quan hệ này trong sơ đồ ER với một đường có dấu gạch ngang ở mỗi đầu:



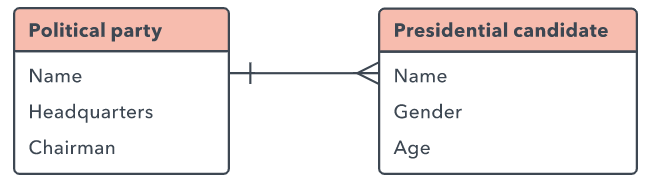
Trừ khi bạn có lý do chính đáng để không làm vậy, mối quan hệ 1: 1 thường chỉ ra rằng bạn nên kết hợp dữ liệu của hai bảng thành một bảng duy nhất

Tuy nhiên, bạn có thể muốn tạo các bảng có mối quan hệ 1: 1 trong các trường hợp cụ thể. Nếu bạn có một trường có dữ liệu tùy chọn, chẳng hạn như “mô tả”, bị trống cho nhiều bản ghi, bạn có thể di chuyển tất cả các mô tả vào bảng riêng của chúng, loại bỏ không gian trống và cải thiện hiệu suất cơ sở dữ liệu.

Để đảm bảo rằng dữ liệu khớp chính xác, bạn phải include ít nhất một cột giống hệt nhau trong mỗi bảng, rất có thể là khóa chính.

\*One-to-many relationships

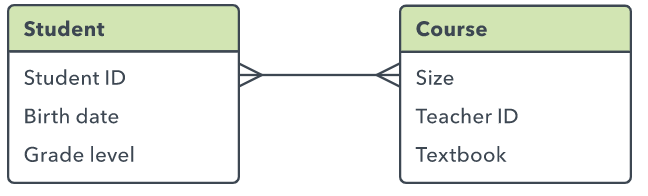
Những mối quan hệ này xảy ra khi một bản ghi trong một bảng được liên kết với nhiều mục nhập trong một bảng khác. Ví dụ: một khách hàng có thể đã đặt nhiều đơn đặt hàng hoặc một khách hàng quen có thể gửi nhiều cuốn sách từ thư viện ra cùng một lúc. Mối quan hệ một-nhiều (1: M) được biểu thị bằng cái được gọi là "Ký hiệu vết chân chim", như trong ví dụ sau:



Để triển khai mối quan hệ 1: M khi bạn thiết lập cơ sở dữ liệu, chỉ cần thêm khóa chính từ phía "một" của mối quan hệ làm thuộc tính trong bảng kia. Khi một khóa chính được liệt kê trong một bảng khác theo cách này, nó được gọi là khóa ngoại. Bảng ở phía "1" của mối quan hệ được coi là bảng mẹ đối với bảng con ở phía bên kia.

\*Many-to-many relationships

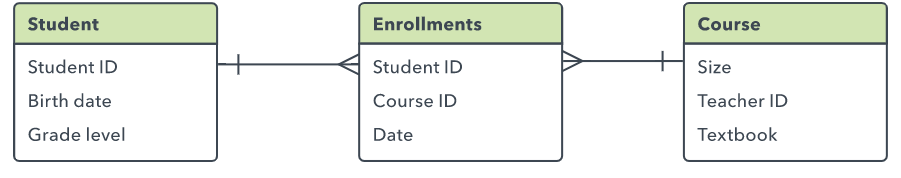
Khi nhiều thực thể từ một bảng có thể được liên kết với nhiều thực thể trong bảng khác, chúng được cho là có mối quan hệ nhiều-nhiều (M: N). Điều này có thể xảy ra trong trường hợp học sinh và lớp học, vì một học sinh có thể học nhiều lớp và một lớp có thể có nhiều học sinh.



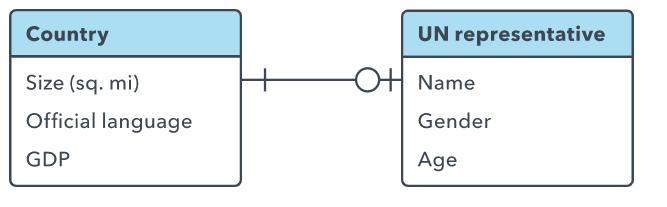
Rất tiếc, không thể trực tiếp triển khai loại mối quan hệ này trong cơ sở dữ liệu. Thay vào đó, bạn phải chia nó thành hai mối quan hệ một-nhiều.

Để làm như vậy, hãy tạo một thực thể mới giữa hai bảng đó. Nếu mối quan hệ M: N tồn tại giữa bán hàng và sản phẩm, bạn có thể gọi thực thể mới đó là “sản phẩm đã bán”, vì nó sẽ hiển thị nội dung của từng đợt bán hàng. Cả bảng bán hàng và sản phẩm sẽ có mối quan hệ 1: M với các sản phẩm được bán. Loại thực thể đi giữa này được gọi là bảng liên kết, thực thể liên kết hoặc bảng nối trong các mô hình khác nhau.

Mỗi bản ghi trong bảng liên kết sẽ khớp với nhau hai thực thể trong các bảng lân cận (nó cũng có thể bao gồm thông tin bổ sung). Ví dụ: bảng liên kết giữa sinh viên và lớp học có thể trông giống như sau:

Mandatory or not?

Một cách khác để phân tích mối quan hệ là xem xét mặt nào của mối quan hệ phải tồn tại để mặt kia tồn tại. Bên không bắt buộc có thể được đánh dấu bằng một vòng tròn trên dòng có dấu gạch ngang. Ví dụ, một quốc gia phải tồn tại để quốc gia đó có đại diện tại Liên Hợp Quốc, nhưng điều ngược lại là không đúng:



Hai thực thể có thể phụ thuộc lẫn nhau (một thực thể không thể tồn tại nếu không có thực thể kia).

