**1. DATABASE INTRODUCTION**

- **Data (dữ liệu)** – mô tả sự vật, sự việc, hiện tượng,… dưới dạng hình ảnh, âm thanh, văn bản,…

=> Data rất quan trọng nên cần được lưu trữ.

- **Database** (**cơ sở dữ liệu**) – là nhóm dữ liệu có tổ chức (dạng bảng):

+ Row/ record/ tuple: hàng.

+ Column/ field: cột.

- **RDB** (**Relational Database**) – CSDL quan hệ, là mối quan hệ giữa các bảng với nhau (1-1, 1-n, n-n).

- **RDBMS** (**Relational Database Management System**) – Hệ quản trị CSDL quan hệ, là ứng dụng máy tính để quản lí CSDL.

Ví dụ: MySql, Sql server, Oracle,…

- **MySQL** – là hệ quản trị **CSDL**:

**+ MySql server** – dịch vụ để quản trị CSDL.

**+ MySql work bench** – là chương trình giúp lập trình viên có thể tương tác với CSDL.

- **SQL** (**Structured Query Language**) – ngôn ngữ truy vấn cấu trúc:

+ **DDL** (**Data Definition Language**) – định nghĩa cấu trúc dữ liệu (CREATE, ALTER, DROP,…).

+ **DML** (**Data Manipulation Language**) – thao tác, truy vấn dữ liệu

(SELECT, INSERT, UPDATE, DELETE,…).

+ **DCL** (**Data Control Language**) – điều khiển, phân quyền (GRANT, REVOKE,…).

+ **TCL** (**Transaction Control Language**) – tiến trình.

- **So sánh DELETE & TRUNCATE:**

+ **Giống:** đều là xóa **dữ liệu** của bảng (khác với DROP là xóa bay luôn cả bảng).

+ **Khác:**

|  |  |
| --- | --- |
| **DELETE** | **TRUNCATE** |
| - Thuộc nhóm câu lệnh **DML**.  - Dùng để xóa **table**, xóa **record** (có thể dùng chung với điều kiện).  - Đối với **id** có sử dụng **AUTO\_INCREMENT**, Khi chạy lệnh **DELETE** thì **SQL** sẽ **log** lại từng dòng đã xóa vào **transaction log**, vì thế nên khi bạn tạo 1 **record** mới, giá trị của **id** sẽ không bắt đầu từ **1** mà sẽ có giá trị **n+1** với **n** là giá trị của **record** cuối cùng được tạo.  - Dữ liệu bị xóa bởi **DELETE** có thể phục hồi (**restored)** được. | - Thuộc nhóm câu lệnh **DDL**.  - Chỉ có thể dùng cho **table** chứ không thể dùng cho từng **record**.  - Khi chạy lệnh **TRUNCATE** thì **SQL** sẽ xóa hết dữ liệu của bảng và **reset transaction log**, vì thế khi tạo 1 **record** mới, giá trị của **id** sẽ bắt đầu từ **1**, đây cũng chính là khác biệt lớn nhất của **DELETE** và **TRUNCATE**.  - Xóa toàn bộ các dòng của bảng, giải phóng bộ nhớ và không thể phục hồi lại.  - **TRUNCATE** không thể xóa đc bảng có **foreign key** trỏ đến bảng khác.  - **TRUNCATE** sẽ luôn nhanh hơn vì nó tiêu tốn ít bộ nhớ hơn. |

**2. DATABASE DESIGN**

- **ERD (Entity Relationship Diagram)** – là sơ đồ quan hệ giữa các thực thể.

=> Vì yêu cầu khách hàng (**SRS** **– Software Requirement Specification**) quá nhiều chữ, khô khan

🡪 khó tạo **DB** 🡪 cần thiết kế **ERD** để dễ dàng tạo **DB**.

- **Các bước thiết kế ERD:**

1. Xác định các thực thể (**Entity**) từ **SRS**.

2. Xác định các thuộc tính của **Entity**.

3. Xác định mối quan hệ giữa các **Entity** (**1-1, 1-n, n-n**).

4. Vẽ **ERD**.

- **Kiểu dữ liệu:**

+ **Số**: int, bigint, float, double,…

+ **Chuỗi**: char, varchar, text, longtext,…

+ **Ngày**: date, datetime,…

- **Constrain** (**ràng buộc**) – là quy tắc (**rule**) được quy định cho bảng.

=> Hạn chế những dữ liệu đưa vào bảng.

=> Giúp cho dữ liệu chính xác, tin cậy và toàn vẹn.

=> 1 số **constrain**:

+ **Primary key**: khóa chính (**not null** + **unique**).

+ **Not null**: không cho phép **null** (bỏ trống).

+ **Unique**: không được trùng lặp.

+ **Foreign key**: khóa ngoại (tham chiếu đến bảng).

+ **Check**: kiểm tra điều kiện.

+ **Default**: giá trị mặc định (khi không thêm giá trị cho cột thì cột sẽ mang giá trị mặc định).

+ **Index**: tăng tốc độ truy vấn.

