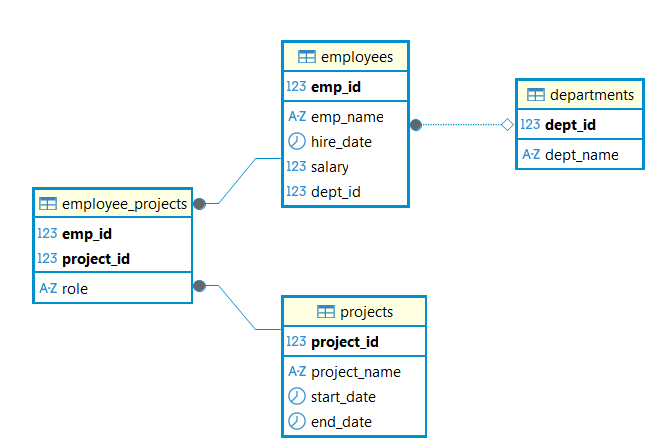
Query

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

**Thứ tự thực thi của một câu lệnh SQL cơ bản là:**

1. **FROM**: Chọn bảng hoặc các bảng nguồn dữ liệu.
2. **ON**: Lọc các hàng trong quá trình nối (JOIN) giữa các bảng.
3. **JOIN**: Kết hợp các bảng đã được lọc bởi ON.
4. **WHERE**: Lọc các hàng sau khi đã nối.
5. **GROUP BY**: Nhóm các hàng thành các nhóm dựa trên các giá trị đã chỉ định.
6. **HAVING**: Lọc các nhóm được tạo bởi GROUP BY.
7. **SELECT**: Chọn các cột để trả về.
8. **DISTINCT**: Loại bỏ các hàng trùng lặp.
9. **ORDER BY**: Sắp xếp các hàng theo một thứ tự cụ thể.
10. **LIMIT / OFFSET**: Giới hạn số hàng được trả về

2. Tìm hiểu và lấy ví dụ về :

  
+ Các loại JOIN

**INNER JOIN**

Trả về các hàng chỉ khi có sự khớp giá trị giữa các bảng.

**select** *e*.emp\_id, *e*.emp\_name , *d*.dept\_name

**from** employees *e* **inner** **join** departments *d*

**on** *e*.dept\_id = *d*.dept\_id

A screenshot of a computer

AI-generated content may be incorrect.

**LEFT JOIN (hoặc LEFT OUTER JOIN)**

Trả về tất cả các hàng từ bảng bên trái (products), và các hàng khớp từ bảng bên phải. Các hàng không khớp từ bảng bên phải sẽ có giá trị NULL.

**select** *e*.emp\_id, *e*.emp\_name , *d*.dept\_name

**from** employees *e* **left** **join** departments *d*

**on** *e*.dept\_id = *d*.dept\_id

A screenshot of a computer

AI-generated content may be incorrect.

**RIGHT JOIN (hoặc RIGHT OUTER JOIN)**

Tương tự như LEFT JOIN, nhưng trả về tất cả các hàng từ bảng bên phải (categories) và các hàng khớp từ bảng bên trái. Các hàng không khớp từ bảng bên trái sẽ có giá trị NULL.

**select** *e*.emp\_id, *e*.emp\_name , *d*.dept\_name

**from** employees *e* **right** **join** departments *d*

**on** *e*.dept\_id = *d*.dept\_id

A screenshot of a computer

AI-generated content may be incorrect.

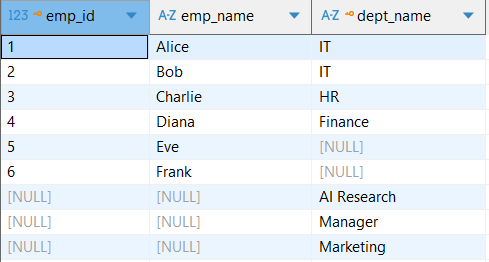
**FULL OUTER JOIN**

Trả về tất cả các hàng khi có sự khớp giữa các bảng, hoặc nếu có hàng không khớp ở một trong hai bảng. Các cột không khớp sẽ có giá trị NULL.

**select** *e*.emp\_id, *e*.emp\_name , *d*.dept\_name

**from** employees *e* **full** **outer** **join** departments *d*

**on** *e*.dept\_id = *d*.dept\_id



+ Sub query

**Subquery** là một truy vấn nằm bên trong một truy vấn SQL khác. Nó có thể được sử dụng trong các mệnh đề SELECT, FROM, WHERE, HAVING.

Subquery trong mệnh đề WHERE

**select** *e*.emp\_id, *e*.emp\_name , *e*.salary

**from** employees *e*

**where** *e*.salary > (

**select** **AVG**(*e2*.salary ) **from** employees *e2*

)

dA screenshot of a computer

AI-generated content may be incorrect.

Subquery trong mệnh đề FROM

**select** *d*.dept\_name, *s*.*salary*

**from** departments *d*

**inner** **join** (

**select** *e*.dept\_id ,**sum**(*e*.salary ) **as** *salary*

**from** employees *e*

**group** **by** *e*.dept\_id

) *s*

**on** *d*.dept\_id = *s*.dept\_id

A screenshot of a computer

AI-generated content may be incorrect.

