# Database

**Query**

# **1.** Trong một câu query, thứ tự thực hiện của các thành phần như thế nào**?**

### **Thứ tự thực hiện của các thành phần trong một câu truy vấn SQL:**

1. **FROM**:
   * Bước đầu tiên là chọn bảng (hoặc các bảng) từ đó dữ liệu sẽ được lấy ra.
   * Nếu có **JOIN**, các bảng sẽ được kết hợp với nhau ở bước này.
2. **WHERE**:
   * Sau khi dữ liệu được lấy từ các bảng ở bước FROM, điều kiện lọc sẽ được áp dụng. WHERE xác định các hàng nào sẽ được giữ lại dựa trên điều kiện nhất định.
3. **GROUP BY**:
   * Sau khi lọc dữ liệu ở bước WHERE, dữ liệu có thể được nhóm lại dựa trên một hoặc nhiều cột (hoặc biểu thức).
   * Các nhóm được tạo dựa trên các giá trị khác nhau của cột/biểu thức đã chỉ định.
4. **HAVING**:
   * HAVING tương tự như WHERE, nhưng được áp dụng sau khi dữ liệu đã được nhóm lại (sau GROUP BY). Nó được sử dụng để lọc các nhóm dữ liệu.
5. **SELECT**:
   * Sau khi dữ liệu đã được lọc và nhóm, các cột hoặc biểu thức cụ thể sẽ được chọn để hiển thị.
   * Lúc này, các phép tính tổng hợp như SUM(), AVG(), COUNT(),... sẽ được thực hiện nếu có.
6. **DISTINCT**:
   * Nếu có từ khóa DISTINCT, nó sẽ loại bỏ các giá trị trùng lặp trong kết quả của câu truy vấn.
7. **ORDER BY**:
   * Sau khi dữ liệu đã được chọn, kết quả có thể được sắp xếp theo một hoặc nhiều cột cụ thể.
8. **LIMIT / OFFSET** (hoặc **FETCH FIRST**):
   * Cuối cùng, nếu có giới hạn số lượng kết quả cần trả về, LIMIT hoặc OFFSET sẽ được áp dụng để chỉ trả về một số lượng hàng cụ thể.

# 2.Các loại JOIN trong database?

### **1. INNER JOIN**

* **Mô tả**: INNER JOIN trả về các hàng có giá trị phù hợp trong cả hai bảng. Nếu không có kết quả phù hợp, hàng đó sẽ bị loại bỏ.
* **Ứng dụng**: Dùng khi bạn chỉ muốn lấy những hàng mà có dữ liệu tương ứng ở cả hai bảng.

### **2. LEFT JOIN (Hoặc LEFT OUTER JOIN)**

* **Mô tả**: LEFT JOIN trả về tất cả các hàng từ bảng bên trái (bảng đầu tiên) và những hàng phù hợp từ bảng bên phải. Nếu không có hàng nào phù hợp ở bảng bên phải, giá trị trả về sẽ là NULL.
* **Ứng dụng**: Dùng khi bạn muốn lấy tất cả dữ liệu từ bảng bên trái và chỉ lấy dữ liệu từ bảng bên phải nếu có kết quả tương ứng.

### **3. RIGHT JOIN (Hoặc RIGHT OUTER JOIN)**

* **Mô tả**: RIGHT JOIN trả về tất cả các hàng từ bảng bên phải và những hàng phù hợp từ bảng bên trái. Nếu không có kết quả phù hợp ở bảng bên trái, giá trị trả về sẽ là NULL.
* **Ứng dụng**: Dùng khi bạn muốn lấy tất cả dữ liệu từ bảng bên phải và chỉ lấy dữ liệu từ bảng bên trái nếu có kết quả tương ứng.

### **4. FULL JOIN (Hoặc FULL OUTER JOIN)**

* **Mô tả**: FULL JOIN kết hợp kết quả của cả LEFT JOIN và RIGHT JOIN. Nó trả về tất cả các hàng khi có sự phù hợp trong bảng trái hoặc bảng phải. Nếu không có sự phù hợp, các cột không có giá trị phù hợp sẽ chứa NULL.
* **Ứng dụng**: Dùng khi bạn muốn lấy tất cả dữ liệu từ cả hai bảng, bất kể có sự tương ứng hay không.

### **5. CROSS JOIN**

* **Mô tả**: CROSS JOIN trả về tích Cartesian của hai bảng. Điều này có nghĩa là mỗi hàng trong bảng đầu tiên sẽ được kết hợp với **tất cả** các hàng trong bảng thứ hai.
* **Ứng dụng**: Dùng khi bạn muốn kết hợp tất cả các hàng của hai bảng lại với nhau mà không cần điều kiện nào.

