TOÁN TỬ

▼ TOÁN TỬ ALL

- Toán tử `ALL` trong SQL được sử dụng để so sánh một giá trị với **TẤT CẢ CÁC GIÁ TRỊ** trong một tập hợp con, tập hợp con này là kết quả từ một truy vấn con (subquery).
- Khi sử dụng toán tử `ALL`, điều kiện so sánh phải **ĐÚNG VỚI TẤT CẢ CÁC GIÁ TRỊ** trong tập hợp con để điều kiện tổng thể được coi là đúng.

Dưới đây là cú pháp tổng quát của toán tử `ALL`:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL (subquery);
```

Trong đó:

- `operator` là toán tử so sánh như `=`, `!=`, `>`, `<`, `>=`, hoặc `<=`.
- `subquery` là một truy vấn con trả về một tập hợp các giá trị.

Ví dụ minh họa

Giả sử chúng ta có hai bảng `Products` và `Orders`:

```
Products
+----+--------------+-------+
| ID | ProductName  | Price |
+----+--------------+-------+
| 1  | Product A    | 100   |
| 2  | Product B    | 200   |
| 3  | Product C    | 150   |
+----+--------------+-------+

Orders
+----+---------+----------+
    | ID | Product | Quantity |
+----+---------+----------+
```

```
| 1  | Product A | 10     |
| 2  | Product B | 5      |
| 3  | Product C | 7      |
+----+---------+----------+
```

Giả sử chúng ta muốn tìm tất cả các sản phẩm có giá lớn hơn giá của tất cả các sản phẩm trong bảng `Products` mà có tên là "Product A". Truy vấn SQL sẽ như sau:

```
SELECT ProductName
FROM Products
WHERE Price > ALL (SELECT Price FROM Products WHERE Prod
uctName = 'Product A');
```

Trong ví dụ trên:

- Truy vấn con `(SELECT Price FROM Products WHERE ProductName = 'Product A')` sẽ trả về tập hợp chứa giá của "Product A", tức là `[100]`.
- Điều kiện `Price > ALL (SELECT Price FROM Products WHERE ProductName = 'Product A')` tương đương với `Price > 100`.

Kết quả truy vấn sẽ trả về các sản phẩm có giá lớn hơn 100.

```
+--------------+
| ProductName  |
+--------------+
| Product B    |
| Product C    |
+--------------+
```

Điều này xảy ra vì:

- Giá của "Product A" là 100.
- Truy vấn con `(SELECT Price FROM Products WHERE ProductName = 'Product A')` sẽ trả về `[100]`.
- Điều kiện `Price > ALL (SELECT Price FROM Products WHERE ProductName = 'Product A')` trở thành `Price > 100`.

Do đó, sản phẩm nào có giá lớn hơn 100 sẽ được trả về:

- "Product B" có giá 200 (thỏa mãn điều kiện `Price > 100`).
- "Product C" có giá 150 (thỏa mãn điều kiện `Price > 100`).

Một số điểm cần lưu ý:

- Nếu truy vấn con không trả về bất kỳ hàng nào, điều kiện `ALL` sẽ luôn đúng.
- Toán tử `ALL` thường được sử dụng trong các tình huống cần kiểm tra điều kiện so sánh trên toàn bộ tập hợp giá trị trả về từ truy vấn con

Toán tử `ALL` có thể được kết hợp với các toán tử so sánh khác nhau để thực hiện các điều kiện so sánh phức tạp trên các tập hợp con trong SQL.

▼ TOÁN TỬ ANY

- Toán tử `ANY` trong SQL được sử dụng để so sánh một giá trị với **BẤT KỲ GIÁ TRỊ** nào trong một tập hợp con, tập hợp con này là kết quả từ một truy vấn con (subquery).
- Khi sử dụng toán tử `ANY`, điều kiện so sánh **CHỈ CẦN ĐÚNG VỚI ÍT NHẤT MỘT GIÁ TRỊ** trong tập hợp con để điều kiện tổng thể được coi là đúng.

Dưới đây là cú pháp tổng quát của toán tử `ANY`:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY (subquery);
```

Trong đó:

- `operator` là toán tử so sánh như `=`, `!=`, `>`, `<`, `>=`, hoặc `<=`.
- `subquery` là một truy vấn con trả về một tập hợp các giá trị.

Ví dụ minh họa

Giả sử chúng ta có hai bảng `Products` và `Orders` giống như trong ví dụ trước:

```
Products
+----+--------------+-------+
| ID | ProductName  | Price |
+----+--------------+-------+
```

```
| 1  | Product A    | 100   |
| 2  | Product B    | 200   |
| 3  | Product C    | 150   |
+----+--------------+-------+

Orders
+----+---------+----------+
| ID | Product | Quantity |
+----+---------+----------+
| 1  | Product A | 10     |
| 2  | Product B | 5      |
| 3  | Product C | 7      |
+----+---------+----------+
```

Giả sử chúng ta muốn tìm tất cả các sản phẩm có giá lớn hơn ít nhất một trong các giá trị giá của sản phẩm trong bảng `Products` mà có tên là "Product A". Truy vấn SQL sẽ như sau:

```
SELECT ProductName
FROM Products
WHERE Price > ANY (SELECT Price FROM Products WHERE Prod
uctName = 'Product A');
```

Trong ví dụ trên:

- Truy vấn con `(SELECT Price FROM Products WHERE ProductName = 'Product A')` sẽ trả về tập hợp chứa giá của "Product A", tức là `[100]`.
- Điều kiện `Price > ANY (SELECT Price FROM Products WHERE ProductName = 'Product A')` tương đương với `Price > 100`.