- **Các bước chuyển đổi ERD -> table:**

+ **Bước 1**: Biến đổi thực thể (**Entity**) thành **table**.

=> Thuộc tính khóa của **Entity** là khóa chính của **table**.

=> Thuộc tính thường của **Entity** là thuộc tính của **table**.

+ **Bước 2:** Biến đổi thuộc tính đa trị.

=> Tách thuộc tính đa trị 🡪 đưa vào bảng mới 🡪 xác định mối quan hệ giữa bảng mới và bảng còn lại.

+ **Bước 3:** Biến đổi mối quan hệ **1-1**.

=> Đặt thuộc tính khóa ở phía tùy chọn sang phía bắt buộc.

+ **Bước 4:** Biến đổi mối quan hệ **1-n**.

=> Đặt khóa ngoại (**FK**) ở bên nhiều (**n**) và khóa chính (**PK**) ở bên **1**.

+ **Bước 5:** Biến đổi mối quan hệ **n-n**.

=> Tách thành bảng mới, có khóa chính là tập hợp 2 khóa của của 2 thực thể tham gia vào mối quan hệ (gọi là khóa phức hợp – **composite key**).

- **Ví dụ về các mối quan hệ:**

+ **1-1**: mỗi công dân có 1 CCCD và 1 CCCD chỉ thuộc về 1 công dân; mỗi học viên codeGym có 1 tài khoàn James và mỗi tài khoản james chỉ thuộc về 1 học viên,…

+ **1-n**: 1 bộ phận có nhiều nhân viên và 1 nhân viên chỉ thuộc 1 bộ phận,…

+ **n-n**: 1 khách hàng có thể order nhiều sản phẩm và 1 sản phẩm có thể đc order bởi nhiều khách hàng (thông qua hóa đơn),…

**3. DATABASE QUERY**

- **Join** – dùng để lấy thông tin trên 2 hay nhiều bảng, kết nối với nhau thông qua các trường chung (bản chất là gộp dữ liệu theo cột). Bao gồm:

+ **Inner join** – lấy phần chung của 2 bảng.

+ **Outer join:**

- **Left join** – lấy phần chung & phần bên trái.

- **Right join** – lấy phần chung & phần bên phải.

- **Full join** – lấy phần chung, phần bên trái & phần bên phải.

(**MySql** không hỗ trợ **Full join**, muốn sử dụng thì phải kết hợp giữa **Left join**, **Right join** và **Union**).

- **Where** – Mệnh đề điều kiện, dùng để truy xuất dữ liệu theo 1 điều kiện nào đó (một số toán tử theo sau **where**: <, >, < >, =, or, and, not, between, like, in,…).

- **Group by** – nhóm dữ liệu trên 1 cột, sắp xếp các hàng của truy vấn theo nhóm (thường dùng với các hàm count(), sum(), avg(), max(), min(),…).

- **Having** – Mệnh đề điều kiện, thường sử dụng sau **Group by** và thường sử dụng để làm điều kiện cho các hàm thống kê, sử dụng để lọc các bản ghi và chỉ lấy những bản ghi phù hợp với yêu cầu hoặc thực sự cần thiết tương tự như mệnh đề **WHERE**. Tuy nhiên:

+ **WHERE** là câu lệnh điều kiện trả kết quả đối chiếu với từng dòng.

+ **HAVING** là câu lệnh điều kiện trả kết quả đối chiếu cho nhóm được tạo bởi mệnh đề **GROUP BY**.

=> Vì vậy mà sau **GROUP BY** thì sẽ chỉ dùng được **HAVING** còn **WHERE** không dùng được sau **GROUP BY**.

- **Order by** – sử dụng sắp xếp theo 1 hoặc nhiều cột (**asc**/**desc**).

**4. HÀM THÔNG DỤNG**

- **Hàm gộp** – 1 số hàm gộp hay sử dụng với group by: sum, avg, count,…

- **Union & Unionall –** Dùng để kết nối kết quả 2 hay nhiều câu truy vấn (bản chất là gộp dữ liệu theo hàng, số hàng sẽ có sự thay đổi).

+ **Union**: gộp các record trùng nhau.

+ **Unionall**: không gộp các record trùng nhau.

+ Điều kiện: 2 câu truy vấn phải có cùng số lượng cột, tên cột và kiểu dữ liệu (ở MySQL thì không bắt buộc phải cùng kiểu dữ liệu).

- **Limit** – giới hạn số lượng record đc hiển thị (lấy ra).

+ **limit** A,B //A là vị trí cần lấy, B là số lượng cần lấy.

+ **limit** C //C là số lượng cần lấy, vị trí cần lấy mặc định là đầu tiên.

- **Truy vấn con** (**subquery**) – là 1 câu truy vấn lồng vào 1 câu truy vấn khác (hạn chế sử dụng truy vấn con vì nó làm giảm tốc độ tìm kiếm).

