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

* Trong 1 câu query thứ tự thực hiện khác so với thứ tự xuất hiện trong câu Thứ tự thực hiện của các thành phần trong một câu query SQL là:
  + **1 FROM**: Hệ quản trị cơ sở dữ liệu (DBMS) đầu tiên sẽ tìm kiếm các bảng hoặc view được chỉ định trong câu lệnh.
  + **2 JOIN**: Nếu có phép kết hợp giữa các bảng, bước này sẽ xử lý các phép kết hợp (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, v.v.).
  + **3 WHERE**: Điều kiện lọc hàng dựa trên điều kiện chỉ định trong mệnh đề WHERE.
  + **4 GROUP BY**: Các hàng sau khi lọc sẽ được nhóm lại dựa trên các cột trong mệnh đề GROUP BY.
  + **5 HAVING**: Sau khi nhóm, mệnh đề HAVING sẽ áp dụng điều kiện để lọc các nhóm.
  + **6 SELECT**: Sau khi các hàng và nhóm được xử lý, DBMS sẽ chọn các cột hoặc biểu thức được chỉ định trong SELECT.
  + **7 DISTINCT**: Loại bỏ các bản ghi trùng lặp nếu từ khóa DISTINCT được sử dụng.
  + **8 ORDER BY**: Sắp xếp kết quả dựa trên các cột được chỉ định trong ORDER BY.
  + **9 LIMIT/OFFSET**: Giới hạn số lượng kết quả trả về và bỏ qua các kết quả đầu tiên nếu có LIMIT và OFFSET.
* Ví dụ : Câu query sau
  + SELECT DISTINCT name, COUNT(\*) FROM employees WHERE salary > 5000 GROUP BY department\_id HAVING COUNT(\*) > 10 ORDER BY name LIMIT 5;
  + thứ tự thực hiện :
    - FROM employees
    - WHERE salary > 5000
    - GROUP BY department\_id
    - HAVING COUNT(\*) > 10
    - SELECT DISTINCT name, COUNT(\*)
    - ORDER BY name
    - LIMIT 5

1. Các loại Join

* **INNER JOIN**
  + **Đặc điểm**: Chỉ trả về các bản ghi có giá trị tương ứng trong cả hai bảng. Nếu không có giá trị tương ứng trong một trong hai bảng, bản ghi đó sẽ không được hiển thị.
  + **Cách hoạt động**:Kết hợp hai bảng chỉ khi các điều kiện trong câu ON là đúng.
  + Ví dụ : SELECT customers.name, orders.order\_id FROM customers INNER JOIN orders ON customers.customer\_id = orders.customer\_id.
  + Trong ví dụ này, chỉ những khách hàng có đơn hàng trong bảng orders mới được hiển thị.
* **LEFT JOIN** 
  + **Đặc điểm**: Trả về tất cả các bản ghi từ bảng bên trái ngay cả khi không có bản ghi tương ứng trong bảng bên phải Nếu không có bản ghi tương ứng, các cột từ bảng bên phải sẽ trả về giá trị NULL.
  + **Cách hoạt động**:Tất cả dữ liệu từ bảng bên trái được giữ lại, và nếu không có giá trị tương ứng trong bảng bên phải thì sẽ trả về NULL cho các cột từ bảng bên phải.
  + Ví dụ : SELECT customers.name, orders.order\_id FROM customers LEFT JOIN orders ON customers.customer\_id = orders.customer\_id;
  + Trong ví dụ này, tất cả khách hàng sẽ được hiển thị, bao gồm cả những khách hàng không có đơn hàng.
* **RIGHT JOIN** 
  + **Đặc điểm**: Ngược lại với LEFT JOIN , trả về tất cả các bản ghi từ bảng bên phải, ngay cả khi không có bản ghi tương ứng trong bảng bên trái .
  + **Cách hoạt động**:Tất cả dữ liệu từ bảng bên phải được giữ lại, và nếu không có giá trị tương ứng trong bảng bên trái thì sẽ trả về NULL cho các cột từ bảng bên trái.
  + Ví dụ : SELECT customers.name, orders.order\_id FROM customers RIGHT JOIN orders ON customers.customer\_id = orders.customer\_id;
  + Trong ví dụ này, tất cả đơn hàng sẽ được hiển thị, ngay cả khi chúng không liên kết với bất kỳ khách hàng nào.
* **FULL JOIN (FULL OUTER JOIN)**
  + **Đặc điểm**: Kết hợp tất cả các bản ghi từ cả hai bảng. Nếu không có giá trị tương ứng trong một trong hai bảng, các cột từ bảng không có giá trị sẽ trả về NULL.
  + **Cách hoạt động**:Trả về tất cả các bản ghi từ cả hai bảng, ngay cả khi không có bản ghi tương ứng trong bảng kia.
  + Ví dụ : SELECT customers.name, orders.order\_id FROM customers FULL JOIN orders ON customers.customer\_id = orders.customer\_id;
  + Trong ví dụ này, tất cả các khách hàng và đơn hàng sẽ được hiển thị. Nếu khách hàng không có đơn hàng hoặc ngược lại, các cột tương ứng sẽ hiển thị NULL.
* **CROSS JOIN**
  + **Đặc điểm**: Tạo ra kết hợp giữa tất cả các bản ghi từ cả hai bảng, tức là số lượng bản ghi kết quả sẽ là tích của số bản ghi trong cả hai bảng. CROSS JOIN không yêu cầu điều kiện ON.
  + **Cách hoạt động**: Mỗi bản ghi từ bảng đầu tiên sẽ được kết hợp với tất cả các bản ghi từ bảng thứ hai.
  + Ví dụ : SELECT customers.name, orders.order\_id FROM customers CROSS JOIN orders;
  + Trong ví dụ này, nếu bảng customers có 10 bản ghi và bảng orders có 5 bản ghi, kết quả sẽ có 50 bản ghi (10 \* 5).