\*Recursive relationships(Mối quan hệ đệ quy)

Đôi khi một bảng chỉ lại chính nó. Ví dụ: một bảng nhân viên có thể có thuộc tính "người quản lý" đề cập đến một cá nhân khác trong cùng bảng đó. Đây được gọi là mối quan hệ đệ quy.

Redundant relationships(Mối quan hệ dư thừa)

Mối quan hệ thừa là mối quan hệ được thể hiện nhiều hơn một lần. Thông thường, bạn có thể xóa một trong các mối quan hệ mà không làm mất bất kỳ thông tin quan trọng nào. Ví dụ: nếu một thực thể “sinh viên” có mối quan hệ trực tiếp với một thực thể khác được gọi là “giáo viên” nhưng cũng có mối quan hệ gián tiếp với giáo viên thông qua “lớp học”, bạn muốn xóa mối quan hệ giữa “sinh viên” và “giáo viên”. Tốt hơn là xóa mối quan hệ đó vì cách duy nhất mà học sinh được chỉ định cho giáo viên là thông qua các lớp học.

**Database normalization(Chuẩn hóa cơ sở dữ liệu)**

Có thể áp dụng các quy tắc chuẩn hóa để đảm bảo các bảng được cấu trúc chính xác.

Không phải tất cả các cơ sở dữ liệu đều là ứng cử viên tốt để chuẩn hóa. Nói chung, cơ sở dữ liệu xử lý giao dịch trực tuyến (viết tắt là OLTP), trong đó người dùng quan tâm đến việc CRUD bản ghi, nên được chuẩn hóa.

Cơ sở dữ liệu xử lý phân tích trực tuyến (OLAP) ưu tiên phân tích và báo cáo có thể hoạt động tốt hơn với mức độ không chuẩn hóa, vì trọng tâm là tốc độ tính toán. Chúng bao gồm các ứng dụng hỗ trợ quyết định trong đó dữ liệu cần được phân tích nhanh chóng nhưng không được thay đổi.

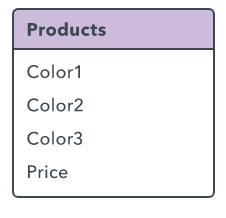
Mỗi biểu mẫu, hoặc mức độ chuẩn hóa, bao gồm các quy tắc được liên kết với các biểu mẫu thấp hơn.

\*First normal form

Dạng chuẩn đầu tiên (viết tắt là 1NF) chỉ định rằng mỗi ô trong bảng chỉ có thể có một giá trị, không bao giờ là danh sách các giá trị, vì vậy một bảng như thế này không tuân thủ:



Bạn có thể muốn giải quyết vấn đề này bằng cách tách dữ liệu đó thành các cột bổ sung, nhưng điều đó cũng trái với quy tắc: một bảng có các nhóm thuộc tính lặp lại hoặc có liên quan chặt chẽ với nhau không đáp ứng dạng thông thường đầu tiên. Ví dụ: bảng dưới đây không tuân thủ:



Thay vào đó, hãy chia dữ liệu thành nhiều bảng hoặc nhiều bản ghi cho đến khi mỗi ô chỉ chứa một giá trị và không có cột thừa. Tại thời điểm đó, dữ liệu được cho là nguyên tử, hoặc được chia nhỏ thành kích thước hữu ích nhỏ nhất.

\*Second normal form

Dạng chuẩn thứ hai (2NF) yêu cầu mỗi thuộc tính phải hoàn toàn phụ thuộc vào toàn bộ khóa chính. Điều đó có nghĩa là mỗi thuộc tính nên phụ thuộc trực tiếp vào khóa chính, thay vì gián tiếp thông qua một số thuộc tính khác.

Ví dụ: thuộc tính “tuổi” phụ thuộc vào “ngày sinh” mà lần lượt phụ thuộc vào “studentID” được cho là có phụ thuộc hàm một phần và bảng chứa các thuộc tính này sẽ không đáp ứng được dạng chuẩn thứ hai.

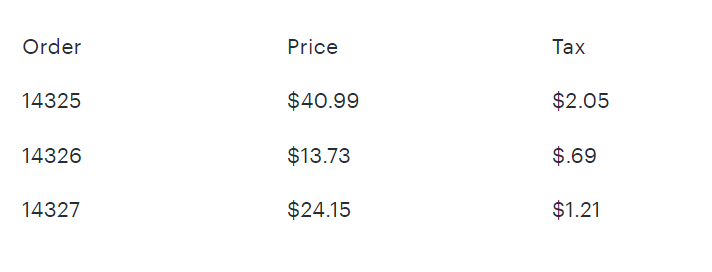
Hơn nữa, một bảng có khóa chính được tạo thành từ nhiều trường vi phạm dạng chuẩn thứ hai nếu một hoặc nhiều trường khác không phụ thuộc vào mọi phần của khóa.Do đó, một bảng có các trường này sẽ không đáp ứng dạng Second normal form, vì thuộc tính "product name" phụ thuộc vào ID sản phẩm chứ không phụ thuộc vào số đơn đặt hàng:

* Order number (primary key)
* Product ID (primary key)
* Product name

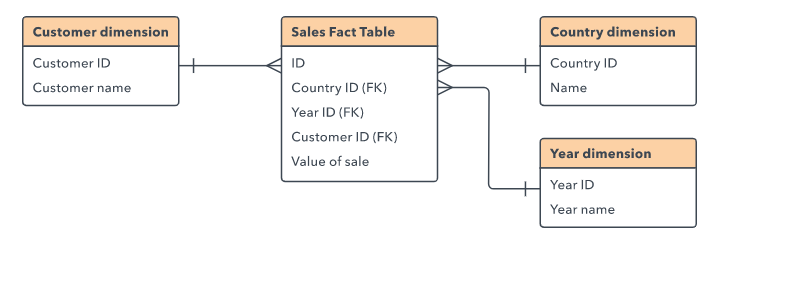
\*Third normal form

Dạng chuẩn thứ ba (3NF) bổ sung vào các quy tắc này yêu cầu rằng mọi cột không phải khóa phải độc lập với mọi cột khác. Nếu việc thay đổi một giá trị trong một cột không phải là khóa làm cho một giá trị khác thay đổi, thì bảng đó không đáp ứng dạng chuẩn thứ ba.