- **Một số hàm thông dụng:** length, concat, date, now, month,…

- **Phân biệt EXIST và IN?**

- **Giống:** đều đc sử dụng trong mệnh đề điều kiện và sử dụng với subquery.

- **Khác:**

|  |  |
| --- | --- |
| **EXIST** | **IN** |
| - Câu query cha có liên quan đến câu query con (có tương quan).  - Ứng với 1 dữ liệu, 1 record của câu query cha, sẽ truy xuất dữ liệu trong câu query con. Khi tìm thấy sẽ kết thúc tìm kiếm.  - Dùng với bảng có dữ liệu lớn. | - Câu query cha không liên quan đến câu query con (không tương quan).  - Chạy câu query con trước, và sẽ luôn duyệt hết dữ liệu trong câu query con.  - Dùng với bảng có dữ liệu nhỏ. |

- **Chuẩn hóa dữ liệu:** tránh dư thưa dữ liệu, đảm bảo tính toàn vẹn.

+ **Chuẩn hóa 1 NF:**

- Loại bỏ các thuộc tính đa trị (ví dụ: 1 người có thể có nhiều số đt, thì nên tách số đt thành 1 bảng mới).

- Loại bỏ các thuộc tính có giá trị được tính toán hoặc duy diễn từ các thuộc tính khác (ví dụ: 1 học sinh có 1 cột điểm, thì không nên có thêm 1 cột xếp loại khác suy ra từ cột điểm đó).



+ **Chuẩn hóa 2 NF:**

- Phải đạt chuẩn 1 NF.

- Loại bỏ các phụ thuộc bộ phận, tức là loại bỏ các thuộc tính không khóa và nó phụ thuộc vào 1 bộ phận nào đó của khóa primary key



+ **Chuẩn hóa 3 NF:**

- Phải đạt chuẩn 2 NF.

- Loại bỏ các phụ thuộc bắc cầu.



**5. INDEX, STORED PROCEDUCE, VIEW**

- **Index (chỉ mục)** – là 1 bảng tra cứu đặc biệt có thể sự dụng để tăng hiệu quả tìm kiếm dữ liệu.

+ **Cú pháp tạo:**

- Cách 1: **CREATE INDEX** i\_name **ON** table\_name (column);

- Cách 2: **ALTER TABLE** table\_name **ADD INDEX** i\_name (column);

+ **Cú pháp xóa:**

- Cách 1: **DROP INDEX** i\_name **ON** table\_name;

- Cách 2: **ALTER TABLE** table\_name **DROP INDEX** i\_name;

+ **[Lưu ý]**

- Không nên tạo **index** cho các bảng nhỏ, các bảng thường xuyên **update**, **insert** (các lệnh thao tác dữ liệu đòi hỏi nhiều thời gian xử lí hơn, vì chúng cần cập nhật sự thay đổi tới các **index**) hoặc có quá nhiều giá trị **null**.

- Chỉ có những cột hay tìm kiếm thì mới cần tạo **index** vì các các bảng có các **index** thì sẽ đòi hỏi nhiều dung lượng bộ nhớ hơn trong CSDL.

- **Index** sẽ duy trì thứ tự sắp xếp của dữ liệu, cho phép tìm kiếm dữ liệu trong bảng mà không cần phải duyệt qua toàn bộ bảng.

- **View** **(bảng ảo/giả)** – là các truy vấn (**select**) đc lưu lại dưới dạng bảng và sau đó có thể truy vấn dữ liệu như là bảng thật.

**+ Cú pháp tạo:** **CREATE VIEW** w\_name **AS** select\_statement;

**+ Cú pháp xóa: DROP VIEW** w\_name;

**+ Mục đích:**

- Tái sử dụng mã nguồn.

- Bảo mật thông tin cho bảng chính.

- **Stored Procedure** – tương tự hàm void trong **Java**, là tập hợp câu lệnh **query** để thực hiện 1 nhiệm vụ nào đó.

+ Cú pháp tạo:

**DELIMITER //**

**CREATE PROCEDURE** sp\_name (param);

**BEGIN**

-- code;

**END //**

**DELIMITER ;**

+ Trong đó: param – tham số (có 3 kiểu):

- **IN** – truyền tham trị.

- **OUT** – truyền tham chiếu.

- **INOUT** – kết hợp IN và OUT.

- **Funtion** – dùng thực hiện 1 nhiệm vụ nào đó (có trả về giá trị).

- **Trigger** – cũng dùng để thực hiện 1 nhiệm vụ nào đó.

- **So sánh Stored Procedure, Function, Trigger:**

|  |  |  |
| --- | --- | --- |
| **Stored Procedure** | **Function** | **Trigger** |
| - Tham số kiểu **IN/OUT/INOUT**.  - Không có dữ liệu trả về.  - Dùng câu lệnh **call** để gọi.  - Gọi đc **Function**. | - Tham số khiểu **IN**.  - Có dữ liệu trả về.  - Gọi như gọi hàm trong **Java**.  - Không gọi đc **SP**. | - Không có tham số.  - Không có dữ liệu trả về.  - Tự động chạy.  - Gọi đc **Function** & **SP**. |

**6. CSS**

- **CSS** (**Cascading Style Sheets**) – mô tả các hiển thị của các thẻ **HTML** trên trang web.