Kết quả truy vấn sẽ trả về các sản phẩm có giá lớn hơn 100.

Một số điểm cần lưu ý:

- Nếu truy vấn con không trả về bất kỳ hàng nào, điều kiện `ANY` sẽ luôn sai.
- Toán tử `ANY` thường được sử dụng trong các tình huống cần kiểm tra điều kiện so sánh với ít nhất một giá trị trong tập hợp giá trị trả về từ truy vấn con.

Kết quả của ví dụ

Kết quả của truy vấn trên sẽ là:

```
+--------------+
| ProductName  |
+--------------+
| Product B    |
| Product C    |
+--------------+
```

Điều này xảy ra vì:

- Giá của "Product A" là 100.
- Truy vấn con `(SELECT Price FROM Products WHERE ProductName = 'Product A')` sẽ trả về `[100]`.
- Điều kiện `Price > ANY (SELECT Price FROM Products WHERE ProductName = 'Product A')` trở thành `Price > 100`.

Do đó, sản phẩm nào có giá lớn hơn 100 sẽ được trả về:

- "Product B" có giá 200 (thỏa mãn điều kiện `Price > 100`).
- "Product C" có giá 150 (thỏa mãn điều kiện `Price > 100`).

▼ SO SÁNH TOÁN TỬ ALL VÀ ANY

Toán tử `ALL` và `ANY` trong SQL đều được sử dụng để so sánh một giá trị với một tập hợp con các giá trị từ một truy vấn phụ (subquery). Tuy nhiên, chúng có sự khác biệt quan trọng trong cách mà các điều kiện so sánh được thực hiện. Dưới đây là sự so sánh giữa hai toán tử này:

Toán tử `ALL`

- **Mục đích**: Kiểm tra xem điều kiện có đúng với tất cả các giá trị trong tập hợp kết quả của truy vấn con hay không.
- **Cách hoạt động**: Điều kiện so sánh phải đúng với tất cả các giá trị trả về từ truy vấn con. Nếu có ít nhất một giá trị không thỏa mãn điều kiện, thì toàn bộ điều kiện sẽ là sai.
- **Ví dụ**:Điều kiện `Price > ALL (SELECT Price FROM Products WHERE ProductName = 'Product A')` sẽ chỉ đúng nếu `Price` lớn hơn tất cả các giá trị trả về từ truy vấn con

```
SELECT ProductName
FROM Products
WHERE Price > ALL (SELECT Price FROM Products WHERE P
roductName = 'Product A');
```

Toán tử `ANY`

- **Mục đích**: Kiểm tra xem điều kiện có đúng với ít nhất một giá trị trong tập hợp kết quả của truy vấn phụ hay không.
- **Cách hoạt động**: Điều kiện so sánh chỉ cần đúng với ít nhất một giá trị trả về từ truy vấn phụ. Nếu có ít nhất một giá trị thỏa mãn điều kiện, thì toàn bộ điều kiện sẽ là đúng.
- **Ví dụ**:Điều kiện `Price > ANY (SELECT Price FROM Products WHERE ProductName = 'Product A')` sẽ đúng nếu `Price` lớn hơn ít nhất một giá trị trả về từ truy vấn con.

```
SELECT ProductName
FROM Products
WHERE Price > ANY (SELECT Price FROM Products WHERE P
roductName = 'Product A');
```

So sánh chi tiết

1. **Phạm vi kiểm tra**:
   - `ALL`: Kiểm tra điều kiện so sánh với tất cả các giá trị trong tập hợp.
   - `ANY`: Kiểm tra điều kiện so sánh với ít nhất một giá trị trong tập hợp.
2. **Kết quả điều kiện**:
   - `ALL`: Trả về `true` chỉ khi điều kiện đúng với mọi giá trị trong tập hợp.
   - `ANY`: Trả về `true` nếu điều kiện đúng với ít nhất một giá trị trong tập hợp.
3. **Các ví dụ cụ thể**:
   - Với `ALL`: Giả sử tập hợp giá trị `[100, 200, 300]`, điều kiện `Price > ALL ([100, 200, 300])` chỉ đúng nếu `Price` lớn hơn 300.

- Với `ANY`: Giả sử tập hợp giá trị `[100, 200, 300]`, điều kiện `Price > ANY ([100, 200, 300])` đúng nếu `Price` lớn hơn 100, hoặc 200, hoặc 300.