Điều này giúp bạn không lưu trữ bất kỳ dữ liệu bắt nguồn nào trong bảng, chẳng hạn như cột "thuế" bên dưới, trực tiếp phụ thuộc vào tổng giá của đơn đặt hàng:



**Multidimensional data(Dữ liệu đa chiều)**

Một số người dùng có thể muốn truy cập nhiều thứ của một loại dữ liệu, đặc biệt là trong cơ sở dữ liệu OLAP. Ví dụ: họ có thể muốn biết doanh số bán hàng theo khách hàng, tiểu bang và tháng. Trong trường hợp này, cách tốt nhất là tạo một bảng dữ kiện trung tâm mà các bảng khách hàng, tiểu bang và tháng khác có thể tham khảo, như sau: 

Data integrity rules(Quy tắc toàn vẹn dữ liệu)

Bạn cũng nên cấu hình cơ sở dữ liệu của mình để xác thực dữ liệu theo các quy tắc thích hợp. Nhiều hệ thống quản lý cơ sở dữ liệu, chẳng hạn như Microsoft Access, thực thi một số quy tắc này một cách tự động.

Quy tắc toàn vẹn thực thể nói rằng khóa chính không bao giờ có thể là NULL. Nếu khóa được tạo thành từ nhiều cột, không cột nào trong số chúng có thể là NULL. Nếu không, nó có thể không xác định duy nhất bản ghi.

Quy tắc toàn vẹn tham chiếu yêu cầu mỗi khóa ngoại được liệt kê trong một bảng phải được khớp với một khóa chính trong bảng mà nó tham chiếu. Nếu khóa chính thay đổi hoặc bị xóa, những thay đổi đó sẽ cần được thực hiện ở bất kỳ nơi nào khóa đó được tham chiếu trong cơ sở dữ liệu.

Các quy tắc toàn vẹn logic nghiệp vụ đảm bảo rằng dữ liệu phù hợp với các tham số logic nhất định. Ví dụ, một cuộc hẹn sẽ phải rơi vào giờ làm việc bình thường.

Adding indexes and views

Index về cơ bản là một bản sao được sắp xếp của một hoặc nhiều cột, với các giá trị theo thứ tự tăng dần hoặc giảm dần. Thêm một index cho phép người dùng tìm các bản ghi nhanh hơn. Thay vì sắp xếp lại cho từng truy vấn, hệ thống có thể truy cập các bản ghi theo thứ tự được chỉ định bởi index.

Mặc dù các index tăng tốc độ truy xuất dữ liệu, nhưng chúng có thể làm chậm việc chèn, cập nhật và xóa, vì index phải được xây dựng lại bất cứ khi nào bản ghi được thay đổi.

Một view chỉ đơn giản là một truy vấn được lưu trên dữ liệu. Chúng có thể kết hợp hữu ích dữ liệu từ nhiều bảng hoặc hiển thị một phần của bảng.

**Extended properties(Thuộc tính mở rộng)**

Khi bạn đã hoàn thành bố cục cơ bản, bạn có thể tinh chỉnh cơ sở dữ liệu với các thuộc tính mở rộng, chẳng hạn như văn bản hướng dẫn, dấu hiệu nhập và quy tắc định dạng áp dụng cho một schema, view hoặc cột cụ thể. Ưu điểm là, bởi vì các quy tắc này được lưu trữ trong chính cơ sở dữ liệu, nên việc trình bày dữ liệu sẽ nhất quán trên nhiều chương trình truy cập dữ liệu.

**SQL and UML**

Ngôn ngữ mô hình hóa hợp nhất (UML) là một cách trực quan khác để diễn đạt các hệ thống phức tạp được tạo ra bằng ngôn ngữ hướng đối tượng. Một số khái niệm được đề cập trong hướng dẫn này được biết đến trong UML dưới các tên khác nhau. Ví dụ, một thực thể được gọi là một lớp trong UML.

**Database management systems**

Nhiều lựa chọn thiết kế bạn sẽ thực hiện tùy thuộc vào hệ quản trị cơ sở dữ liệu bạn sử dụng. Một số hệ thống phổ biến nhất bao gồm:

* Oracle DB
* MySQL
* Microsoft SQL Server
* PostgreSQL
* IBM DB2

Khi được lựa chọn, hãy chọn một hệ thống quản lý cơ sở dữ liệu thích hợp dựa trên chi phí, hệ điều hành, tính năng và hơn thế nữa.

**Q7) Interface design**

1. UI là gì? Thiết kế UI là gì?

• UI (user interface) – Giao diện người dùng là nơi tương tác giữa người dùng và thiết bị điện tử: những màn hình mà người dùng tương tác trên app, những nút họ nhấp vào khi sử dụng wesite…

• Thiết kế giao diện người dùng (UI) - là quy trình mà các designer sử dụng để xây dựng giao diện (text, nút bấm, icon, màu sắc, khoảng cách, hình ảnh, màn hình ứng dụng…) trong phần mềm hoặc thiết bị máy tính, tập trung vào bề ngoài hoặc style. Các nhà thiết kế hướng tới việc tạo ra các giao diện mà người dùng cảm thấy dễ sử dụng và hài lòng.