**select** *e2*.emp\_name , *t*.*max\_salary* , *d*.dept\_name

**from** (

**select** *e*.dept\_id , **max**(*e*.salary ) **as** *max\_salary*

**from** employees *e*

**group** **by** *e*.dept\_id

) *t*

**inner** **join** employees *e2* **on** *e2*.salary = *t*.*max\_salary*

**inner** **join** departments *d* **on** *d*.dept\_id = *t*.dept\_id

A screenshot of a computer screen

AI-generated content may be incorrect.

Subquery với EXISTS

+ CTE (Common Table Expression)

Common Table Expressions (CTE) là tập hợp kết quả tạm thời được đặt tên mà có thể tham chiếu trong câu Select, Insert, Update, Delete.

Mục đích chính của CTE là làm cho các truy vấn phức tạp trở nên dễ đọc và dễ quản lý hơn. Thay vì sử dụng các truy vấn con (subquery) lồng nhau, bạn có thể chia nhỏ truy vấn thành các CTE riêng biệt, mỗi CTE thực hiện một phần logic cụ thể.

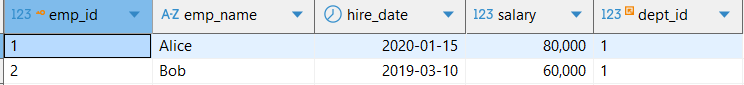
**with** *Highsalary*(*employId*) **as**(

**select** *e*.emp\_id **from** employees *e* **where** *e*.salary > 50000

)

**select** \* **from** employees *e*

**where** *e*.emp\_id **in** (**select** *employid* **from** *Highsalary* )



CTE thường được ưu tiên khi một truy vấn phức tạp và cần được chia thành các phần nhỏ hơn, dễ quản lý hơn, đồng thời nó cũng làm tăng khả năng sử dụng lại và giúp duy trì truy vấn. Trong khi subquery rất hữu ích khi truy vấn cần truy xuất dữ liệu cần thiết cho truy vấn bên ngoài.

|  |  |  |
| --- | --- | --- |
| **Đặc điểm** | **CTE** | **Subquery** |
| **Độ phức tạp** | Tốt hơn cho truy vấn phức tạp. | Tốt hơn cho truy vấn đơn giản. |
| **Khả năng đọc** | Rất cao, chia nhỏ logic. | Có thể khó đọc khi lồng nhiều lớp. |
| **Tái sử dụng** | Có thể tái sử dụng nhiều lần trong cùng một câu lệnh. | Chỉ dùng một lần. |
| **Truy vấn đệ quy** | Hỗ trợ. | Không hỗ trợ. |
| **Hiệu suất** | Tùy thuộc vào trình tối ưu hóa truy vấn của database, thường tương đương hoặc tốt hơn subquery lồng. | Tùy thuộc vào trình tối ưu hóa. Subquery lồng sâu có thể kém hiệu quả. |

+ Ranking : ROW\_NUMBER, RANK, DENSE\_RANK

**ROW\_NUMBER:**

Là việc gán một số thứ tự duy nhất và liên tiếp cho mỗi hàng trong tập hợp kết quả, bắt đầu từ 1.

**Cách hoạt động:** Nó không quan tâm đến các giá trị trùng lặp. Mỗi hàng, dù có giá trị giống nhau, vẫn nhận một số thứ tự riêng.

**SELECT** deptno,

ename,

hiredate,

sal,

**ROW\_NUMBER** () **OVER** (**PARTITION** **BY** deptno **ORDER** **BY** hiredate) *num*

**FROM** emp

**ORDER** **BY** deptno, *num*;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DEPTNO | ENAME | HIREDATE | SAL | NUM |
| 10 | CLARK | 09/06/1981 | 2450 | 1 |
| 10 | KING | 17/11/1981 | 5000 | 2 |
| 10 | MILLER | 23/01/1982 | 1300 | 3 |
| 20 | SMITH | 17/12/1980 | 800 | 1 |
| 20 | JONES | 02/04/1981 | 2975 | 2 |
| 20 | FORD | 03/12/1981 | 3000 | 3 |
| 20 | SCOTT | 09/12/1982 | 3000 | 4 |
| 20 | ADAMS | 12/01/1983 | 1100 | 5 |
| 30 | ALLEN | 20/02/1981 | 1600 | 1 |
| 30 | WARD | 22/02/1981 | 1250 | 2 |
| 30 | BLAKE | 01/05/1981 | 2850 | 3 |
| 30 | TURNER | 08/09/1981 | 1500 | 4 |
| 30 | MARTIN | 28/09/1981 | 1250 | 5 |
| 30 | JAMES | 03/12/1981 | 950 | 6 |

**RANK, DENSE\_RANK:**

**RANK**: Gán một số thứ tự cho mỗi hàng. Nếu có các hàng có cùng giá trị, chúng sẽ nhận cùng một thứ hạng. Thứ hạng tiếp theo sẽ bị bỏ qua (có một "khoảng trống").

