Mục lục:

1. Datatype của SQL
2. Phân biệt char, varchar
3. Sự chuyển đổi giữa các kiểu dữ liệu
4. Phân biệt truncate vs drop vs delete
5. Phân biệt identity vs sequence
6. Constraint trong SQL?
7. Có bao nhiêu cách tạo constraint trong SQL?
8. Phân biệt Unique vs PK.
9. Trình bày về index
10. Trình bày thứ tự chạy sql
11. Phân biệt where vs having
12. Trình bày về union, merge, pivot, intersect
13. Trình bày về relationship, các chuẩn hóa 1nf, 2nf, 3nf
14. Trình bày về group by
15. Các function trong SQL
16. Trình bày về join, 2 table ko có relationship có join được ko?
17. Trình bày về view
18. Trình bày về subquery
19. Trình bày về ranking, row_number
20. trình bày về CTE
21. Trình bày về các câu lệnh điều kiển trong SQL
22. Trình bày về UDF trong sql
23. Trình bày về SP trong SQL
24. Trình bày về exception trong sql
25. Trình bày về transaction trong sql
26. Trình bày về Trigger trong sql
27. Trình bày về ACID

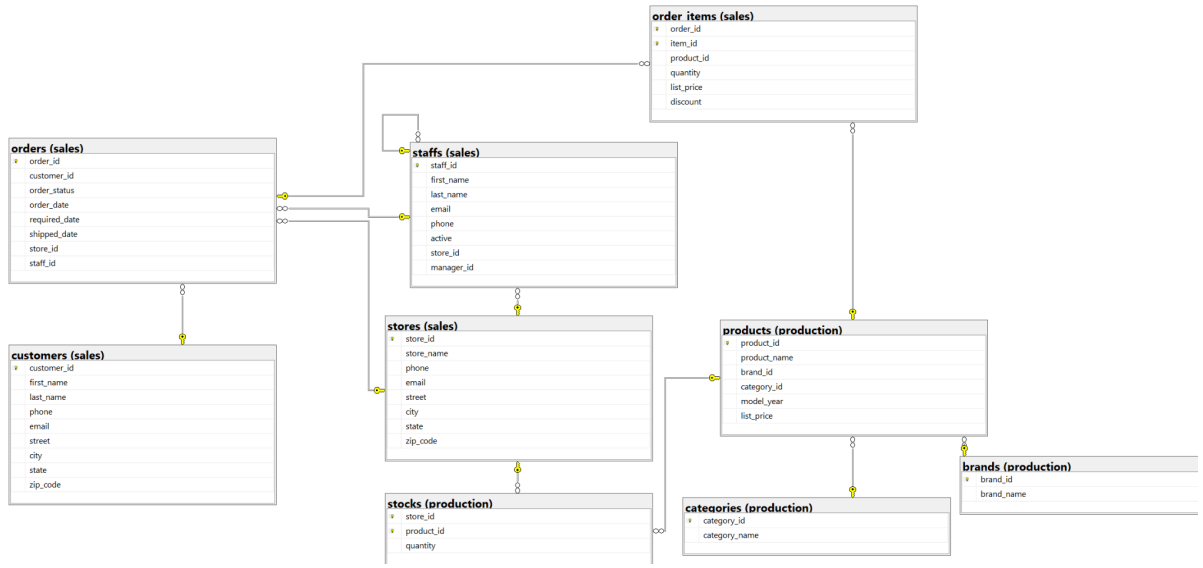
“Vì hơi dài nên hãy ctrl F tìm theo mục lục nhé”

Hoàng Hoàng - Cũ

Link tải data cho SQL Server

https://www.sqlservertutorial.net/getting-started/sql-server-sample-database/

DATABASE DIAGRAM chính để làm tài liệu:



1. Datatype của SQL

Numeric Types (Kiểu số)

- TINYINT: Số nguyên từ 0 đến 255.
- SMALLINT: Số nguyên nhỏ (-32,768 đến 32,767).
- INT hoặc INTEGER: Số nguyên lớn (-2,147,483,648 đến 2,147,483,647).
- BIGINT: Số nguyên rất lớn.
- DECIMAL(p, s) hoặc NUMERIC(p, s): Số thập phân với p tổng số chữ số và s chữ số phần thập phân.
- FLOAT(n): Kiểu số thực dùng cho số lớn hoặc nhỏ hơn, có độ chính xác đôi khi không hoàn hảo.