+ 3 cách nhúng:

- **Inline** – sử dụng thuộc tính style của thẻ **HTML** để viết **CSS**.

- **Internal** – sử dụng cặp thẻ **<style></style>** viết trong thẻ **<head>**.

- **External** – tạo 1 **file.css** và sử dụng thẻ **<link>** để nhúng vào **HTML** trong thẻ **<head>**.

+ Độ ưu tiên: **Inline** 🡪 **Internal**/**External** (tùy thuộc vào thẻ nào gần body hơn, theo cơ chế ghi đè).

- **Selector** (**bộ chọn**) – tìm/chọn các thẻ **HTML** để **style**.

+ Cú pháp: **selector** {property 1: value 1; property 2: value 2;…}

+ Các loại **selector**:

**- Element / tag** – tên thẻ (h1, p, div, a,…).

- **Id** – sử dụng **#id\_name** để định nghĩa.

- **Class** – sử dụng **.class\_name** để định nghĩa.

+ Lưu ý:

- 1 **id** chỉ dùng cho 1 thẻ **HTML** và ngược lại.

- 1 **class** dùng đc cho nhiều thẻ **HTML** và ngược lại.

- Độ ưu tiên: **id** 🡪 **class** 🡪 **element** / **tag**.

- **Box model:**

+ Là 1 cái hộp bao quanh thẻ **HTML** (sự chiếm không gian của các phần tử trong trang web).

+ Có 4 thành phần:

- **Content** – nội dung phần tử.

- **Padding** – khoảng cách từ **contend** đến **border**.

- **Border** – đường viền của thẻ **HTML**.

- **Margin** – khoảng cách từ **border** của **Box model** này đến **Box model** kia (lấy theo **margin** lớn hơn).

- **Display** – quy định cách hiển thị của thẻ **HTML**.

+ **None** – không hiển thị.

+ **Block** – bắt đầu bằng 1 dòng mới (chiếm toàn bộ chiều rộng màn hình).

+ **Inline** – không bắt đầu bằng dòng mới, chỉ chiếm chỗ hết phần chiều dài của nội dung (không phụ thuộc vào các thuộc tính **width** và **height**.

+ **Inline – Block:** kết hợp giữa **inline** & **block**.

- **Position** – là thuộc tính **CSS** để xác định cách thức hiển thị vị trí của các thẻ **HTML**. Có các giá trị:

+ **Static** – vị trí mặc định của thẻ **HTML** (không chịu ảnh hưởng bởi các thuộc tính top, left, right, bottom).

+ **Relative** – là vị trí tương đối, lấy chính nó làm gốc tọa độ (chịu ảnh hưởng bởi các thuộc tính top, left, right, bottom).

+ **Absolute** (tuyệt đối) – lấy định vị theo thẻ cha (với điều kiện thẻ cha không đc dùng static).

+ **Fixed** – là vị trí cố định với màn hình (không chịu ảnh hưởng bởi các **sroll bar** – thanh cuộn).

+ **Sticky** – khi cuộn sroll bar đến 1 vị trí nào đó (do LTV quy định) thì phần tử **HTML** sẽ dán cố định tại vị trí trên trang web.

- **Định dạng văn bản bằng CSS**

+ **color:** xxx; (màu sắc văn bản).

+ **text-align:** xxx; (căn chỉnh văn bản).

// **justify** - mỗi dòng sẽ được kéo dãn do đó chúng có chiều rộng bằng nhau, và các lề trái và phải thẳng hàng nhau.

+ **text-decoration:** none; (bỏ gạch chân liên kết).

+ **text-transform:** xxx;

// **uppercase** (in hoa tất cả), **lowercase** (in thường tất cả), **capitalize** (in hoa chữ cái đầu).

+ **text-indent:** xxx; (thụt lề văn bản).

+ **letter-spacing:** xxx; (khoảng cách giữa các kí tự).

+ **word-spacing:** xxx; (khoảng cách giữa các từ).

+ **line-height:** xxx; (chiều cao dòng).

+ **direction:** xxx; (hướng của văn bản).

**7. WEBSITE LAYOUT DESIGN**

- **Layout** (**bố cục**) – là cách phân bố nội dung trên website sao cho hợp lý. Cách thể hiện website phải chuyên nghiệp và cuốn hút người xem.

- **Grid View** – sử dụng để thiết kế web với nhiều hàng, mỗi hàng có thể có nhiều cột (thường chia làm 12 cột).

=> Dễ thiết kế, dễ đặt nội dung.

- **Responsive Web** – là cách thể hiện trang web trên nhiều loại thiết bị với nhiều màn hình khác nhau nhưng vẫn đảm bảo đc nội dung.

- **Cách thiết kế web dựa trên RWD Media Queries:**

+ Sử dụng **Grid View** **+** **@media queries**.