**Cách hoạt động:** Nếu có 2 hàng cùng hạng 2, thứ hạng tiếp theo sẽ là 4. **Sử dụng khi** cần xếp hạng các hàng và muốn các hàng có giá trị bằng nhau có cùng thứ hạng.

**DENSE\_RANK:** Tương tự như RANK(), nhưng không bỏ qua thứ hạng tiếp theo khi có các giá trị trùng lặp.

**Cách hoạt động:** Nếu có 2 hàng cùng hạng 2, thứ hạng tiếp theo sẽ là 3 (không có "khoảng trống").

**Sử dụng khi:** cần xếp hạng các hàng và muốn các hàng có giá trị bằng nhau có cùng thứ hạng, nhưng không muốn có khoảng trống trong chuỗi thứ hạng.

**SELECT** deptno,

ename,

hiredate,

sal,

**RANK** () **OVER** (**PARTITION** **BY** deptno **ORDER** **BY** sal) *rank\_sal*,

**DENSE\_RANK** () **OVER** (**PARTITION** **BY** deptno **ORDER** **BY** sal) *dense\_rank\_sal*

**FROM** emp

**ORDER** **BY** deptno;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DEPTNO | ENAME | HIREDATE | SAL | RANK\_SAL | DENSE\_RANK\_SAL |
| 10 | MILLER | 1/23/1982 | 1300 | 1 | 1 |
| 10 | CLARK | 6/9/1981 | 2450 | 2 | 2 |
| 10 | KING | 11/17/1981 | 5000 | 3 | 3 |
| 20 | SMITH | 12/17/1980 | 800 | 1 | 1 |
| 20 | ADAMS | 1/12/1983 | 1100 | 2 | 2 |
| 20 | JONES | 4/2/1981 | 2975 | 3 | 3 |
| 20 | SCOTT | 12/9/1982 | 3000 | 4 | 4 |
| 20 | FORD | 12/3/1981 | 3000 | 4 | 4 |
| 30 | JAMES | 12/3/1981 | 950 | 1 | 1 |
| 30 | MARTIN | 9/28/1981 | 1250 | 2 | 2 |
| 30 | WARD | 2/22/1981 | 1250 | 2 | 2 |
| 30 | TURNER | 9/8/1981 | 1500 | 4 | 3 |
| 30 | ALLEN | 2/20/1981 | 1600 | 5 | 4 |
| 30 | BLAKE | 5/1/1981 | 2850 | 6 | 5 |

Hàm **RANK** và **DENSE\_RANK** có những điểm như sau:

* Giống nhau: với những dòng có cùng SAL thì sẽ có cùng hạng
* Khác nhau:

+ Hàm RANK sẽ bỏ qua thứ hạng tiếp theo cho những dòng có cùng hạng trước đó  
+ Hàm DENSE\_RANK không bỏ qua thứ hạng tiếp theo cho những dòng có cùng hạng trước đó

+ VIEW : Có thể CRUD trên VIEW được không ?

**View** là một **bảng ảo** trong cơ sở dữ liệu. Nó được tạo ra từ kết quả của một câu lệnh SELECT. Dữ liệu của view không được lưu trữ vật lý; thay vào đó, mỗi khi bạn truy vấn view, cơ sở dữ liệu sẽ thực thi câu lệnh SELECT đã định nghĩa để tạo ra tập hợp kết quả.

**Create (INSERT), Update, Delete:** **Có thể**, nhưng chỉ trong các trường hợp đơn giản.

* View chỉ dựa trên **một bảng duy nhất**.
* View không chứa các hàm tổng hợp như SUM(), AVG(), COUNT().
* View không chứa các mệnh đề như GROUP BY, HAVING, DISTINCT.
* View không chứa các JOIN phức tạp.

**create** **view** bao\_cao\_phong\_ban **as**

**select** *d*.dept\_name , *e*.emp\_name , *e*.hire\_date , *e*.salary

**from** departments *d* **left** **join** employees *e*

**on** *d*.dept\_id = *e*.dept\_id

**order** **by** *d*.dept\_id

**select** \* **from** bao\_cao\_phong\_ban

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

**PROCEDURE** (thủ tục) là một tập hợp các câu lệnh SQL đã được biên dịch và lưu trữ trên cơ sở dữ liệu.

Mục đích chính của việc tạo ra một PROCEDURE là:

* **Tái sử dụng mã:** Thay vì viết đi viết lại một khối lệnh SQL phức tạp, bạn có thể lưu nó dưới dạng một PROCEDURE và gọi lại khi cần.
* **Tăng hiệu suất:** Các thủ tục được biên dịch trước, giúp giảm thời gian phân tích và tối ưu hóa truy vấn khi thực thi.
* **Bảo mật:** Bạn có thể cấp quyền truy cập vào thủ tục mà không cần cấp quyền truy cập trực tiếp vào các bảng cơ sở, giúp bảo vệ dữ liệu nhạy cảm.
* **Tính toàn vẹn dữ liệu:** PROCEDURE cho phép thực hiện nhiều thao tác CSDL trong một khối lệnh duy nhất, đảm bảo tính toàn vẹn của dữ liệu bằng cách sử dụng giao dịch (transaction).