Kết luận

Toán tử `ALL` và `ANY` đều mạnh mẽ trong việc so sánh với tập hợp giá trị, nhưng chúng phục vụ các mục đích khác nhau:

- Sử dụng `ALL` khi bạn cần đảm bảo rằng điều kiện đúng với tất cả các giá trị trong tập hợp.
- Sử dụng `ANY` khi bạn cần điều kiện đúng chỉ với một giá trị duy nhất trong tập hợp

MỘT VÍ DỤ KHÁC

Bảng Employees:

employeeid	firstname	lastname	departmentid	salary
1	John	Doe	101	50000.00
2	Jane	Smith	102	60000.00
3	Mike	Johnson	103	55000.00
4	Emily	Davis	101	70000.00
5	David	Wilson	102	40000.00

Bảng Departments:

departmentid	departmentname
101	HR
102	Engineering
103	Marketing

1.1. Tìm các nhân viên có lương lớn hơn BẤT KỲ mức lương nào trong bộ phận 'Engineering'

```
SELECT * FROM Employees
WHERE Salary > ANY (SELECT salary from Employees WHERE dep
```

Kết quả:

employeeid	firstname	lastname	departmentid	salary
1	John	Doe	101	50000.00
2	Jane	Smith	102	60000.00
3	Mike	Johnson	103	55000.00
4	Emily	Davis	101	70000.00

1.2. Tìm các nhân viên có lương lớn hơn TẤT CẢ mức lương nào trong bộ phận 'Engineering'

```
SELECT * FROM Employees
WHERE Salary > ALL (SELECT salary from Employees WHERE dep
```

Kết quả:

employeeid	firstname	lastname	departmentid	salary
4	Emily	Davis	101	70000.00

2.1. Tìm các nhân viên có lương ít hơn BẤT KỲ mức lương nào trong bộ phận 'HR'

Kết quả:

```
SELECT * FROM Employees
WHERE Salary < ANY (SELECT salary from Employees WHERE dep
```

Kết quả:

employeeid	firstname	lastname	departmentid	salary
1	John	Doe	101	50000.00
2	Jane	Smith	102	60000.00
3	Mike	Johnson	103	55000.00
5	David	Wilson	102	40000.00

2.1. Tìm các nhân viên có lương ít hơn TẤT CẢ mức lương nào trong bộ phận 'HR'

```
SELECT * FROM Employees
WHERE Salary < ALL (SELECT salary from Employees WHERE dep
```

employeeid	firstname	lastname	departmentid	salary
5	David	Wilson	102	40000.00

▼ TOÁN TỬ BETWEEN

- Toán tử `BETWEEN` trong SQL được sử dụng để chọn các giá trị trong một phạm vi nhất định.
- Toán tử này có thể được áp dụng cho các giá trị số, ngày tháng và thậm chí các chuỗi văn bản. Giá trị bắt đầu và kết thúc của phạm vi được bao gồm trong kết quả.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Trong đó:

- `column_name(s)` là tên của các cột mà bạn muốn chọn.
- `table_name` là tên của bảng chứa dữ liệu.
- `column_name` là cột mà bạn muốn kiểm tra giá trị.
- `value1` và `value2` xác định phạm vi giá trị để chọn.

Ví dụ minh họa

Giả sử chúng ta có một bảng `Products` như sau:

```
Products
+----+--------------+-------+
| ID | ProductName  | Price |
+----+--------------+-------+
| 1  | Product A    | 100   |
| 2  | Product B    | 200   |
| 3  | Product C    | 150   |
```

```
| 4  | Product D    | 250   |
| 5  | Product E    | 300   |
+----+--------------+-------+
```

Ví dụ 1: Sử dụng `BETWEEN` với giá trị số

Giả sử chúng ta muốn tìm các sản phẩm có giá nằm trong khoảng từ 150 đến 300. Truy vấn SQL sẽ như sau:

```
SELECT ProductName, Price
FROM Products
WHERE Price BETWEEN 150 AND 300;
```

Kết quả sẽ là:

```
+--------------+-------+
| ProductName  | Price |
+--------------+-------+
| Product C    | 150   |
| Product D    | 250   |
| Product E    | 300   |
+--------------+-------+
```

Ví dụ 2: Sử dụng `BETWEEN` với ngày tháng

Giả sử chúng ta có một bảng `Orders` như sau:

```
Orders
+----+------------+------------+
| ID | OrderDate  | CustomerID |
+----+------------+------------+
| 1  | 2024-01-10 | 1          |
| 2  | 2024-02-15 | 2          |
| 3  | 2024-03-20 | 3          |
| 4  | 2024-04-25 | 4          |
| 5  | 2024-05-30 | 5          |
+----+------------+------------+
```

Giả sử chúng ta muốn tìm các đơn hàng được đặt trong khoảng thời gian từ ngày 2024-02-01 đến ngày 2024-04-30. Truy vấn SQL sẽ như sau:

```
SELECT ID, OrderDate, CustomerID
FROM Orders
WHERE OrderDate BETWEEN '2024-02-01' AND '2024-04-30';
```