### **6. SELF JOIN**

* **Mô tả**: SELF JOIN là phép JOIN một bảng với chính nó. Nó hữu ích khi bạn muốn so sánh các hàng trong cùng một bảng với nhau.
* **Ứng dụng**: Dùng khi có dữ liệu đệ quy hoặc khi cần so sánh giữa các bản ghi trong cùng một bảng.

| Loại JOIN | Mô tả |
| --- | --- |
| INNER JOIN | Trả về các hàng có sự tương ứng giữa cả hai bảng. |
| LEFT JOIN | Trả về tất cả các hàng từ bảng bên trái, và các hàng tương ứng từ bảng bên phải (nếu có). |
| RIGHT JOIN | Trả về tất cả các hàng từ bảng bên phải, và các hàng tương ứng từ bảng bên trái (nếu có). |
| FULL JOIN | Trả về tất cả các hàng từ cả hai bảng. Nếu không có hàng tương ứng, giá trị sẽ là NULL. |
| CROSS JOIN | Trả về tích Cartesian của hai bảng. |
| SELF JOIN | JOIN một bảng với chính nó. |

# 3. Các loại JOIN trong database?

Subquery trong cơ sở dữ liệu cho phép bạn lồng một truy vấn bên trong một truy vấn khác. Chúng có thể được sử dụng trong nhiều mệnh đề khác nhau như SELECT, FROM, WHERE, và HAVING. Dưới đây là một số thông tin về subquery kèm theo ví dụ trong SQL, đặc biệt là với MySQL.

### **Các loại Subquery**

1. **Single-Row Subquery**: Trả về một hàng duy nhất và có thể được sử dụng với các toán tử so sánh.
2. **Multiple-Row Subquery**: Trả về nhiều hàng và có thể được sử dụng với IN, ANY, hoặc ALL.
3. **Correlated Subquery**: Một subquery tham chiếu đến các cột từ truy vấn bên ngoài.

# 4. CTE (Common Table Expression) ?

Common Table Expressions (CTEs) là một tính năng trong SQL cho phép bạn tạo ra một bảng tạm thời có thể được sử dụng trong một truy vấn. CTE thường được sử dụng để làm cho mã SQL dễ đọc hơn, phân chia các phần phức tạp thành những phần dễ hiểu hơn và cũng có thể giúp thực hiện các truy vấn đệ quy.

### **Lợi ích của CTE**

* **Dễ đọc**: CTE giúp mã SQL trở nên dễ đọc hơn bằng cách phân chia các phần phức tạp thành các phần riêng biệt.
* **Có thể tái sử dụng**: Một CTE có thể được sử dụng nhiều lần trong cùng một truy vấn.
* **Hỗ trợ truy vấn đệ quy**: CTE cho phép thực hiện các truy vấn đệ quy, giúp xử lý các cấu trúc dữ liệu phức tạp.

# 5. ROW\_NUMBER, RANK, DENSE\_RANK?

### **1. ROW\_NUMBER():**

ROW\_NUMBER() trả về số thứ tự duy nhất cho mỗi hàng trong tập hợp kết quả, không quan tâm đến việc có các hàng có giá trị bằng nhau hay không. Kết quả luôn là số nguyên tăng dần liên tục, bất kể giá trị của các hàng.

### **2. RANK():**

RANK() trả về thứ hạng của mỗi hàng trong tập hợp kết quả, với các hàng có giá trị bằng nhau sẽ nhận cùng một thứ hạng. Tuy nhiên, thứ hạng tiếp theo sẽ bị bỏ qua tương ứng với số hàng trùng lặp.

### **3. DENSE\_RANK():**

DENSE\_RANK() cũng gán thứ hạng cho các hàng có giá trị bằng nhau giống như RANK(), nhưng không bỏ qua các thứ hạng tiếp theo. Thứ hạng sẽ tiếp tục liên tục.

# 6. VIEW : Có thể CRUD trên VIEW được không?

**VIEW** là một bảng ảo được tạo ra từ kết quả của một truy vấn. Nó không lưu trữ dữ liệu thực sự mà chỉ lưu trữ câu truy vấn, và mỗi lần bạn truy vấn VIEW, hệ thống sẽ chạy câu truy vấn để lấy dữ liệu.

View không thể CRUD.

# 7. PROCEDURE : mục đích khi tạo ra 1 PROCEDURE là gì ? Làm thế nào để sử dụng PROCEDURE trong ứng dụng java?

**Stored Procedure** là một tập hợp các câu lệnh SQL đã được biên dịch và lưu trữ trong cơ sở dữ liệu. Mục đích chính của việc tạo ra một PROCEDURE bao gồm:

1. **Tái sử dụng mã**: Giúp giảm thiểu việc lặp lại mã SQL. Một PROCEDURE có thể được gọi ở nhiều nơi trong ứng dụng mà không cần viết lại toàn bộ câu lệnh SQL.
2. **Tối ưu hóa hiệu suất**: Vì PROCEDURE được biên dịch và tối ưu hóa trước khi lưu trữ, chúng có thể thực hiện nhanh hơn so với các câu lệnh SQL thông thường được chạy trực tiếp từ ứng dụng.
3. **Bảo mật và quản lý quyền truy cập**: Bạn có thể cấp quyền cho người dùng chỉ được phép gọi PROCEDURE mà không cần truy cập trực tiếp vào bảng dữ liệu, giúp tăng cường bảo mật.
4. **Logic phức tạp**: PROCEDURE cho phép thực hiện các logic phức tạp với các vòng lặp, điều kiện (IF, CASE), và xử lý lỗi. Điều này giúp di chuyển một số phần xử lý dữ liệu từ ứng dụng sang cơ sở dữ liệu.
5. **Quản lý giao dịch**: PROCEDURE có thể bao gồm nhiều câu lệnh SQL và có thể quản lý giao dịch (transaction) một cách dễ dàng (sử dụng BEGIN, COMMIT, ROLLBACK).