+ **@media queries** – dùng để **CSS** 1 khối thuộc tính dựa trên 1 điều kiện nào đó (ví dụ như thay đổi kích thước màn hình).

**8. BOOTSTRAP**

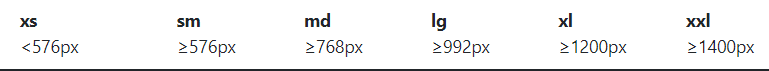
- **Bootstrap** – là 1 framework của CSS 🡪 hỗ trợ tạo trang web nhanh hơn, sinh động hơn và có hỗ trợ Reponsive web.

- **Cách nhúng:**

+ Dùng CDN: content delivery network (online).

+ Download về dự án 🡪 nhúng vào trong HTML.

- **Các loại màn hình do bootstrap hỗ trợ:**



- **Một số thành phần:** table, form, card, modal, nav bav, spinner, carousel,…

**9. JAVA WEB (JSP, SERVLET)**

- **HTTP** (**HyperText Transfer Protocol)** - giao thức truyền tải siêu văn bản.

- **HTTP / HTTPs:**

**+ Giống:** đều là 2 giao thức truyền tải siêu văn bản dùng cho **World Wide Web** (www) truyền tải dữ liệu dưới dạng văn bản, hình ảnh, âm thanh,… từ web **server** 🡪 trình duyệt web, **client** và ngược lại.

+ **Khác:** **HTTPs** đc tích hợp thêm chứng chỉ **SSL** (Secure Sockets Layer) 🡪 mã hóa dữ liệu và thông điệp giao tiếp 🡪 gia tăng tính bảo mật.

=> **HTTPs** là phiên bản **HTTP** nhưng an toàn hơn, bảo mật hơn.

**- So sánh Dynamic/Static web:**

**+ Giống:** đều đc xây dựng dựa trên các ngôn ngữ **front-end** (HTML, CSS, JS).

+ **Khác:**

|  |  |
| --- | --- |
| **Static web** | **Dynamic web** |
| - Là web tĩnh, nội dung hầu như không thay đổi.  - Không cần sử dụng các ngôn ngữ **back-end** để thiết kế.  - Thường là các web nhỏ, có ít nội dung.  - Không có sự tương tác với người dùng.  - Giao diện thân thiện với các cơ chế tìm kiếm hơn so với website động. Bởi vì các URL của html trong website tĩnh không chứa dấu chấm hỏi như trong website động.  - Chi phí đầu tư thấp, thời gian thiết kế ngắn nhưng bảo trì và nâng cấp khó khăn. | - Là web động có nội dung thay đổi tùy thuộc vào thời gian, người dùng, ngữ cảnh.  - Xây dựng dựa trên ngôn ngữ **front-end** và **back-end** (Java, Servlet, TSP, PHP, C#,…).  - Là các web lớn (tin tức, blog, thương mại điện tử,…).  - Có sự tương tác với người dùng.  - Chí phí xây dựng cao, thời dan thiết kế dài hơn nhưng dễ dàng nâng cấp và bảo trì. |

**- So sánh Get/Post:**

**+ Giống:**

- Đều là hai phương thức của giao thức HTTP.

- Đều gửi dữ liệu về **server** xử lí, sau khi người dùng nhập thông tin vào **form** và thực hiện **submit**.

- Trước khi gửi thông tin, nó sẽ được mã hóa bằng cách sử dụng một giản đồ gọi là **url** **encoding**. Giản đồ này là các cặp **name/value** được kết hợp với các kí hiệu = và các kí hiệu khác nhau được ngăn cách bởi dấu **&**.

**+ Khác:**

|  |  |
| --- | --- |
| **Get** | **Post** |
| - Truyền thông tin thông qua **URL**.  - Data hiển thị trên **URL**.  - Kém bảo mật.  - Data chỉ gửi đc dạng text, dung lượng gửi bị giới hạn (tối đa 2048).  - Có thể đc **cache** bởi trình duyệt.  - Dữ liệu gửi đi được lưu lại trong lịch sử web và có thể xem lại  - Thực thi nhanh hơn vì những dữ liệu gửi đi luôn được webbrowser cached lại.  (Ứng với cùng một yêu cầu đó **webbrowser** sẽ xem trong cached có kết quả tương ứng với yêu cầu đó không và trả về ngay không cần phải thực thi các yêu cầu đó ở phía server).  - Không gửi được nhị phân.  - Đối với dữ liệu ít thay đổi thường dùng phương thức **GET** để truy xuất và xử lý nhanh hơn. | - Truyền thông tin thông qua **HTTP header**.  - Data không hiển thị trên **URL**.  - Bảo mật hơn.  - Data gửi đc dạng text, hình ảnh, âm thanh,.. không giới hạn dung lượng gửi.  - Không **cache** bởi trình duyệt.  - Không được lưu lại trong lịch sử.  - Thực thi chậm hơn (vì phải mã hóa thông tin).  (Khi dùng phương thức **POST** thì server luôn thực thi và trả về kết quả cho **client**).  - Có thể gửi đc ASCII cũng như dữ liệu nhị phân.  - Đối với những dữ liệu luôn được thay đổi thì thường sử dụng phương thức **POST** |

**- Request** – yêu cầu từ client 🡪 server.

**- Response** – phản hồi (đáp trả) từ server 🡪 client.

- **So sánh JSP / Servlet:**

**+ Giống:**

- Đều nằm ở phía **server**.