Kết quả sẽ là:

```
+----+------------+------------+
| ID | OrderDate  | CustomerID |
+----+------------+------------+
| 2  | 2024-02-15 | 2          |
| 3  | 2024-03-20 | 3          |
| 4  | 2024-04-25 | 4          |
+----+------------+------------+
```

Một số điểm cần lưu ý:

1. **Bao gồm giá trị biên**: `BETWEEN` bao gồm cả giá trị bắt đầu và kết thúc trong phạm vi. Ví dụ, `BETWEEN 150 AND 300` bao gồm cả 150 và 300.
2. **Tính tương đương**: Sử dụng `BETWEEN value1 AND value2` tương đương với sử dụng `>= value1 AND <= value2`.
3. **Đối với chuỗi văn bản**: `BETWEEN` cũng có thể được sử dụng với các chuỗi văn bản, nhưng kết quả phụ thuộc vào thứ tự sắp xếp của các chuỗi đó.

Ví dụ:

```
SELECT column_name
FROM table_name
WHERE column_name BETWEEN 'A' AND 'C';
```

Truy vấn này sẽ chọn tất cả các giá trị trong `column_name` nằm trong khoảng từ 'A' đến 'C' theo thứ tự sắp xếp.

Kết luận

Toán tử `BETWEEN` rất hữu ích khi bạn muốn chọn các giá trị nằm trong một phạm vi nhất định và có thể được áp dụng cho các loại dữ liệu khác nhau

trong SQL

▼ TOÁN TỬ EXITST

- Toán tử `EXISTS` trong SQL được sử dụng để kiểm tra sự tồn tại của các hàng trong kết quả của một truy vấn con (subquery).
- Nó trả về `TRUE` nếu truy vấn con trả về ít nhất một hàng và `FALSE` nếu không có hàng nào được trả về.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS (subquery);
```

Trong đó:

- `column_name(s)` là tên của các cột mà bạn muốn chọn.
- `table_name` là tên của bảng chứa dữ liệu.
- `subquery` là một truy vấn con, kết quả của nó sẽ được kiểm tra bởi `EXISTS`.

Ví dụ minh họa

Giả sử chúng ta có hai bảng `Products` và `Orders`:

```
Products
+----+--------------+-------+
| ID | ProductName  | Price |
+----+--------------+-------+
| 1  | Product A    | 100   |
| 2  | Product B    | 200   |
| 3  | Product C    | 150   |
+----+--------------+-------+

Orders
+----+---------+----------+
| ID | Product | Quantity |
+----+---------+----------+
| 1  | Product A | 10     |
```

```
| 2  | Product B | 5       |
| 3  | Product C | 7       |
| 4  | Product A | 3       |
+----+---------+----------+
```

Ví dụ 1: Sử dụng `EXISTS` để kiểm tra sự tồn tại của sản phẩm trong đơn hàng

Giả sử chúng ta muốn tìm tất cả các sản phẩm có ít nhất một đơn hàng. Truy vấn SQL sẽ như sau:

```
SELECT ProductName
FROM Products p
WHERE EXISTS (SELECT 1 FROM Orders o WHERE o.Product =
p.ProductName);
```

Trong ví dụ trên:

- Truy vấn con `(SELECT 1 FROM Orders o WHERE o.Product = p.ProductName)` kiểm tra xem có bất kỳ hàng nào trong bảng `Orders` mà `Product` khớp với `ProductName` của bảng `Products` hay không.
- Nếu truy vấn con trả về ít nhất một hàng, điều kiện `EXISTS` sẽ là `TRUE` và sản phẩm đó sẽ được bao gồm trong kết quả.

Kết quả sẽ là:

```
+--------------+
| ProductName  |
+--------------+
| Product A    |
| Product B    |
| Product C    |
+--------------+
```

Ví dụ 2: Sử dụng `EXISTS` để tìm khách hàng có đơn hàng trong một khoảng thời gian nhất định

Giả sử chúng ta có bảng `Customers` và `Orders` như sau:

```
Customers
+----+------------+
| ID | CustomerID |
+----+------------+
| 1  | Cust1      |
| 2  | Cust2      |
| 3  | Cust3      |
+----+------------+

Orders
+----+------------+------------+
| ID | CustomerID | OrderDate  |
+----+------------+------------+
| 1  | Cust1      | 2024-01-10 |
| 2  | Cust2      | 2024-02-15 |
| 3  | Cust1      | 2024-03-20 |
| 4  | Cust3      | 2024-04-25 |
+----+------------+------------+
```

Giả sử chúng ta muốn tìm tất cả các khách hàng có ít nhất một đơn hàng được đặt trong khoảng thời gian từ ngày 2024-02-01 đến ngày 2024-04-30. Truy vấn SQL sẽ như sau:

```
SELECT CustomerID
FROM Customers c
WHERE EXISTS (
    SELECT 1
    FROM Orders o
    WHERE o.CustomerID = c.CustomerID
    AND o.OrderDate BETWEEN '2024-02-01' AND '2024-04-3
0'
);
```

Kết quả sẽ là:

```
+------------+
| CustomerID |
+------------+
```

```
| Cust1      |
| Cust2      |
| Cust3      |
+------------+
```

Một số điểm cần lưu ý:

1. **Hiệu suất**: `EXISTS` có thể hiệu quả hơn so với các toán tử khác (như `IN`) trong một số trường hợp, đặc biệt khi truy vấn con chứa các phép nối phức tạp hoặc khi bảng con lớn.
2. **Không quan tâm giá trị trả về**: Truy vấn con trong `EXISTS` chỉ kiểm tra sự tồn tại của các hàng mà không quan tâm đến giá trị trả về. Vì vậy, thông thường chúng ta sử dụng `SELECT 1` hoặc `SELECT *` trong truy vấn con.
3. **Kết hợp với các toán tử khác**: `EXISTS` có thể được kết hợp với các toán tử khác như `NOT EXISTS` để kiểm tra sự không tồn tại của các hàng.

Kết luận

Toán tử `EXISTS` rất hữu ích khi bạn cần kiểm tra sự tồn tại của các hàng trong kết quả của một truy vấn con. Nó có thể được áp dụng trong nhiều tình huống khác nhau để kiểm tra mối quan hệ giữa các bảng trong cơ sở dữ liệu.

▼ TOÁN TỬ IN

- Toán tử `IN` trong SQL được sử dụng để xác định xem một giá trị có nằm trong một tập hợp các giá trị cụ thể hay không.
- Nó thường được sử dụng trong mệnh đề `WHERE` để lọc dữ liệu dựa trên danh sách các giá trị.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Trong đó:

- `column_name(s)` là tên của các cột mà bạn muốn chọn.
- `table_name` là tên của bảng chứa dữ liệu.

- `column_name` là cột mà bạn muốn kiểm tra giá trị.
- `value1, value2, ...` là danh sách các giá trị để kiểm tra.

Ví dụ minh họa

Giả sử chúng ta có một bảng `Products` như sau:

```
Products
+----+--------------+-------+
| ID | ProductName  | Price |
+----+--------------+-------+
| 1  | Product A    | 100   |
| 2  | Product B    | 200   |
| 3  | Product C    | 150   |
| 4  | Product D    | 250   |
| 5  | Product E    | 300   |
+----+--------------+-------+
```

Ví dụ 1: Sử dụng `IN` với danh sách giá trị

Giả sử chúng ta muốn tìm các sản phẩm có giá là 150, 200, hoặc 300. Truy vấn SQL sẽ như sau:

```
SELECT ProductName, Price
FROM Products
WHERE Price IN (150, 200, 300);
```

Kết quả sẽ là:

```
+--------------+-------+
| ProductName  | Price |
+--------------+-------+
| Product B    | 200   |
| Product C    | 150   |
| Product E    | 300   |
+--------------+-------+
```

Ví dụ 2: Sử dụng `IN` với truy vấn con

Giả sử chúng ta có bảng `Orders` như sau:

```
Orders
+----+---------+----------+
| ID | Product | Quantity |
+----+---------+----------+
| 1  | Product A | 10     |
| 2  | Product B | 5      |
| 3  | Product C | 7      |
| 4  | Product D | 3      |
+----+---------+----------+
```

Giả sử chúng ta muốn tìm các sản phẩm có đơn hàng. Truy vấn SQL sẽ như sau:

```
SELECT ProductName
FROM Products
WHERE ProductName IN (SELECT Product FROM Orders);
```

Kết quả sẽ là:

```
+--------------+
| ProductName  |
+--------------+
| Product A    |
| Product B    |
| Product C    |
| Product D    |
+--------------+
```

So sánh với toán tử khác

`IN` vs `=`:

- `IN` được sử dụng khi bạn muốn so sánh một giá trị với nhiều giá trị khác.
- `=` được sử dụng khi bạn chỉ cần so sánh với một giá trị duy nhất.

Ví dụ:

```
-- Sử dụng IN
SELECT ProductName
FROM Products
WHERE Price IN (150, 200, 300);