1. SubQuery

* **Subquery** (truy vấn con) trong SQL là một truy vấn nằm bên trong một truy vấn khác.
* **Cách triển khai** : Subquery thường được đặt trong dấu ngoặc đơn và được sử dụng để trả về một kết quả tạm thời, sau đó kết quả này có thể được dùng làm điều kiện lọc hoặc tính toán trong truy vấn chính.
* **Chạy trước truy vấn chính:** Subquery sẽ được thực thi trước, kết quả của nó được sử dụng bởi truy vấn chính.
* **Độc lập với truy vấn chính:** Subquery có thể hoạt động độc lập với truy vấn chính nhưng có thể phụ thuộc vào các tham chiếu trong truy vấn chính.
* **Sử dụng khi** : muốn tách nhỏ các truy vấn phức tạp thành các phần dễ quản lý hơn và tối ưu hóa kết quả

1. CTE

* Định nghĩa: CTE (Common Table Expression) là một biểu thức bảng tạm thời (temporary result set) mà bạn có thể định nghĩa trong một câu truy vấn SQL.
* Một số đặc điểm chính của CTE:
  + Được định nghĩa tạm thời trong phạm vi một câu truy vấn.
  + Có thể sử dụng nhiều lần trong cùng câu truy vấn.
  + Giúp câu truy vấn dễ hiểu hơn, đặc biệt khi cần làm việc với các truy vấn con phức tạp.
* **Cách hoạt động:**CTE được khai báo bằng từ khóa WITH theo sau là tên của CTE, một danh sách các cột tùy chọn, và một câu truy vấn trả về kết quả cho CTE. Sau đó, CTE này có thể được sử dụng giống như một bảng trong câu truy vấn chính.
* ví dụ : WITH DepartmentAvgSalary AS ( SELECT Department, AVG(Salary) AS AvgSalary FROM Employees GROUP BY Department ) SELECT e.Name, e.Department, e.Salary FROM Employees e JOIN DepartmentAvgSalary d ON e.Department = d.Department WHERE e.Salary > d.AvgSalary;
* Kết quả trả về sẽ là những nhân viên có mức lương cao hơn mức trung bình của phòng ban họ.