• Thiết kế giao diện người dùng cho người dùng

Giao diện người dùng là điểm truy cập nơi người dùng tương tác với các thiết kế. Chúng có ba định dạng:

+ Giao diện người dùng đồ họa (GUI) - Người dùng tương tác với các biểu diễn trực quan trên bảng điều khiển kỹ thuật số. Màn hình nền của máy tính là GUI.

+ Giao diện điều khiển bằng giọng nói (VUI) - Người dùng tương tác với các giao diện này thông qua giọng nói của họ. Hầu hết các trợ lý thông minh

ví dụ: Siri trên iPhone và Alexa trên các thiết bị Amazon đều là VUI.

+ Giao diện dựa trên cử chỉ - Người dùng tham gia vào không gian thiết kế 3D thông qua các chuyển động của cơ thể: ví dụ: trong trò chơi thực tế ảo (VR) .

2. Để thiết kế giao diện người dùng tốt nhất, bạn nên chú ý:

• Người dùng đánh giá thiết kế nhanh chóng và quan tâm đến khả năng sử dụng và khả năng thích ứng. Họ không quan tâm đến thiết kế của bạn , mà quan tâm đến việc hoàn thành nhiệm vụ của họ một cách dễ dàng và với nỗ lực tối thiểu. Do đó, thiết kế của bạn phải "vô hình", người dùng không nên tập trung vào nó mà phải hoàn thành nhiệm vụ,

ví dụ: đặt bánh pizza trên ứng dụng Domino's Zero Click.

=>> Cần hiểu bối cảnh và luồng tác vụ của người dùng, để tinh chỉnh giao diện người dùng tốt nhất, trực quan nhất mang lại trải nghiệm liền mạch.

• Giao diện người dùng cũng phải thú vị (hoặc ít nhất là thỏa mãn và không gây thất vọng).

Khi thiết kế của bạn dự đoán được nhu cầu của người dùng, họ có thể tận hưởng những trải nghiệm được cá nhân hóa và phong phú hơn. Vui lòng cho họ, và họ sẽ tiếp tục quay trở lại.

Khi thích hợp, các yếu tố của trò chơi hóa có thể làm cho thiết kế của bạn thú vị hơn.

• Giao diện người dùng phải truyền đạt các giá trị thương hiệu và củng cố lòng tin của người dùng.

Thiết kế tốt là thiết kế cảm xúc . Người dùng liên kết cảm xúc tốt với các thương hiệu phù hợp với họ ở mọi cấp độ và giữ cho sự kỳ diệu của những trải nghiệm thú vị, liền mạch luôn tồn tại.

3. Cách tạo giao diện người dùng tuyệt vời

Để cung cấp GUI ấn tượng, hãy nhớ người dùng là con người, với những nhu cầu như sự thoải mái và giới hạn về năng lực tinh thần của họ nên làm theo các nguyên tắc sau:

• Làm cho các nút và các yếu tố phổ biến khác hoạt động có thể đoán trước được (bao gồm các phản hồi như chụm để thu phóng) để người dùng có thể sử dụng chúng ở mọi nơi một cách vô thức . Biểu mẫu nên theo chức năng.

• Duy trì khả năng khám phá cao . Ghi nhãn rõ ràng các biểu tượng và làm cho nó dễ được chú ý : ví dụ: bóng cho các nút.

• Giữ giao diện đơn giản (chỉ với các yếu tố giúp phục vụ mục đích của người dùng ) và tạo cảm giác "vô hình" .

• Tôn trọng mắt nhìn và sự chú ý của người dùng về bố cục . Tập trung vào hệ thống phân cấp và khả năng đọc:

• Sử dụng căn chỉnh thích hợp. Thông thường, chọn căn chỉnh cạnh (trên trung tâm).

• Thu hút sự chú ý đến các tính năng chính bằng cách sử dụng :

Màu sắc, độ sáng và độ tương phản. Tránh bao gồm màu sắc hoặc nút quá mức.

Văn bản thông qua các kích thước phông chữ, kiểu in đậm / trọng số, in nghiêng, viết hoa và khoảng cách giữa các chữ cái. Người dùng nên chọn nghĩa chỉ bằng cách quét.

• Giảm thiểu số lượng hành động để thực hiện các tác vụ nhưng tập trung vào một chức năng chính trên mỗi trang . Hướng dẫn người dùng bằng cách chỉ ra các hành động ưu tiên. Dễ dàng thực hiện các nhiệm vụ phức tạp bằng cách thông tin đến user 1 cách rõ rang.

• 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.

• Thông báo cho người dùng về các phản hồi / hành động của hệ thống bằng phản hồi như rung, thông báo.

• 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 các biểu mẫu). Cẩn thận với việc sử dụng các mẫu tối, bao gồm hộp kiểm chọn tham gia / chọn không tham gia được điền sẵn khó nhìn thấy và chuyển các mặt hàng vào giỏ hàng của người dùng.

• Duy trì tính nhất quán của thương hiệu .

• Luôn cung cấp các bước tiếp theo mà người dùng có thể suy luận một cách tự nhiên , bất kể bối cảnh của họ .