-- Sử dụng =
SELECT ProductName
FROM Products
WHERE Price = 150 OR Price = 200 OR Price = 300;
```

`IN` vs `EXISTS`:

- `IN` thường dễ đọc và ngắn gọn hơn khi làm việc với danh sách các giá trị.
- `EXISTS` thường hiệu quả hơn với các truy vấn con phức tạp hoặc các bảng lớn.

Một số điểm cần lưu ý:

1. **Danh sách giá trị**: Danh sách giá trị trong `IN` có thể là các giá trị cụ thể hoặc kết quả của một truy vấn con.
2. **Hiệu suất**: Với các danh sách giá trị lớn, `IN` có thể không hiệu quả. Trong các tình huống như vậy, sử dụng các phương pháp tối ưu hóa khác hoặc chỉ mục có thể cải thiện hiệu suất.
3. **Kiểu dữ liệu**: Các giá trị trong danh sách phải có cùng kiểu dữ liệu với cột mà bạn đang so sánh.

Kết luận

Toán tử `IN` rất hữu ích khi bạn cần kiểm tra một giá trị có nằm trong một tập hợp các giá trị cụ thể hay không. Nó làm cho truy vấn dễ đọc hơn và thường được sử dụng trong các tình huống yêu cầu so sánh với nhiều giá trị.

▼ TOÁN TỬ LIKE

Toán tử `LIKE` trong SQL được sử dụng để tìm kiếm các giá trị trong một cột khớp với một mẫu cụ thể. Nó thường được sử dụng trong mệnh đề `WHERE` để lọc dữ liệu dựa trên các mẫu chuỗi.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

Trong đó:

- `column_name(s)` là tên của các cột mà bạn muốn chọn.
- `table_name` là tên của bảng chứa dữ liệu.
- `column_name` là cột mà bạn muốn kiểm tra giá trị.
- `pattern` là mẫu mà bạn muốn so khớp.

Ký tự đại diện trong mẫu

- `%`: Đại diện cho bất kỳ chuỗi ký tự nào, bao gồm cả chuỗi rỗng.
- `_`: Đại diện cho một ký tự đơn.

Ví dụ minh họa

Giả sử chúng ta có một bảng `Customers` như sau:

```
Customers
+----+---------+---------------+
| ID | Name    | City          |
+----+---------+---------------+
| 1  | Alice   | New York      |
| 2  | Bob     | Los Angeles   |
| 3  | Charlie | New Orleans   |
| 4  | David   | Chicago       |
| 5  | Eve     | San Francisco |
+----+---------+---------------+
```

Ví dụ 1: Sử dụng `%` để tìm chuỗi bất kỳ

Giả sử chúng ta muốn tìm tất cả các khách hàng có tên bắt đầu bằng chữ "A". Truy vấn SQL sẽ như sau:

```
SELECT Name, City
FROM Customers
```

```
WHERE Name LIKE 'A%';
```

Kết quả sẽ là:

```
+-------+----------+
| Name  | City     |
+-------+----------+
| Alice | New York |
+-------+----------+
```

Ví dụ 2: Sử dụng `_` để tìm một ký tự cụ thể

Giả sử chúng ta muốn tìm tất cả các khách hàng có tên dài 3 ký tự. Truy vấn SQL sẽ như sau:

```
SELECT Name, City
FROM Customers
WHERE Name LIKE '___';
```

Kết quả sẽ là:

```
+------+---------------+
| Name | City          |
+------+---------------+
| Bob  | Los Angeles   |
| Eve  | San Francisco |
+------+---------------+
```

Ví dụ 3: Kết hợp `%` và `_` để tìm mẫu phức tạp

Giả sử chúng ta muốn tìm tất cả các khách hàng sống ở thành phố có từ "New" trong tên. Truy vấn SQL sẽ như sau:

```
SELECT Name, City
FROM Customers
WHERE City LIKE '%New%';
```

Kết quả sẽ là:

```
+---------+-------------+
| Name    | City        |
+---------+-------------+
| Alice   | New York    |
| Charlie | New Orleans |
+---------+-------------+
```

Sử dụng `NOT LIKE`

Toán tử `NOT LIKE` được sử dụng để tìm các giá trị không khớp với mẫu cụ thể. Ví dụ:

```
SELECT Name, City
FROM Customers
WHERE City NOT LIKE '%New%';
```

Kết quả sẽ là:

```
+---------+---------------+
| Name    | City          |
+---------+---------------+
| Bob     | Los Angeles   |
| David   | Chicago       |
| Eve     | San Francisco |
+---------+---------------+
```

So sánh `LIKE` với `=`

- `LIKE` được sử dụng để so khớp với các mẫu chuỗi, bao gồm các ký tự đại diện.
- `=` được sử dụng để so sánh giá trị chính xác.

Ví dụ:

```
-- Sử dụng LIKE
SELECT Name, City
FROM Customers
WHERE City LIKE 'New%';
```

```
-- Sử dụng =
SELECT Name, City
FROM Customers
WHERE City = 'New York';
```

Một số điểm cần lưu ý:

1. **Phân biệt chữ hoa chữ thường**: Tùy thuộc vào hệ quản trị cơ sở dữ liệu (DBMS), toán tử `LIKE` có thể hoặc không phân biệt chữ hoa chữ thường. Trong MySQL, `LIKE` không phân biệt chữ hoa chữ thường, nhưng bạn có thể sử dụng `BINARY` để phân biệt chữ hoa chữ thường.
2. **Hiệu suất**: Sử dụng `LIKE` với các mẫu bắt đầu bằng `%` có thể ảnh hưởng đến hiệu suất vì nó yêu cầu quét toàn bộ bảng.
3. **ESCAPE**: Khi bạn cần tìm các ký tự `%` hoặc `_`, bạn có thể sử dụng mệnh đề `ESCAPE` để định nghĩa một ký tự thoát.

Ví dụ:

```
SELECT Name, City
FROM Customers
WHERE City LIKE '100\%' ESCAPE '\';
```

Kết luận

Toán tử `LIKE` rất hữu ích khi bạn cần tìm kiếm các giá trị trong một cột khớp với một mẫu cụ thể. Nó cung cấp sự linh hoạt thông qua việc sử dụng các ký tự đại diện và có thể được kết hợp với các toán tử khác để xây dựng các truy vấn mạnh mẽ.

▼ TOÁN TỬ IS NULL

- Toán tử `IS NULL` trong SQL được sử dụng để kiểm tra xem một giá trị có phải là `NULL` hay không.
- Giá trị `NULL` trong SQL đại diện cho dữ liệu thiếu hoặc không xác định.
- Toán tử `IS NULL` và `IS NOT NULL` được sử dụng để lọc dữ liệu dựa trên sự hiện diện hay vắng mặt của các giá trị `NULL`.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL;
```

hoặc

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NOT NULL;
```

Ví dụ minh họa

Giả sử chúng ta có một bảng `Employees` như sau:

```
Employees
+----+--------+-----------+--------+
| ID | Name   | Department| Salary |
+----+--------+-----------+--------+
| 1  | Alice  | HR        | 5000   |
| 2  | Bob    | IT        | NULL   |
| 3  | Charlie| NULL      | 7000   |
| 4  | David  | IT        | 6000   |
| 5  | Eve    | HR        | NULL   |
+----+--------+-----------+--------+
```

Ví dụ 1: Sử dụng `IS NULL`

Giả sử chúng ta muốn tìm tất cả các nhân viên có mức lương chưa xác định (tức là `NULL`). Truy vấn SQL sẽ như sau:

```
SELECT Name, Salary
FROM Employees
WHERE Salary IS NULL;
```

Kết quả sẽ là:

```
+-------+--------+
| Name  | Salary |
```

```
+-------+--------+
| Bob   | NULL   |
| Eve   | NULL   |
+-------+--------+
```

Ví dụ 2: Sử dụng `IS NOT NULL`

Giả sử chúng ta muốn tìm tất cả các nhân viên có phòng ban được chỉ định (tức là không phải `NULL`). Truy vấn SQL sẽ như sau:

```
SELECT Name, Department
FROM Employees
WHERE Department IS NOT NULL;
```

Kết quả sẽ là:

```
+--------+-----------+
| Name   | Department|
+--------+-----------+
| Alice  | HR        |
| Bob    | IT        |
| David  | IT        |
+--------+-----------+
```

Sử dụng `IS NULL` với các phép nối

Giả sử chúng ta có bảng `Departments` như sau:

```
Departments
+----+------------+
| ID | Department |
+----+------------+
| 1  | HR         |
| 2  | IT         |
+----+------------+
```

Và chúng ta muốn tìm tất cả các nhân viên và phòng ban của họ, bao gồm cả những nhân viên không có phòng ban. Truy vấn SQL sẽ như sau:

```
SELECT e.Name, d.Department
FROM Employees e
LEFT JOIN Departments d ON e.Department = d.Department
WHERE e.Department IS NULL;
```

Kết quả sẽ là:

```
+---------+------------+
| Name    | Department |
+---------+------------+
| Charlie | NULL       |
+---------+------------+
```

So sánh với `=`

Trong SQL, bạn không thể sử dụng toán tử `=` để so sánh với `NULL` vì `NULL` không được xem là một giá trị cụ thể mà là sự thiếu vắng giá trị. Do đó, sử dụng `=` hoặc `!=` với `NULL` sẽ không hoạt động như mong đợi. Thay vào đó, bạn phải sử dụng `IS NULL` hoặc `IS NOT NULL`.

Một số điểm cần lưu ý:

1. **NULL không phải là giá trị**: `NULL` đại diện cho sự thiếu vắng giá trị, do đó, nó không thể so sánh bằng các toán tử thông thường.
2. **Hiệu suất**: Kiểm tra `IS NULL` thường nhanh hơn vì không cần so sánh giá trị cụ thể.
3. **Tương thích với các hàm tổng hợp**: Các hàm tổng hợp như `COUNT`, `SUM`, `AVG`, `MIN`, `MAX` thường bỏ qua các giá trị `NULL`, trừ khi bạn sử dụng các hàm đặc biệt như `COUNT(*)`.

Kết luận

Toán tử `IS NULL` và `IS NOT NULL` rất quan trọng trong SQL để xử lý các giá trị thiếu hoặc không xác định. Chúng giúp bạn dễ dàng kiểm tra và lọc dữ liệu dựa trên sự hiện diện hay vắng mặt của các giá trị `NULL`, đảm bảo rằng bạn có thể làm việc với dữ liệu một cách hiệu quả và chính xác.

▼ **TOÁN TỬ AND - OR - NOT**

Toán tử `AND`, `OR`, và `NOT` trong SQL được sử dụng để kết hợp hoặc đảo ngược các điều kiện trong mệnh đề `WHERE`, giúp lọc dữ liệu theo nhiều điều kiện phức tạp hơn.

Toán tử AND

Toán tử `AND` được sử dụng để kết hợp nhiều điều kiện, chỉ trả về các hàng mà tất cả các điều kiện đều đúng.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE condition1 AND condition2 AND ...;
```