- Đều chạy với **web container**.

- Đều là những công nghệ quan trọng của **Java EE** (Enterprise Edition).

- Đều là công nghệ đc sử dụng để tạo web động (xử lý dữ liệu “động”).

- Về bản chất, **JSP** chính là **Servlet** vì cả 2 đều viết đc **code HTML + Java**.

**+ Khác:**

|  |  |
| --- | --- |
| **JSP** | **Servlet** |
| - Đc viết dựa trên **HTML** (có thể nhúng code **Java**, mã **Java** nằm trong mã **HTML**).  - Mạnh về hiển thị (dùng để thiết kế giao diện, làm **view** trong **MVC**).  - Chỉ chấp nhận duy nhất yêu cầu của giao thức **HTTP**.  - Không cho phép ghi đè.  - Sửa lỗi nhanh. | - Đc viết dựa trên **Java** (có thể nhúng code **HTML**, mã **HTML** nằm trong mã **Java**).  - Mạnh về xử lý logic (dùng để xử lý nghiệp vụ, làm **controller** trong **MVC**).  - Chấp nhận tất cả yêu cầu của các giao thức.  - Cho phép ghi đè lên phương thức.  - Khó sửa lỗi, tốn thời gian thực thi (tải lại, tái biên dịch, khởi động máy chủ). |

- **Phân biệt forward/redirect:**

|  |  |
| --- | --- |
| **Forward** (chuyển tiếp) | **Redirect** (chuyển hướng) |
| - Chỉ chuyển hướng trong nội bộ trang web, ứng dụng, không ra ngoài đc.  - Khi chuyển tiếp có thể mang theo dữ liệu.  - Tốc độ xử lý nhanh hơn.  - Khi chuyển hướng vẫn lưu đường dẫn cũ. | - Có thể chuyển hướng ra ngoài trang web, ứng dụng.  - Khi chuyển hướng không mang theo dữ liệu, muốn mang theo dữ liệu phải kèm theo điều kiện.  - Tốc độ xử lý chậm hơn.  - Khi chuyển hướng sẽ hiển thị đường dẫn đích, đường dẫn mới. |

**10. JSTL**

- **JSP** (**Java server pages)** – là một ngôn ngữ kịch bản phía **server**, cho phép người dùng tạo ra các trang web động.

- **JSTL** (**JSP Standard Tag Library**) – là thư viện thẻ chuẩn đc xây dựng dựa trên ngôn ngữ **Java** 🡪 hỗ trợ việc hiển thị **JSP** dễ dàng hơn.

- **Cách sử dụng thẻ JSTL:** import thư viện JSTL vào trong dự án (mvnrepository.com) 🡪 copy đường dẫn vào trong JSP cần sử dụng.

- **Các nhóm thẻ JSTL:**

+ Các thẻ cơ bản (**Core Tags**).

+ Các thẻ định dạng (**Formatting Tags**) .

+ Các thẻ SQL (**SQL Tags**).

+ Các thẻ XML (**XML Tags**).

+ Các thẻ hàm JSTL (**JSTL Functions Tags**).

- **1 số thẻ JSTL-core hay dùng:**

+ **<c:redirect url=**”đường dẫn chuyển hướng”**> </>**

+ **<c:forEach varStatus=**”trạng thái vòng lặp” **var=**”biến đại diện” **items=**”danh sách”**> </c:forEach>**

+ **<c:if test=**”điều kiện”**> //code </c:if>**

+ **<c:choose>**

**<c:when=**”điều kiện”**> //**code **</c:when>**

**<c:otherWise>** //code **</c: otherWise>**

**</ c:choose>**

(**choose** gần giống với **switch**, **when** gần giống với **case**, **otherWise** gần giống với **default**).