Các trường hợp thiết kế UI tệ

• Không tạo được ấn tượng cho khách hàng

• Gây khó khăn cho quá trình tham khảo

• Không có yếu tố dẫn dắt, thu hút người dùng

• Form điền thông tin rắc rối khiến người dùng cảm thấy khó đăng ký

• Text khó đọc khiến cho người dùng khó tham khảo và tìm thấy thông tin họ cần

• Sử dụng nhiều icon lạ, không đồng bộ gây bối rối cho quá trình sử dụng

Các thiết kế UI đem lại trải nghiệm tốt cho người dùng

Một thiết kế UI tốt cung cấp cho người dùng chính xác những gì họ cần theo cách nhanh chóng và đơn giản nhất sẽ giúp bạn tạo được ấn tượng tốt, nâng cao uy tín thương hiệu

• Giúp người dùng dễ dàng tiếp cận thông tin

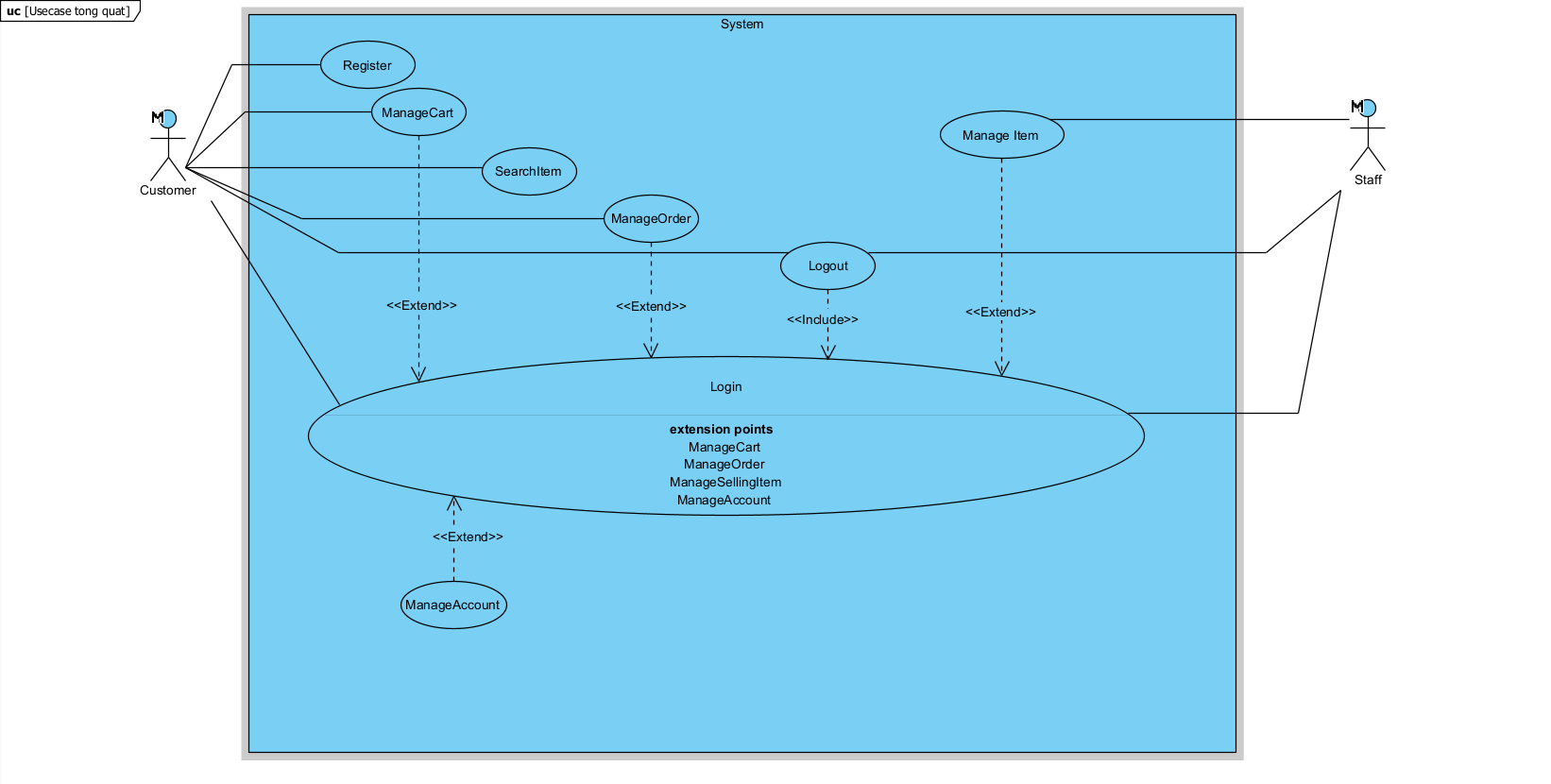
• Hướng dẫn trực quan, dễ sử dụng

• Thúc đẩy khách hàng thực hiện chuyển đổi (quyết định mua hàng, sử dụng dịch vụ, liên hệ, để lại thông tin…)