1. RANK

* Là các hàm thứ hạng được sử dụng để gán số thứ tự cho các hàng trong một tập kết quả dựa trên thứ tự.
* **ROW\_NUMBER**
  + **Đặc điểm**: Hàm ROW\_NUMBER trả về một số thứ tự duy nhất cho mỗi hàng trong kết quả, không quan tâm đến việc các giá trị trong các hàng đó có bằng nhau hay không.
  + **Cách triển khai**: Được sử dụng để đánh số thứ tự cho từng hàng mà không phân biệt giá trị.
  + Ví dụ : SELECT name, salary, ROW\_NUMBER() OVER (ORDER BY salary DESC) AS row\_num FROM employees;
  + Trong ví dụ trrnhân viên sẽ được gán thứ tự dù có cùng lương.
* **RANK**
  + **Đặc điểm**: Hàm RANK sẽ gán cùng một giá trị thứ hạng cho các hàng có cùng giá trị, nhưng nó sẽ bỏ qua thứ tự tiếp theo.
  + **Cách triển khai**: Thích hợp khi muốn duy trì khoảng cách giữa các thứ hạng.
  + Ví dụ : SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rank FROM employees;
  + Trong ví dụ trên nhân viên có cùng mưacs lương đánh tứ tự giống nhau và nhân viên tiếp theo sẽ bỏ qua 1 số thứ tự vd có 2 nhân viên đc đánh là 2 thì tiếp thoe là 4 chứ ko phải 3.
* **DENSE\_RANK**
  + **Đặc điểm**: Hàm DENSE\_RANK tương tự như RANK, nhưng không bỏ qua thứ hạng tiếp theo, dù có các hàng có giá trị giống nhau.
  + **Cách triển khai**: Sử dụng khi không muốn có khoảng trống trong thứ hạng.
  + Ví dụ : SELECT name, salary, DENSE\_RANK() OVER (ORDER BY salary DESC) AS dense\_rank FROM employees;
  + Trong ví dụ trên nhân viên có cùng mức lương đánh tứ tự giống nhau vàvd có 2 nhân viên đc đánh là 2 thì tiếp theo vẫn là 3 .

1. View

* Định nghĩa: View trong SQL là một bảng ảo (virtual table) được tạo ra từ kết quả của một query. View không lưu trữ dữ liệu thực tế, mà chỉ lưu trữ câu truy vấn để tạo ra bảng.
* Đặc điểm :
  + **Bảo mật**: View có thể giúp che giấu cấu trúc của bảng thực hoặc ẩn đi các cột không cần thiết, giúp tăng tính bảo mật.
  + **Tái sử dụng**: View có thể được sử dụng lại nhiều lần, giúp đơn giản hóa các câu truy vấn phức tạp.
  + **Tính nhất quán**: View có thể đảm bảo tính nhất quán cho các truy vấn phức tạp bằng cách định nghĩa trước logic cần thiết.
* Có 2 loại view
  + **Simple View:** Được tạo từ một bảng duy nhất, không chứa các hàm tổng hợp, GROUP BY, hoặc JOIN.Dễ dàng cập nhập và chỉnh sửa dữ liệu trực tiếp.
  + **Complex View:** Được tạo từ nhiều bảng, có thể bao gồm các phép tính tổng hợp như SUM, AVG, hoặc các phép nối (JOIN).Khó khaăn khi cập nhập và chỉnh sửa view.
* Cách triển khai view
  + Khởi tạo 1 view mới bằng câu lệnh CREAT VIEWview AS câu query vào view.
  + khi sử dụng view ta dùng cậu lệnh SELECT 8 FROM VIEW và view sẽ thực thi lai câu query gốc luuw trữ trong view.
  + Muốn update 1 view ta sử dụng câu lệnh CREATE OR REPLACE VIEW.
  + Xoá view đang có sẵn ta dùng lệnh DROP VIEW.

1. Procedure

* **Định nghĩa** : **PROCEDURE** là một tập hợp các câu lệnh SQL đã được biên dịch và lưu trữ trong cơ sở dữ liệu.
* Mục đích tạo ra Procedure:
  + **Tăng tính tái sử dụng**: PROCEDURE giúp bạn tái sử dụng logic SQL mà không cần phải viết lại nhiều lần trong các ứng dụng khác nhau.
  + **Tăng hiệu suất**: Các PROCEDURE đã được biên dịch trước đó nên chạy nhanh hơn so với câu lệnh SQL thông thường.
  + **Dễ bảo trì**: Mọi thay đổi chỉ cần thực hiện một lần trong PROCEDURE thay vì ở nhiều nơi khác nhau trong ứng dụng.
  + **Bảo mật**: Giới hạn truy cập và thao tác trực tiếp với dữ liệu, giúp bảo mật cơ sở dữ liệu tốt hơn.
* **Các bước để sử dụng PROCEDURE trong Java:**
  + **Tạo kết nối đến cơ sở dữ liệu**: Sử dụng DriverManager để tạo kết nối đến cơ sở dữ liệu.
  + **Tạo đối tượng CallableStatement**: Dùng CallableStatement để gọi PROCEDURE từ Java.
  + **Gọi PROCEDURE và truyền tham số (nếu có)**: Truyền tham số vào PROCEDURE nếu PROCEDURE yêu cầu.
  + **Xử lý kết quả**: Xử lý kết quả trả về từ PROCEDURE (nếu có).
  + **Đóng kết nối**: Đảm bảo đóng kết nối và các tài nguyên khác sau khi hoàn thành.