Dưới đây là một ví dụ về cách tạo một cơ sở dữ liệu trong phpMyAdmin bằng PHP:

php

<?php

// Kết nối đến MySQL

$conn = mysqli\_connect('localhost', 'username', 'password');

// Kiểm tra kết nối

if (!$conn) {

die("Không thể kết nối đến MySQL: " . mysqli\_connect\_error());

}

// Tạo cơ sở dữ liệu

$sql = "CREATE DATABASE mydatabase";

if (mysqli\_query($conn, $sql)) {

echo "Cơ sở dữ liệu đã được tạo thành công";

} else {

echo "Lỗi trong quá trình tạo cơ sở dữ liệu: " . mysqli\_error($conn);

}

// Đóng kết nối

mysqli\_close($conn);

?>

Trong ví dụ trên, bạn cần thay đổi 'localhost', 'username', 'password' thành thông tin kết nối MySQL của bạn. Sau khi chạy mã PHP này, nó sẽ kết nối đến MySQL và tạo một cơ sở dữ liệu có tên 'mydatabase'. Nếu quá trình tạo thành công, thông báo "Cơ sở dữ liệu đã được tạo thành công" sẽ được hiển thị. Nếu có lỗi, thông báo lỗi sẽ được hiển thị.

Dưới đây là một ví dụ về cách tạo một bảng trong phpMyAdmin bằng PHP:

php

<?php

// Kết nối đến MySQL

$conn = mysqli\_connect('localhost', 'username', 'password', 'database');

// Kiểm tra kết nối

if (!$conn) {

die("Không thể kết nối đến MySQL: " . mysqli\_connect\_error());

}

// Tạo bảng

$sql = "CREATE TABLE employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

age INT,

email VARCHAR(100)

)";

if (mysqli\_query($conn, $sql)) {

echo "Bảng đã được tạo thành công";

} else {

echo "Lỗi trong quá trình tạo bảng: " . mysqli\_error($conn);

}

// Đóng kết nối

mysqli\_close($conn);

?>

Trong ví dụ trên, bạn cần thay đổi 'localhost', 'username', 'password', 'database' thành thông tin kết nối MySQL của bạn. Sau khi chạy mã PHP này, nó sẽ kết nối đến cơ sở dữ liệu 'database' và tạo một bảng có tên 'employees' với các cột 'id', 'name', 'age' và 'email'. Cột 'id' được đặt là khóa chính và tự động tăng. Nếu quá trình tạo thành công, thông báo "Bảng đã được tạo thành công" sẽ được hiển thị. Nếu có lỗi, thông báo lỗi sẽ được hiển thị.

Dưới đây là một ví dụ về cách cập nhật dữ liệu vào bảng trong phpMyAdmin bằng PHP:

php

<?php

// Kết nối đến MySQL

$conn = mysqli\_connect('localhost', 'username', 'password', 'database');

// Kiểm tra kết nối

if (!$conn) {

die("Không thể kết nối đến MySQL: " . mysqli\_connect\_error());

}

// Cập nhật dữ liệu vào bảng

$sql = "UPDATE employees SET age = 30 WHERE id = 1";

if (mysqli\_query($conn, $sql)) {

echo "Dữ liệu đã được cập nhật thành công";

} else {

echo "Lỗi trong quá trình cập nhật dữ liệu: " . mysqli\_error($conn);

}

// Đóng kết nối

mysqli\_close($conn);

?>

Trong ví dụ trên, bạn cần thay đổi 'localhost', 'username', 'password', 'database' thành thông tin kết nối MySQL của bạn. Mã PHP này sẽ kết nối đến cơ sở dữ liệu 'database' và cập nhật giá trị 'age' thành 30 cho dòng dữ liệu có 'id' là 1 trong bảng 'employees'. Nếu quá trình cập nhật thành công, thông báo "Dữ liệu đã được cập nhật thành công" sẽ được hiển thị. Nếu có lỗi, thông báo lỗi sẽ được hiển thị.