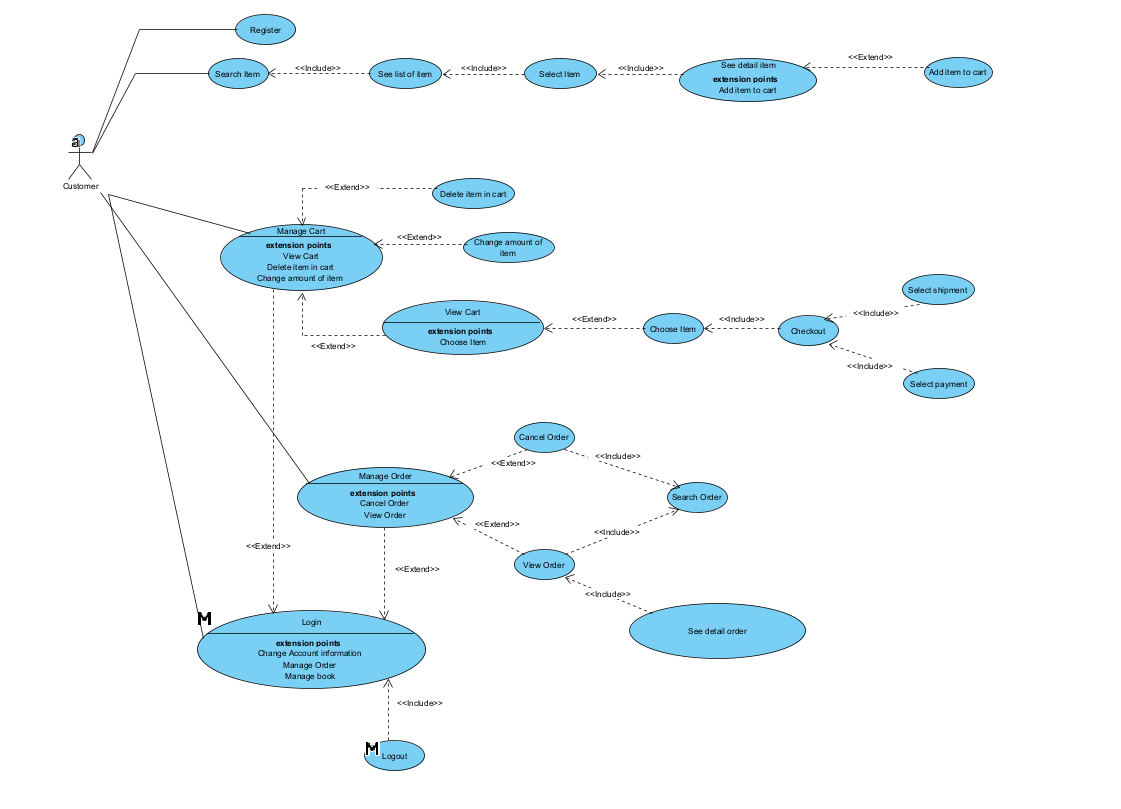
**Part 2 : Project** : Book store online