Ví dụ

Giả sử chúng ta có bảng `Employees` như sau:

```
Employees
+----+--------+-----------+--------+
| ID | Name   | Department| Salary |
+----+--------+-----------+--------+
| 1  | Alice  | HR        | 5000   |
| 2  | Bob    | IT        | 6000   |
| 3  | Charlie| HR        | 7000   |
| 4  | David  | IT        | 6000   |
| 5  | Eve    | HR        | 8000   |
+----+--------+-----------+--------+
```

Nếu bạn muốn tìm các nhân viên làm việc trong bộ phận HR và có mức lương trên 6000, truy vấn SQL sẽ như sau:

```
SELECT Name, Department, Salary
FROM Employees
WHERE Department = 'HR' AND Salary > 6000;
```

Kết quả sẽ là:

```
+---------+-----------+--------+
| Name    | Department| Salary |
+---------+-----------+--------+
| Charlie | HR        | 7000   |
| Eve     | HR        | 8000   |
+---------+-----------+--------+
```

Toán tử OR

Toán tử `OR` được sử dụng để kết hợp nhiều điều kiện, trả về các hàng mà ít nhất một trong các điều kiện là đúng.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE condition1 OR condition2 OR ...;
```

Ví dụ

Nếu bạn muốn tìm các nhân viên làm việc trong bộ phận HR hoặc có mức lương trên 6000, truy vấn SQL sẽ như sau:

```
SELECT Name, Department, Salary
FROM Employees
WHERE Department = 'HR' OR Salary > 6000;
```

Kết quả sẽ là:

```
+---------+-----------+--------+
| Name    | Department| Salary |
+---------+-----------+--------+
| Alice   | HR        | 5000   |
| Bob     | IT        | 6000   |
| Charlie | HR        | 7000   |
| David   | IT        | 6000   |
| Eve     | HR        | 8000   |
+---------+-----------+--------+
```

Toán tử NOT

Toán tử `NOT` được sử dụng để đảo ngược điều kiện, trả về các hàng mà điều kiện là sai.

Cú pháp

```
SELECT column_name(s)
FROM table_name
WHERE NOT condition;
```

Ví dụ

Nếu bạn muốn tìm các nhân viên không làm việc trong bộ phận HR, truy vấn SQL sẽ như sau:

```
SELECT Name, Department, Salary
FROM Employees
WHERE NOT Department = 'HR';
```

Kết quả sẽ là:

```
+-------+-----------+--------+
| Name  | Department| Salary |
+-------+-----------+--------+
| Bob   | IT        | 6000   |
| David | IT        | 6000   |
+-------+-----------+--------+
```

Kết hợp AND, OR và NOT

Bạn có thể kết hợp các toán tử `AND`, `OR`, và `NOT` để tạo ra các điều kiện phức tạp hơn. Sử dụng dấu ngoặc đơn `()` để nhóm các điều kiện và kiểm soát thứ tự thực hiện.

Ví dụ

Nếu bạn muốn tìm các nhân viên làm việc trong bộ phận HR và có mức lương trên 6000, hoặc làm việc trong bộ phận IT, truy vấn SQL sẽ như sau:

```
SELECT Name, Department, Salary
FROM Employees
WHERE (Department = 'HR' AND Salary > 6000) OR Departmen
t = 'IT';
```

Kết quả sẽ là:

```
+---------+-----------+--------+
| Name    | Department| Salary |
+---------+-----------+--------+
| Bob     | IT        | 6000   |
| Charlie | HR        | 7000   |
| David   | IT        | 6000   |
| Eve     | HR        | 8000   |
+---------+-----------+--------+
```

Một số điểm cần lưu ý:

1. **Thứ tự thực hiện**: SQL thực hiện các toán tử `NOT` trước, sau đó đến `AND`, và cuối cùng là `OR`. Bạn có thể sử dụng dấu ngoặc để thay đổi thứ tự thực hiện.
2. **Hiệu suất**: Khi viết các truy vấn phức tạp, hãy cân nhắc hiệu suất bằng cách tối ưu hóa thứ tự và cách viết điều kiện để giảm thiểu số lượng hàng được kiểm tra.

Kết luận

Các toán tử `AND`, `OR`, và `NOT` là những công cụ quan trọng trong SQL để kết hợp và đảo ngược các điều kiện trong truy vấn. Hiểu và sử dụng đúng các toán tử này giúp bạn có thể lọc và truy xuất dữ liệu một cách hiệu quả và chính xác.