**11. MVC**

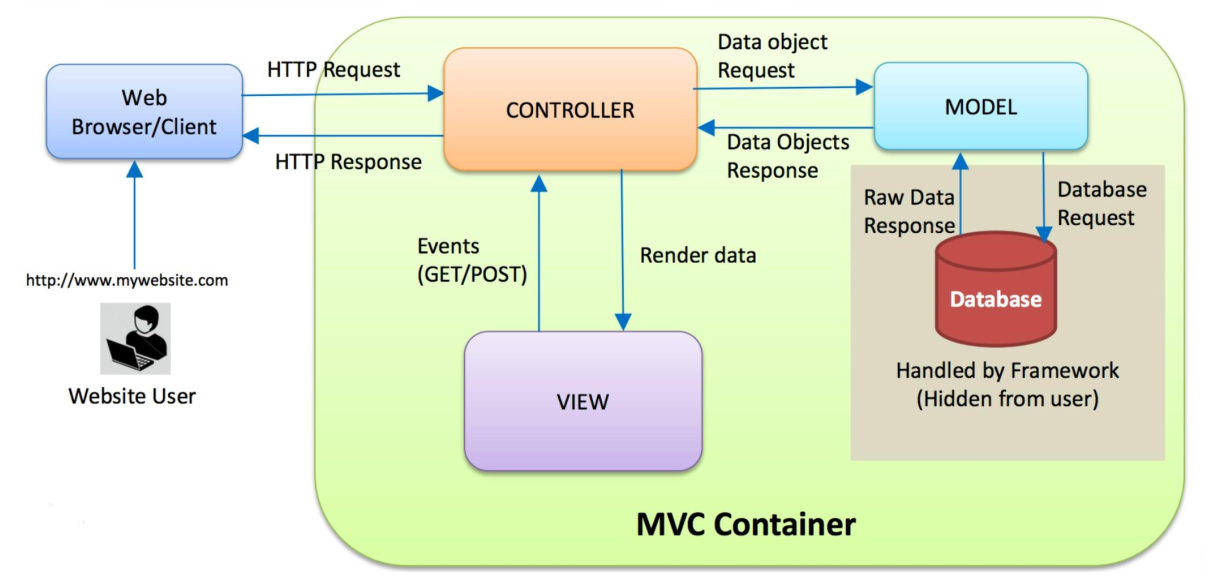
- **MVC** – là mô hình kiến trúc phần mềm, đc chia làm 3 tầng:

+ **View** – là nơi hiển thị dữ liệu + tương tác với người dùng.

+ **Model** – là nơi biểu diễn dữ liệu (**model**), xử lí logic nghiệp vụ (**service**), tương tác với DB (**repository**).

+ **Controller** – đóng vai trò điều hướng, là nơi nhận **request**, xử lý **request**, và là cầu nối giữa **Model** với **View**.

- **Dòng di chuyển của MVC:**



**Khi có 1 request gửi tới**

🡪 **controller** sẽ nhận **request**

🡪 gửi dữ liệu sang **model** xử lý

🡪 **model** sẽ thực hiện kiểm tra (**validate**), tính toán dữ liệu

🡪 nếu cần có thể truy cập vào **DB** (ví dụ như update, create, delete,…), hoặc không (ví dụ như chuyển trang, click vào thẻ <a>,…)

🡪 **model** sẽ trả dữ liệu về cho **controller**

🡪 **controller** sẽ **render** dữ liệu lên **view**

🡪 **controller** sẽ trả **view** về cho người dùng

🡪 hiển thị thông tin cho người dùng.

- **Sự khác nhau giữa MVC1 và MVC2:**

|  |  |
| --- | --- |
| **MVC1** | **MVC2** |
| - Controller và view đều là JSP.  - Dễ dàng và nhanh chóng phát triển web.  - Khó kế thừa, khó bảo trì.  - Chỉ thích hợp với các dự án nhỏ. | - Controller là servlet, model là class trong Javs, view là JSP.  - Cần nhiều thơi gian hơn để phát triển web.  - Tin cậy, tương thích cao, tái sử dụng, dễ bảo trì, hiệu quả và logic hơn,  - Thích hợp với các dự án lớn. |

**12. JDBC**

- **JDBC (Java Database connectivity)** – là 1 chuẩn **API** cho phép kết nối các ứng dụng **Java** & **CSDL** (MySQL, Oracle, SQL server,…).

- **API** (**Application Program Interface**) – là phương thức trung gian kết nối các ứng dụng khác nhau.

- **Các thành phần của JDBC:**

+ **Driver** – liên kết các kết nối với DB, điều khiển các liên kết.

+ **DriverManager** – quan lý danh sách các Driver.

+ **Connection** - biểu thị kết nối tới DB.

=> Dùng để tạo ra Statement, PreparedStatement và CallableStatement.

+ **Statement**, **PreparedStatement**, **CallableStatement** – chứa lệnh SQL gửi tới DB để thực thi.

+ **ResultSet** – biểu diễn một tập kết quả trong DB tạo ra bởi việc sử dụng một câu lệnh SQL là SELECT.

+ **SQLException** – một lớp xử lý lỗi ngoại lệ chứa các lỗi truy cập DB.

- **Các thông số cần thiết để kết nối:**

+ **Driver:** cần import vào project.

+ **Địa chỉ:** localhost:3306.

+ **Username:** root.

+ **Password:** pass khi cài đặt MySQL.

- **So sánh Statement/ PreparedStatement/ CallableStatement:**

+ **Giống:** đều là các **Interface** thực thi câu lệnh truy vấn **SQL** trong **JDBC**.