1. Draw a use case diagram with detail level

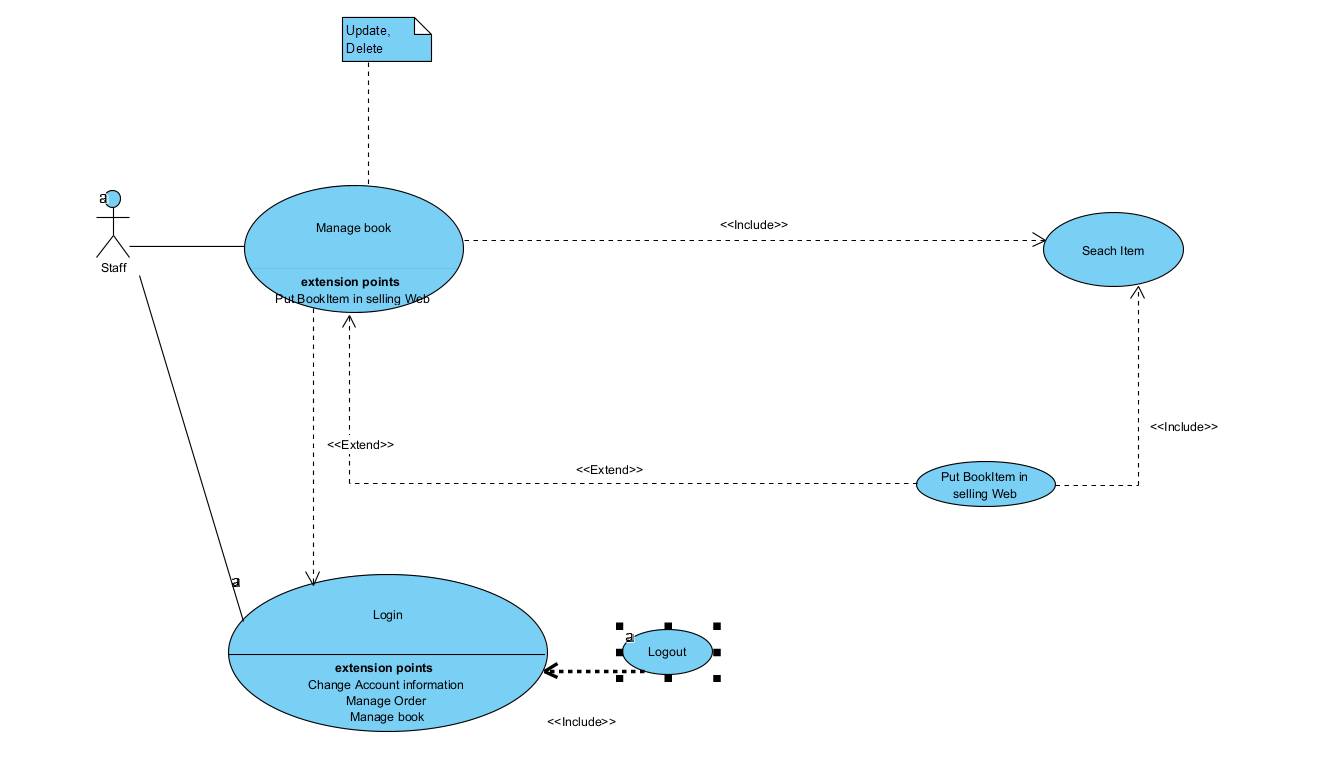
\*Usecase tổng quát

****

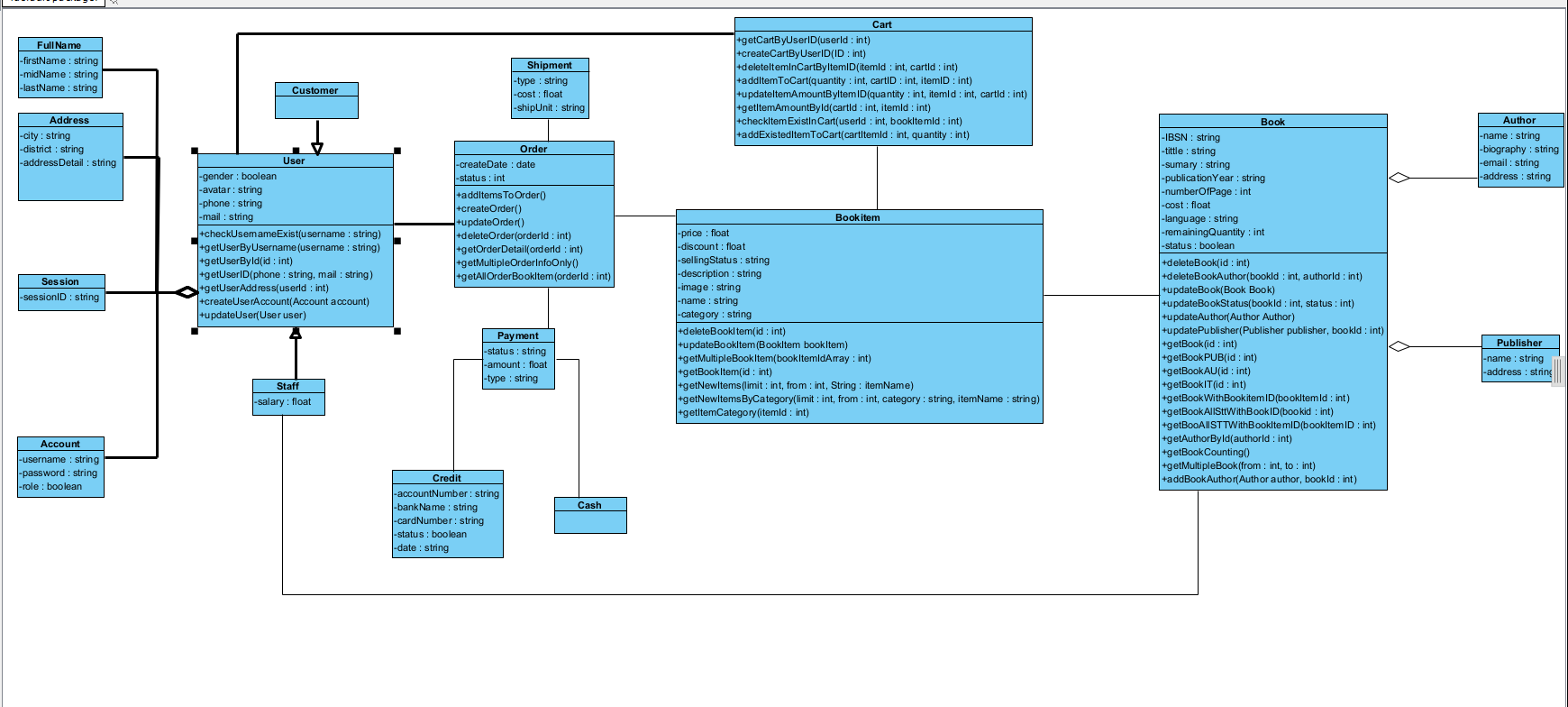
**\*U**secase Customer :



\*Usecase Staff :