Ví dụ trong bảng `products`:

```
CREATE TABLE products (
                    product_id INT PRIMARY KEY,
                    product_name VARCHAR(100),
                    brand_id INT,
                    category_id INT,
                    model_year INT,
                    list_price DECIMAL(10, 2)
                    );
```

2. Phân biệt char, varchar

CHAR:

- Lưu trữ chuỗi ký tự có độ dài cố định.
- Nếu chuỗi ngắn hơn độ dài khai báo, phần còn lại sẽ được điền bằng khoảng trắng.

VARCHAR:

- Lưu trữ chuỗi ký tự có độ dài thay đổi.
- Chỉ sử dụng không gian lưu trữ cần thiết cho dữ liệu.

Ví dụ:

INSERT INTO stores (store_name) VALUES ('BikeStore'); có thể dùng CHAR(20)

INSERT INTO customers(email) VALUES ('abcdefg2342343@gmail.com'); có thể dùng VARCHAR(20) vì customer có thể thay đổi thông tin

3. Sự chuyển đổi giữa các kiểu dữ liệu

- **CAST:** Chuyển đổi kiểu dữ liệu.
- **CONVERT:** Tương tự CAST nhưng có thêm tùy chọn định dạng, chỉ hỗ trợ trong SQL Server

Ví dụ:

SELECT CAST(123 AS VARCHAR); Chuyển số 123 thành chuỗi '123'

SELECT CONVERT(VARCHAR, GETDATE(), 103); Chuyển ngày hiện tại sang định dạng 'dd/MM/yyyy'

4. Phân biệt truncate vs drop vs delete

- **TRUNCATE**: Xóa tất cả dữ liệu trong bảng nhưng giữ lại cấu trúc bảng.
- **DROP**: Xóa cả bảng lẫn dữ liệu.
- **DELETE**: Xóa dữ liệu theo điều kiện, giữ lại cấu trúc bảng.

Ví dụ:

```
TRUNCATE TABLE orders;
DROP TABLE stocks;
DELETE FROM customers WHERE city = 'Tokyo';
```

5. Phân biệt identity vs sequence

- **IDENTITY**: Tự động tăng giá trị trong một cột dành cho một bảng.
- **SEQUENCE**: Tạo chuỗi số có thể được sử dụng trong nhiều bảng.

Ví dụ:

```
CREATE TABLE orders (
```

```
    order_id INT IDENTITY(1, 1) PRIMARY KEY,
    customer_id INT,
    order_date DATE
);
CREATE SEQUENCE seq_transaction_id START WITH 1 INCREMENT BY 1;
```
Có thể dùng seq_transaction_id để quản lý cả order và invoices

6. Constraint trong SQL?

Constraint là các ràng buộc trên cột nhằm bảo vệ tính toàn vẹn dữ liệu. Một số constraint phổ biến:

- **NOT NULL**: Không cho phép giá trị null.
- **UNIQUE**: Đảm bảo giá trị duy nhất.
- **PRIMARY KEY**: Định danh duy nhất cho một hàng.
- **FOREIGN KEY**: Tạo liên kết giữa các bảng.
- **CHECK**: Kiểm tra giá trị theo điều kiện.
- **DEFAULT**: Giá trị mặc định khi không cung cấp dữ liệu.

Ví dụ:

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES
customers(customer_id);
```

7. Có bao nhiêu cách tạo constraint trong SQL?

- Khi tạo bảng:

Ví dụ:

```
CREATE TABLE products ( product_id INT PRIMARY KEY, product_name
VARCHAR(50) NOT NULL );
```

- Sau khi tạo bảng:

Ví dụ:

```
ALTER TABLE products ADD CONSTRAINT unique_product_name UNIQUE
(product_name);
```

8. Phân biệt Unique vs PK.

- **UNIQUE**: Cho phép giá trị duy nhất nhưng có thể có nhiều cột UNIQUE trong một bảng, nhưng cho phép NULL (trừ khi kết hợp với NOT NULL).
- **PRIMARY KEY**: Đảm bảo giá trị duy nhất và không cho phép NULL. Mỗi bảng chỉ có **một** primary key.

Ví dụ:

```
ALTER TABLE stores ADD CONSTRAINT pk_store_id PRIMARY KEY (store_id);
ALTER TABLE customers ADD CONSTRAINT unique_email UNIQUE (email);
```

9. Trình bày về index

- **Index:** là một đối tượng cơ sở dữ liệu được tạo trên một hoặc nhiều cột của bảng.
- Nó lưu trữ các giá trị của cột được chỉ định cùng với con trỏ đến các hàng tương ứng trong bảng.
- Khi truy vấn dữ liệu, hệ thống sẽ sử dụng index để tìm kiếm nhanh chóng thay vì quét toàn bộ bảng

Các loại index:

- **Clustered Index**: Sắp xếp và lưu trữ dữ liệu vật lý của bảng. Một bảng chỉ có thể có một clustered index.
- **Non-clustered Index**: Chỉ lưu trữ con trỏ trỏ đến dữ liệu thực tế. Một bảng có thể có nhiều non-clustered index.
- **Single-Column Index:** Index được tạo trên một cột duy nhất.
- **Composite Index (Multi-Column Index):** Index được tạo trên nhiều cột.
- **Unique Index:** Đảm bảo các giá trị trong cột (hoặc nhóm cột) là duy nhất.

Ví dụ:

```
CREATE INDEX idx_customer_name ON customers (last_name);
CREATE INDEX idx_name_department ON employees(first_name, department_id);
```

10.Trình bày thứ tự chạy sql

Thứ tự thực thi các mệnh đề trong truy vấn SQL:

1. **FROM** (bao gồm JOIN).
2. **WHERE** (lọc dữ liệu).
3. **GROUP BY** (nhóm dữ liệu).
4. **HAVING** (lọc nhóm).
5. **SELECT** (chọn cột).

6. **ORDER BY** (sắp xếp).

* Thứ tự viết câu lệnh SQL không ảnh hưởng đến thứ tự thực thi. Ví dụ, SELECT luôn được viết đầu tiên trong câu lệnh, nhưng nó được thực thi gần cuối.

Ví dụ: danh sách khách hàng có ít nhất 3 đơn đặt hàng từ 2024-01-01

```
SELECT
	c.customer_id,
	c.first_name,
	c.last_name,
	COUNT(o.order_id) AS total_orders
FROM customers AS c
JOIN orders AS o
	ON c.customer_id = o.customer_id
WHERE  o.order_date >= '2024-01-01'
GROUP BY
	c.customer_id,
	c.first_name,
	c.last_name
HAVING COUNT(o.order_id) > 2
ORDER BY total_orders DESC;
```

11. Phân biệt where vs having

- **WHERE**: Lọc dữ liệu trước khi nhóm (**GROUP BY**).
- **HAVING**: Lọc dữ liệu sau khi nhóm.

Ví dụ:

```
SELECT * FROM customers WHERE city = 'Tokyo';

SELECT store_id,
	COUNT(product_id)
FROM stocks
GROUP BY store_id
HAVING COUNT(product_id) > 10;
```

12. Trình bày về union, merge, pivot, intersect

UNION:

- Kết hợp kết quả của hai hoặc nhiều truy vấn thành một tập kết quả duy nhất. Các truy vấn phải có cùng số cột.

- Các cột tương ứng phải có kiểu dữ liệu tương thích.

2 loại

- **`UNION`:** Loại bỏ các hàng trùng lặp.
- **`UNION ALL`:** Giữ lại các hàng trùng lặp.

Ví dụ:

```
SELECT product_name FROM products WHERE category_id = 1
UNION
SELECT product_name FROM products WHERE category_id = 2;
```

MERGE

- Thực hiện các thao tác INSERT, UPDATE, hoặc DELETE trên một bảng dựa trên kết quả của một truy vấn khác.

Ví dụ:

```
MERGE INTO stocks AS target
USING new_stocks AS source
ON target.product_id = source.product_id
WHEN MATCHED THEN
UPDATE SET target.quantity = source.quantity;
```

PIVOT

- Chuyển đổi dữ liệu từ dạng hàng sang dạng cột (xoay bảng).

Ví dụ:

```
SELECT *
FROM (
        SELECT store_id, model_year, list_price
        FROM products
        ) AS SourceTable PIVOT (
                                        SUM(list_price)
                                        FOR model_year
                                        IN ([2022], 2023])
                                          ) AS PivotTable;
```

INTERSECT

- Trả về các hàng chung (giao nhau) giữa hai truy vấn.

Ví dụ:

```
SELECT product_name FROM products WHERE brand_id = 1
INTERSECT
SELECT product_name FROM products WHERE category_id = 3;
```

13.Trình bày về relationship, các chuẩn hóa 1nf, 2nf, 3nf

- **Relationship**: Quan hệ giữa các bảng trong cơ sở dữ liệu (1-1, 1-n, n-n).
- **1NF (First Normal Form)**: Mỗi cột chỉ chứa một giá trị, không có tập hợp hay mảng lồng nhau, mỗi hàng là duy nhất (có khóa chính)

Ví dụ:

Trước khi chuẩn hóa:

student_id	courses
1	Math, Science
2	History

Sau khi chuẩn hóa (1NF):

student_id	course
1	Math
1	Science
2	History

- **2NF (Second Normal Form)**: Thỏa mãn 1NF và không có phụ thuộc bộ phận, mỗi cột phải phụ thuộc vào toàn bộ khóa chính trong bảng (**kể cả toàn bộ composite key**).

Ví dụ:

Trước khi chuẩn hóa:

order_id	product_id	product_name	quantity
1	101	Laptop	2
1	102	Mouse	5
2	101	Laptop	1

Sau khi chuẩn hóa (2NF):

- Bảng `orders`:

order_id	product_id	quantity
1	101	2
1	102	5
2	101	1

- Bảng `products`:

product_id	product_name
101	Laptop
102	Mouse

Khóa chính: (order_id, product_id) **(composite key).**

★ **Vấn đề:**

➔ Cột product_name chỉ phụ thuộc vào product_id, không phụ thuộc vào order_id. Đây là phụ thuộc bộ phận, vi phạm **2NF.**

- **3NF (Third Normal Form)**: Thỏa mãn 2NF và không có phụ thuộc hàm bắc cầu(transitive dependency) nghĩa là các cột không phải khóa chính không được phụ thuộc vào cột khác không phải khóa chính.

Ví dụ:

Trước khi chuẩn hóa:

student_id	student_name	department	department_head
1	John	CS	Dr. Smith
2	Jane	Math	Dr. Brown

Sau khi chuẩn hóa (3NF):

- Bảng `students`:

student_id	student_name	department
1	John	CS
2	Jane	Math

- Bảng `departments`:

department	department_head
CS	Dr. Smith
Math	Dr. Brown

Khóa chính: student_id.

★ **Vấn đề:**

➔ Cột department_head phụ thuộc vào department, không phụ thuộc trực tiếp vào student_id. Đây là phụ thuộc bắc cầu, vi phạm **3NF.**

14. Trình bày về group by

- Dùng để nhóm dữ liệu theo một hoặc nhiều cột, thường kết hợp với hàm tổng hợp.

Ví dụ:

```
SELECT store_id,
        COUNT(product_id) AS total_products
FROM stocks
GROUP BY store_id;
```

15. Các function trong SQL

Hàm tổng hợp:

- **COUNT():** Đếm số lượng.
- **SUM():** Tính tổng.
- **AVG():** Tính trung bình.
- **MIN(), MAX():** Tìm giá trị nhỏ nhất/lớn nhất.

Ví dụ:

```
SELECT AVG(list_price) AS AVG_list_price FROM products;
```

Hàm chuỗi:

- **CONCAT():** Nối chuỗi.
- **LENGTH():** Độ dài chuỗi.
- **SUBSTRING():** Lấy chuỗi con.

Ví dụ:

```
SELECT CONCAT(first_name, ' ', last_name) FROM customers;
```

Hàm ngày tháng:

- **CURRENT_DATE():** Ngày hiện tại.
- **NOW():** Ngày và giờ hiện tại.
- **DATEADD():** Cộng thêm ngày/tháng.
- **DATEPART():** Trích xuất một phần cụ thể của ngày như năm, tháng, hoặc ngày.
- **DATEDIFF():** Tính khoảng cách giữa hai ngày.
- **FORMAT():** Định dạng ngày tháng theo kiểu mong muốn.

Ví dụ:

```
SELECT CURRENT_DATE AS today_date, NOW() AS current_time;
```

16. Trình bày về join, 2 table ko có relationship có join được ko?

Các loại JOIN phổ biến:

- **INNER JOIN**: Chỉ trả về các hàng có điều kiện khớp ở cả hai bảng.
- **LEFT JOIN (LEFT OUTER JOIN)**: Trả về tất cả các hàng từ bảng trái và các hàng khớp từ bảng phải.
- **RIGHT JOIN (RIGHT OUTER JOIN)**: Trả về tất cả các hàng từ bảng phải và các hàng khớp từ bảng trái.
- **FULL OUTER JOIN**: Trả về tất cả các hàng khi có sự khớp hoặc không từ cả hai bảng.

Ví dụ:

```
SELECT o.order_id, c.first_name, c.last_name
FROM orders o
INNER JOIN customers c
ON o.customer_id = c.customer_id;
```

Có thể JOIN hai bảng không có quan hệ: bằng cách sử dụng CROSS JOIN.

Ví dụ:

```
SELECT * FROM products p CROSS JOIN brands b;
```

17. Trình bày về view

VIEW:

- Là bảng ảo được tạo từ một câu truy vấn SQL, không chứa dữ liệu thực tế, nó chỉ là một "cửa sổ" để xem dữ liệu từ các bảng khác.
- Nó giúp đơn giản hóa truy vấn phức tạp và tăng tính bảo mật.

Ví dụ:

```
CREATE VIEW customer_orders AS
SELECT c.customer_id,
       c.first_name,
       o.order_id,
       o.order_date
FROM customers c
```

```
JOIN orders o
                ON c.customer_id = o.customer_id;
```

18. Trình bày về subquery

Subquery: là truy vấn lồng trong một truy vấn khác.

Ví dụ:

```
SELECT first_name,
        last_name
FROM customers
WHERE customer_id IN (
                        SELECT customer_id
                        FROM orders
                        WHERE order_date > '2025-01-01'
                        );
```

19. Trình bày về ranking, row_number, dense_rank

ROW_NUMBER(): gán một số thứ tự duy nhất cho mỗi hàng trong kết quả truy vấn, bắt đầu từ 1. Số thứ tự này được tính dựa trên thứ tự sắp xếp được chỉ định trong mệnh đề ORDER BY.

Ví dụ:

```
SELECT * FROM (
                SELECT *,
                ROW_NUMBER() OVER (
                                ORDER BY list_price DESC
                                ) AS ROW_NUM
                FROM [production].[products]
                ) AS SubQuerys
WHERE ROW_NUM = 3
```

Results Messages

	product_id	product_name	list_price	row_num
1	155	Trek Domane SLR 9 Disc - 2018	11999.99	1
2	149	Trek Domane SLR 8 Disc - 2018	7499.99	2
3	156	Trek Domane SL Frameset - 2018	6499.99	3
4	157	Trek Domane SL Frameset Women's - 2018	6499.99	4
5	169	Trek Emonda SLR 8 - 2018	6499.99	5
6	51	Trek Silque SLR 8 Women's - 2017	6499.99	6
7	50	Trek Silque SLR 7 Women's - 2017	5999.99	7
8	56	Trek Domane SLR 6 Disc - 2017	5499.99	8
9	177	Trek Domane SLR 6 Disc - 2018	5499.99	9
10	154	Trek Domane SLR 6 Disc Women's - 2018	5499.99	10
11	148	Trek Domane SL 8 Disc - 2018	5499.99	11
12	47	Trek Remedy 9.8 - 2017	5299.99	12
13	43	Trek Fuel EX 9.8 27.5 Plus - 2017	5299.99	13

RANK(): gán thứ hạng cho các hàng dựa trên giá trị của cột được chỉ định trong ORDER BY. Nếu có nhiều hàng có cùng giá trị, chúng sẽ được xếp hạng giống nhau, và các hàng tiếp theo sẽ bỏ qua các thứ hạng bị trùng.

Ví dụ:

```
SELECT * FROM (
                    SELECT *,
                    RANK() OVER (
                                        ORDER BY list_price DESC
                                        ) AS ROW_NUM
                    FROM [production].[products]
                    ) AS SubQuerys
```

Results Messages

	product_id	product_name	brand_id	category_id	model_year	list_price	ROW_NUM
1	155	Trek Domane SLR 9 Disc - 2018	9	7	2018	11999.99	1
2	149	Trek Domane SLR 8 Disc - 2018	9	7	2018	7499.99	2
3	156	Trek Domane SL Frameset - 2018	9	7	2018	6499.99	3
4	157	Trek Domane SL Frameset Women's - 2018	9	7	2018	6499.99	3
5	169	Trek Emonda SLR 8 - 2018	9	7	2018	6499.99	3
6	51	Trek Silque SLR 8 Women's - 2017	9	7	2017	6499.99	3
7	50	Trek Silque SLR 7 Women's - 2017	9	7	2017	5999.99	7
8	56	Trek Domane SLR 6 Disc - 2017	9	7	2017	5499.99	8
9	177	Trek Domane SLR 6 Disc - 2018	9	7	2018	5499.99	8
10	154	Trek Domane SLR 6 Disc Women's - 2018	9	7	2018	5499.99	8
11	148	Trek Domane SL 8 Disc - 2018	9	7	2018	5499.99	8
12	47	Trek Remedy 9.8 - 2017	9	6	2017	5299.99	12
13	43	Trek Fuel EX 9.8 27.5 Plus - 2017	9	6	2017	5299.99	12
14	40	Trek Fuel EX 9.8 29 - 2017	9	6	2017	4999.99	14
15	61	Trek Powerfly 8 FS Plus - 2017	9	5	2017	4999.99	14

DENSE_RANK(): tương tự như `RANK()`, nhưng không bỏ qua các thứ hạng bị trùng. Nghĩa là, nếu có nhiều hàng có cùng giá trị, chúng sẽ được xếp hạng giống nhau, và các hàng tiếp theo sẽ nhận thứ hạng liên tiếp.

Ví dụ:

```
SELECT * FROM (
                    SELECT *,
                    DENSE_RANK() OVER (
                                        ORDER BY list_price DESC
                                        ) AS ROW_NUM
                    FROM [production].[products]
                    ) AS SubQuerys
```

Results Messages

	product_id	product_name	brand_id	category_id	model_year	list_price	ROW_NUM
1	155	Trek Domane SLR 9 Disc - 2018	9	7	2018	11999.99	1
2	149	Trek Domane SLR 8 Disc - 2018	9	7	2018	7499.99	2
3	156	Trek Domane SL Frameset - 2018	9	7	2018	6499.99	3
4	157	Trek Domane SL Frameset Women's - 2018	9	7	2018	6499.99	3
5	169	Trek Emonda SLR 8 - 2018	9	7	2018	6499.99	3
6	51	Trek Silque SLR 8 Women's - 2017	9	7	2017	6499.99	3
7	50	Trek Silque SLR 7 Women's - 2017	9	7	2017	5999.99	4
8	56	Trek Domane SLR 6 Disc - 2017	9	7	2017	5499.99	5
9	177	Trek Domane SLR 6 Disc - 2018	9	7	2018	5499.99	5
10	154	Trek Domane SLR 6 Disc Women's - 2018	9	7	2018	5499.99	5
11	148	Trek Domane SL 8 Disc - 2018	9	7	2018	5499.99	5
12	47	Trek Remedy 9.8 - 2017	9	6	2017	5299.99	6
13	43	Trek Fuel EX 9.8 27.5 Plus - 2017	9	6	2017	5299.99	6
14	40	Trek Fuel EX 9.8 29 - 2017	9	6	2017	4999.99	7
15	61	Trek Powerfly 8 FS Plus - 2017	9	5	2017	4999.99	7

20. trình bày về CTE

CTE (Common Table Expression): cho phép bạn tạo một bảng tạm thời (temporary result set) mà bạn có thể tham chiếu trong phạm vi của một câu truy vấn. CTE giúp cải thiện khả năng đọc và bảo trì mã SQL, đặc biệt khi làm việc với các truy vấn phức tạp hoặc lồng nhau.

Ví dụ:

```
WITH ProductSales AS (
                    SELECT product_id,
                    SUM(quantity) AS total_quantity
                    FROM order_items
                    GROUP BY product_id
                    )
SELECT * FROM ProductSales WHERE total_quantity > 10;
```

21. Trình bày về các câu lệnh điều khiển trong SQL

Các câu lệnh điều khiển phổ biến trong SQL:

- **IF...ELSE:** Rẽ nhánh dựa trên điều kiện.

Ví dụ:

```
DECLARE @quantity INT = 7;
IF @quantity > 5
        PRINT 'Số lượng lớn hơn 5';
ELSE
        PRINT 'Số lượng nhỏ hơn hoặc bằng 5';
```

- **CASE:** So sánh và trả về giá trị tương ứng.

Ví dụ:

```
SELECT product_id, product_name,
        CASE
                WHEN list_price > 1000 THEN 'Expensive'
                WHEN list_price BETWEEN 500 AND 1000
                THEN 'Medium' ELSE 'Cheap'
        END AS price_category
FROM products;
```

- **LOOP, WHILE, REPEAT:** Thực hiện vòng lặp.

Ví dụ:

```
DECLARE @counter INT = 1;
WHILE @counter <= 5
BEGIN
        PRINT 'Lặp số ' + CAST(@counter AS VARCHAR);
        SET @counter = @counter + 1;
END;
```

- **DECLARE HANDLER:** Xử lý ngoại lệ.

Ví dụ:

```
BEGIN TRY
        SELECT 1 / 0;
END TRY
BEGIN CATCH
        PRINT 'Có lỗi xảy ra: ' + ERROR_MESSAGE();
END CATCH;
```

- **GOTO:** Nhảy đến nhãn cụ thể.
- **RETURN:** Trả về giá trị từ hàm.

Ví dụ:

```
CREATE FUNCTION get_discount(@price DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
AS
BEGIN
        IF @price > 100
                RETURN @price * 0.1;
```

```
        ELSE
                RETURN @price * 0.05;
END;
```

- **LEAVE:** Thoát khỏi vòng lặp hoặc khối lệnh.
- **ITERATE:** Bỏ qua phần còn lại của vòng lặp và tiếp tục lần lặp tiếp theo.

22. Trình bày về UDF trong sql

UDF (User-Defined Function): là hàm do người dùng tạo ra để thực hiện các thao tác lặp lại trong cơ sở dữ liệu.
Có hai loại chính:

- **Scalar Function:** Trả về một giá trị đơn.
- **Table-Valued Function:** Trả về một bảng.

Ví dụ:

```
CREATE FUNCTION calculate_tax(@price DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
AS
BEGIN
        RETURN @price * 0.1;
END;
```

23. Trình bày về SP trong SQL

Stored Procedure: là một tập hợp câu lệnh SQL được lưu trữ trong cơ sở dữ liệu và có thể gọi thực thi nhiều lần.
Ví dụ:

```
CREATE PROCEDURE get_products
AS
BEGIN
        SELECT * FROM products;
END;
```

24. Trình bày về exception trong sql

Exception trong SQL: xử lý ngoại lệ được thực hiện bằng TRY...CATCH để phát hiện lỗi trong quá trình thực thi.
Ví dụ:

```
BEGIN TRY
        SELECT 1 / 0;
END TRY
BEGIN CATCH
        PRINT 'Có lỗi: ' + ERROR_MESSAGE();
END CATCH;
```

25. Trình bày về transaction trong sql

Transaction: là một nhóm các thao tác SQL được thực thi như một đơn vị.

Các câu lệnh cơ bản:

- **BEGIN TRANSACTION:** Bắt đầu giao dịch.
- **COMMIT:** Xác nhận thay đổi.
- **ROLLBACK:** Hủy giao dịch nếu có lỗi.

Ví dụ:

```
BEGIN TRANSACTION;
UPDATE products SET stock = stock - 1 WHERE product_id = 1;
IF @@ERROR <> 0
        ROLLBACK;
ELSE
        COMMIT;
```

26. Trình bày về Trigger trong sql

Trigger (Trình kích hoạt): là một loại stored procedure đặc biệt được tự động thực thi khi một sự kiện cụ thể xảy ra trong cơ sở dữ liệu, thường được sử dụng để thực hiện các tác vụ tự động như kiểm tra ràng buộc phức tạp

Đặc điểm của Trigger:

- **Tự động thực thi:** Trigger được kích hoạt tự động khi có sự kiện xảy ra (ví dụ: `INSERT`, `UPDATE`, `DELETE`).
- **Không có tham số:** Trigger không nhận tham số đầu vào.
- **Không trả về giá trị:** Trigger không trả về giá trị, nhưng có thể thay đổi dữ liệu trong cơ sở dữ liệu.
- **Có thể lồng nhau:** Trigger có thể gọi các trigger khác (nested triggers).
- **Có thể hủy bỏ thao tác:** Trigger có thể hủy bỏ thao tác gốc bằng cách sử dụng `ROLLBACK`.

Có hai loại trigger chính

1. **DML Trigger (Data Manipulation Language Trigger)**:
    - Được kích hoạt khi có các sự kiện `INSERT`, `UPDATE`, hoặc `DELETE`.
    - Có thể được định nghĩa ở mức **hàng (row-level)** hoặc **câu lệnh (statement-level)**.
2. **DDL Trigger (Data Definition Language Trigger)**:
    - Được kích hoạt khi có các sự kiện liên quan đến thay đổi cấu trúc cơ sở dữ liệu, chẳng hạn như `CREATE`, `ALTER`, hoặc `DROP`.

Ví dụ:

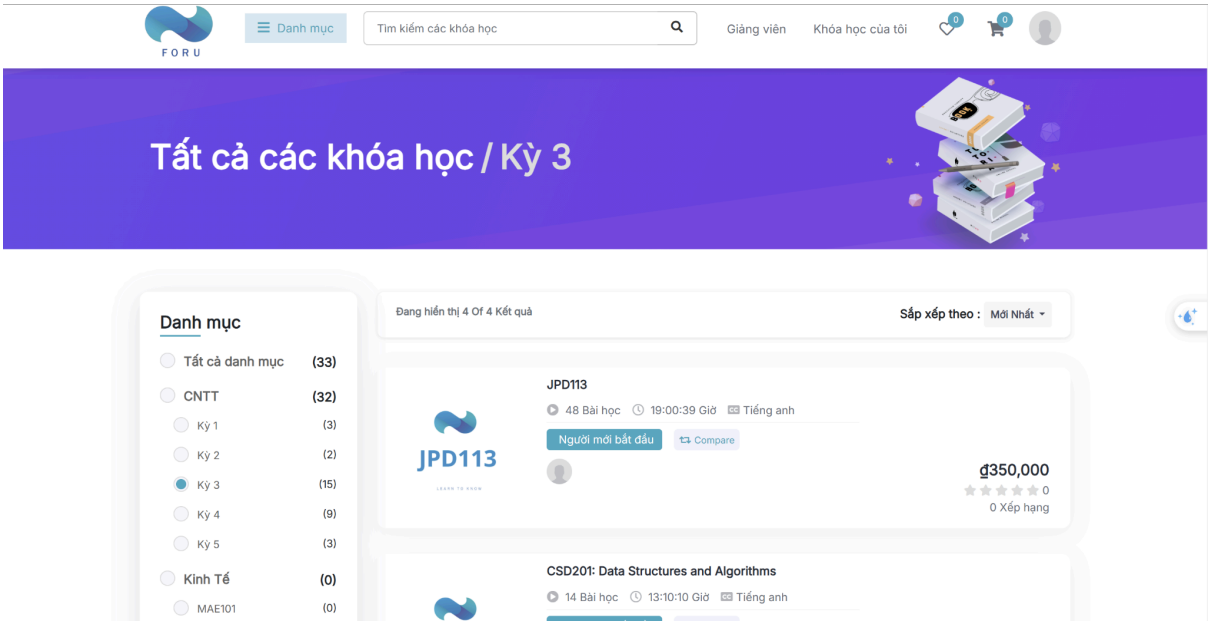
```
CREATE TRIGGER trg_update_stock
ON orders
AFTER INSERT
AS
BEGIN
        UPDATE products
        SET stock = stock - 1
        WHERE product_id = (SELECT product_id FROM inserted);
END;
```

27. Trình bày về ACID

ACID là thuộc tính của hệ thống quản lý cơ sở dữ liệu đảm bảo tính toàn vẹn dữ liệu:

- **Atomicity (Tính nguyên tử)**: Giao dịch thực thi toàn bộ hoặc không thực thi gì cả.
- **Consistency (Tính nhất quán)**: Đảm bảo dữ liệu hợp lệ trước và sau giao dịch.
- **Isolation (Tính biệt lập)**: Giao dịch không ảnh hưởng đến nhau.
- **Durability (Tính bền vững)**: Dữ liệu được lưu vĩnh viễn sau khi giao dịch hoàn tất.

Mình để chế độ mọi người có thể down về để học và bổ sung cho dễ hiểu, nhưng share cho bạn bè thì đừng xóa phần này nha ^^



Mọi người cần

Mua src FE lọc từ EOS theo từng dạng câu kèm note chất lượng các môn chuyên ngành SE, vừa học src vừa học hiểu

Cần đki khóa học cho kỳ 4, các môn cảm thấy học khó khăn

Hãy inb https://www.facebook.com/profile.php?id=100095690977025

hoặc page: https://www.facebook.com/profile.php?id=100069135802463

Link web : https://4user.net/home/courses

“Nếu có ích thì quay lại cảm ơn câu đeeeee”