+ **Khác:**

|  |  |  |
| --- | --- | --- |
| **Statement** | **PreparedStatement** | **CallableStatement** |
| - Sử dụng các câu lệnh truy vấn tĩnh, không thể truyền tham số trong lúc runtime.  - Sử dụng khi câu truy vấn chỉ dùng 1 lần (**DDL**).  - Hiệu năng thấp hơn (ngay cả khi thực hiện cùng 1 truy vấn, chúng luôn được biên dịch trước và sau đó được thực thi, làm cho chúng chậm hơn).  - Muốn đưa tham số vào thì phải ở dạng cộng chuỗi, nên độ bảo mật không cao. | - Sử dụng để thực hiện các câu lệnh truy vấn động, có tham số.  - Sử dụng khi câu truy vấn dùng nhiều lần (**DML**).  - Kế thừa **Statement**, hiệu năng cao hơn (truy vấn được biên dịch lần đầu tiên nhưng sau đó nó được lưu trữ tại máy chủ DB, làm cho lần chạy tiếp theo nhanh hơn).  - Bảo mật. | - Sử dụng để gọi **stored procedures**.  - Kế thừa **PreparedStatement**, nên hiệu năng cao hơn.  - Bảo mật hơn. |

**13. TRANSACTION**

- **Transaction** – là 1 tiến trình có điểm đầu và điểm cuối, đc chia thành các hoạt động, dựa trên các quy tắc:

+ Nếu tất cả các hoạt động thành công thì tiến trình sẽ thành công 🡪 commit dữ liệu vào DB.

+ Nếu 1 trong các hoạt động không thành công 🡪 tiến trình sẽ thất bại 🡪 dữ liệu sẽ rollback (phục hồi) lại trạng thái ban đầu trước khi thực hiện.

+ Các bước tạo 1 transaction:

- **setAutoCommit(false)** – thiết lập “false” để tắt chế độ auto-commit.

- **commit()** – lưu dữ liệu vào DB.

- **rollback()** – quay lại trạng thái ban đầu, khôi phục lại toàn bộ các thay đổi trong **transaction** đã thực hiện xuống DB.

(có thể sử dụng setSavepoint() để lưu lại thời điểm rollback).

- **Tính chất: ACID**

+ **Atomicity** - nghĩa là tất cả thành công hoặc không (all or nothing).

+ **Consistency** – nếu transaction gây ra những vi phạm về ràng buộc dữ liệu, hệ thống sẽ không cho phép thực hiện tiếp và hủy tiến trình (tính nhất quán).

+ **Isolation** – đảm bảo 2 transaction cùng cập nhật 1 dữ liệu, hệ quản trị CSDL sẽ đảm bảo chúng thực hiện 1 cách tuần tự (tính đồng bộ).

+ **Durability** – khi transaction thực hiện thành công (đã commit), thì những cập nhật trở nên cố định và dữ liệu sẽ luôn là như vậy (tính bền vững).

**14. PACKAGING DEPENDENCES MANAGEMENT**

- **Phân biệt Ant/ Maven/ Gradle:**

|  |  |  |
| --- | --- | --- |
| **Ant** | **Maven** | **Gradle** |
| - Ra đời năm 2000.  - Import thư viện 1 cách thủ công, không có file quản lý thư viện. | - Ra đời năm 2004.  - Khắc phục nhược điểm của Ant  🡪 có file pom.xml để quản lý thư viện  🡪 file này tương đối dài dòng. | - Ra đời năm 2012.  - Khắc phục nhược điểm của Maven  🡪 sử dụng file build.gradlew (viết bằng ngôn ngữ Groovy)  🡪 ngắn gọn, hỗ trợ nhiều ngôn ngữ (Java, PHP, C++,…). |

<link rel="stylesheet" href="<https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.1.2/css/all.min.css>">

(nhúng vào thẻ <head>, sau đó vào web <https://fontawesome.com/> để copy icon).

compileOnly group: 'javax.servlet', name: 'javax.servlet-api', version: '4.0.1'

implementation group: 'javax.servlet', name: 'jstl', version: '1.2'

implementation group: 'mysql', name: 'mysql-connector-java', version: '8.0.23'

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

**Trong file build.gradle thêm 2 thư viện sau vào:**

compileJava.options.encoding = 'UTF-8'

tasks.withType(JavaCompile) {options.encoding = 'UTF-8'}

**Trong doGet và doPost thì thêm câu lệnh này đầu tiên:**

request.setCharacterEncoding("UTF-8");

<https://o7planning.org/10429/java-jsp-standard-tag-library>

<https://hoclaptrinh.vn/tutorial/hoc-jsp/cu-phap-trong-jsp>

private static final String NAME = "^([\\p{Lu}][\\p{Ll}]{1,8})(\\s([\\p{Lu}]|[\\p{Lu}][\\p{Ll}]{1,10})){0,5}$";

private static final String EMAIL = "^[a-z]+([\\\_\\.]?[a-z\\d]+)\*@[a-z]{3,7}\\.[a-z]{2,3}$";

private static final String PHONE = "^(090|091|\\(\\+84\\)90|\\(\\+84\\)91)\\d{7}$";

private static final String ID\_CARD = "^(\\d{9}|\\d{12})$";

**The Last Shinobi…**