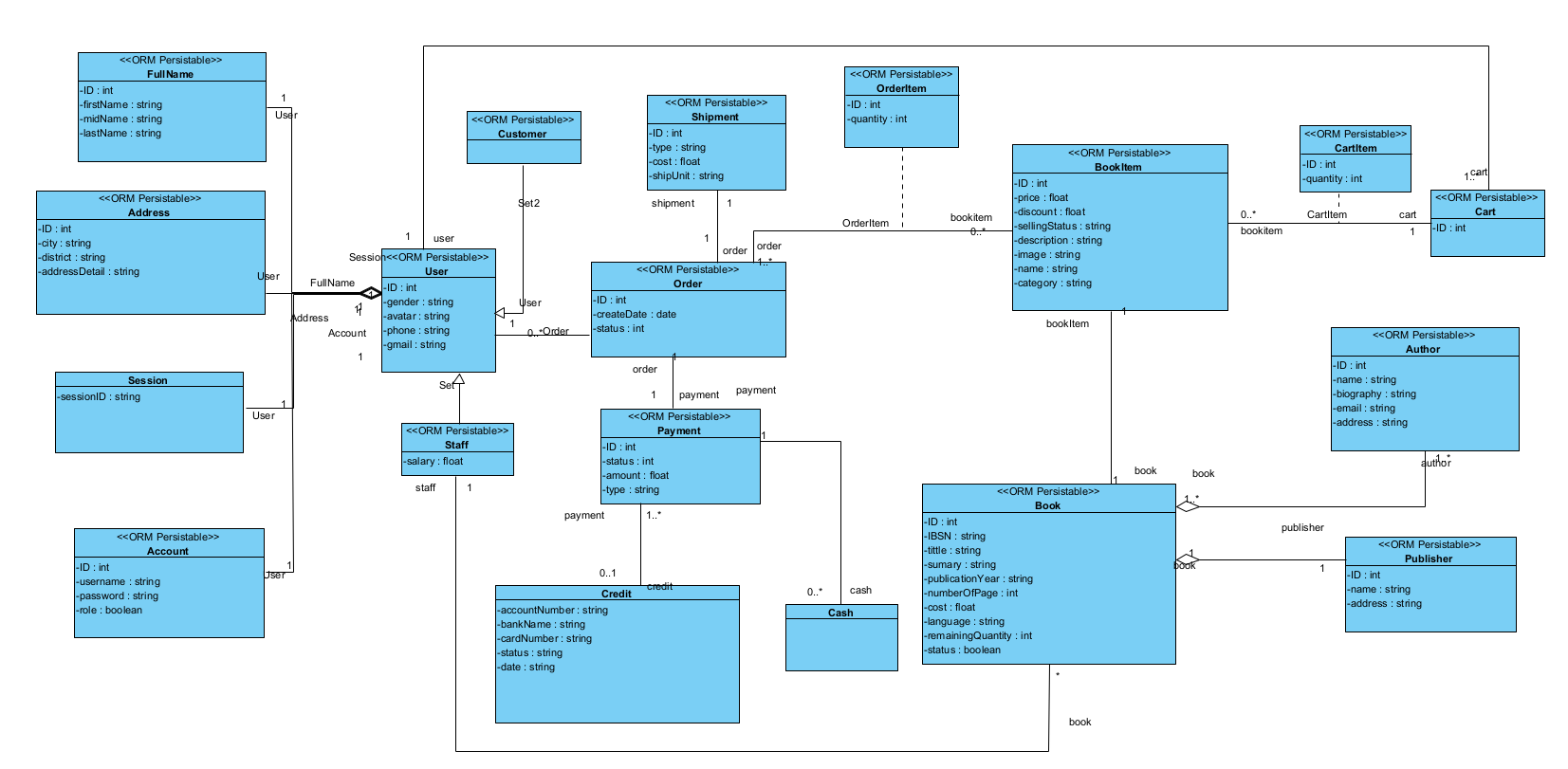


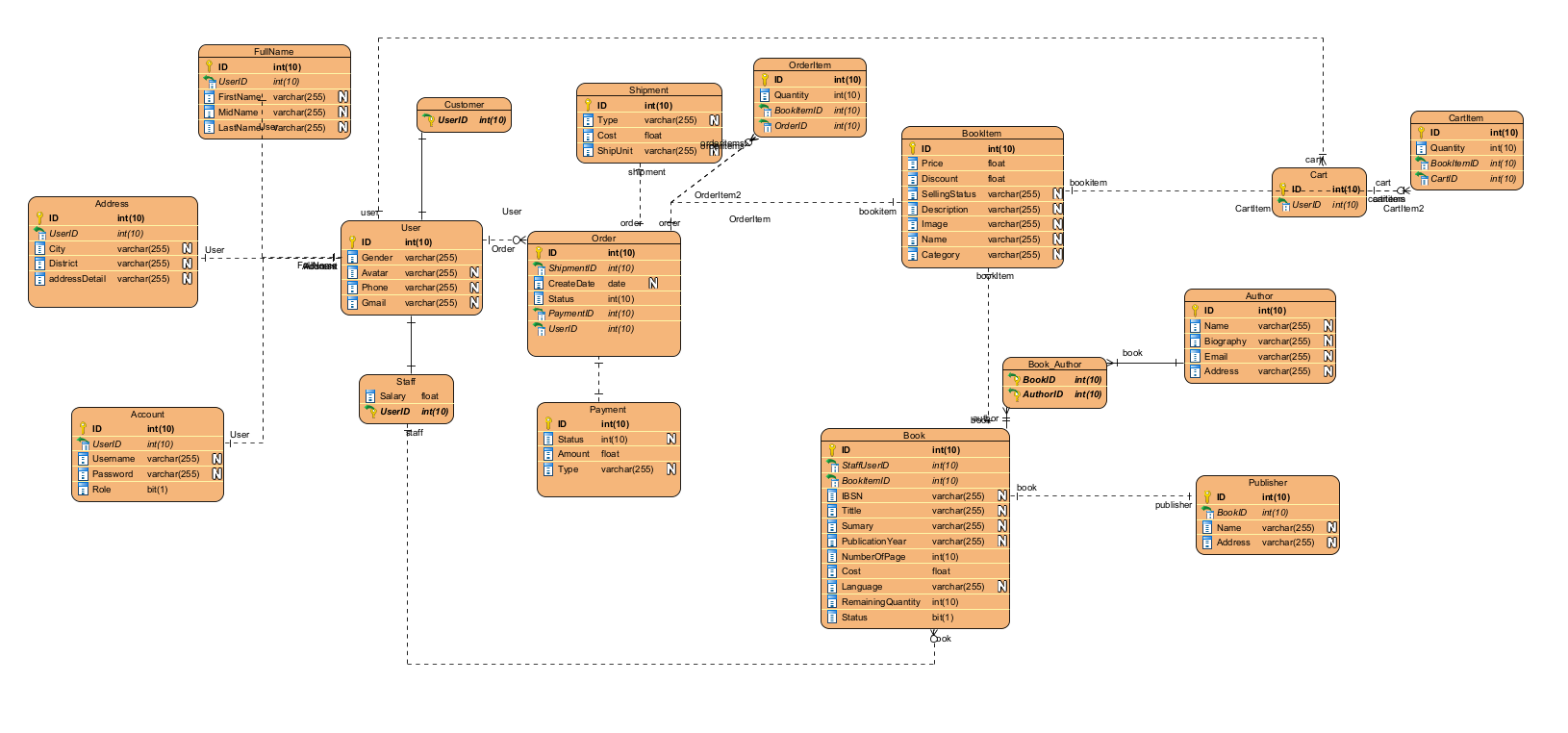
2. Construct a class diagram in analysis (NO 1…m, ORM) with some main methods



3. Construct a class diagram for data model (ADD 1…m, ORM), generate data model and Database mySQL

4. Construct a class diagram in design (ADD more attributes and methods).





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

6. Construct package diagrams with 3-layer architecture with MVC for your systems (follow the pattern given by [tdque@yahoo.com](mailto:tdque@yahoo.com)).

7. Implementation (utilize jsp servlet)