Dưới đây là một số từ khóa phổ biến để xử lý dữ liệu của bảng trong phpMyAdmin:

1. SELECT: Sử dụng để truy vấn và lấy dữ liệu từ bảng.

Ví dụ: SELECT \* FROM table\_name;

Đúng, câu lệnh SELECT \* FROM table\_name được sử dụng để lấy tất cả các dòng và tất cả các cột từ bảng có tên table\_name. Dưới đây là một ví dụ cụ thể:

Giả sử chúng ta có một bảng có tên "employees" với các cột "id", "name", "age", và "position". Để lấy tất cả thông tin của tất cả nhân viên trong bảng "employees", ta có thể sử dụng câu lệnh sau:

SELECT \* FROM employees;

Kết quả trả về sẽ là tất cả các dòng và cột từ bảng "employees".

2. INSERT INTO: Sử dụng để chèn dữ liệu mới vào bảng.

Ví dụ: INSERT INTO table\_name (column1, column2, column3) VALUES (value1, value2, value3);

câu lệnh INSERT INTO được sử dụng để chèn dữ liệu mới vào bảng. Dưới đây là một ví dụ cụ thể:

Giả sử chúng ta có một bảng có tên "customers" với các cột "id", "name" và "email". Để chèn một khách hàng mới vào bảng "customers", ta có thể sử dụng câu lệnh sau:

INSERT INTO customers (name, email) VALUES ('John Doe', 'john.doe@example.com');

Trong ví dụ này, chúng ta chỉ chèn dữ liệu vào hai cột "name" và "email". Giá trị 'John Doe' được chèn vào cột "name" và giá trị 'john.doe@example.com' được chèn vào cột "email".

Nếu bạn muốn chèn dữ liệu vào tất cả các cột của bảng, bạn có thể sử dụng câu lệnh sau:

INSERT INTO customers VALUES (1, 'John Doe', 'john.doe@example.com');

Trong ví dụ này, chúng ta chèn dữ liệu vào tất cả các cột của bảng theo thứ tự tương ứng. Giá trị 1 được chèn vào cột "id", 'John Doe' được chèn vào cột "name" và 'john.doe@example.com' được chèn vào cột "email".

3. UPDATE: Sử dụng để cập nhật dữ liệu trong bảng.

Ví dụ: UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;

Giả sử chúng ta có một bảng có tên "employees" với các cột "id", "name", "age" và "position". Chúng ta muốn cập nhật thông tin của nhân viên có id là 1. Để làm điều này, ta có thể sử dụng câu lệnh UPDATE như sau:

UPDATE employees

SET name = 'John Smith', age = 30, position = 'Manager'

WHERE id = 1;

Trong câu lệnh UPDATE, ta sử dụng từ khóa SET để chỉ định các giá trị mới cho các cột cần cập nhật. Sau đó, ta sử dụng từ khóa WHERE để xác định rằng chỉ những dòng có id bằng 1 sẽ được cập nhật. Trong ví dụ này, thông tin của nhân viên có id là 1 sẽ được cập nhật với tên là "John Smith", tuổi là 30 và vị trí là "Manager".

4. DELETE: Sử dụng để xóa dữ liệu từ bảng.

Ví dụ: DELETE FROM table\_name WHERE condition;

Giả sử chúng ta có một bảng có tên "employees" với các cột "id", "name", "age" và "position". Chúng ta muốn xóa thông tin của nhân viên có id là 1 khỏi bảng "employees". Để làm điều này, ta có thể sử dụng câu lệnh DELETE FROM như sau:

DELETE FROM employees

WHERE id = 1;

Trong câu lệnh DELETE FROM, ta sử dụng từ khóa WHERE để xác định rằng chỉ những dòng có id bằng 1 sẽ được xóa khỏi bảng. Trong ví dụ này, thông tin của nhân viên có id là 1 sẽ bị xóa khỏi bảng "employees".

5. ALTER TABLE: Sử dụng để thay đổi cấu trúc của bảng, bao gồm thêm hoặc xóa cột, thay đổi kiểu dữ liệu, v.v.

Ví dụ: ALTER TABLE table\_name ADD column\_name data\_type;

Giả sử chúng ta có một bảng có tên "employees" với các cột "id", "name", "age" và "position". Chúng ta muốn thay đổi kiểu dữ liệu của cột "age" từ INT thành VARCHAR(3). Để làm điều này, ta có thể sử dụng câu lệnh ALTER TABLE như sau:

ALTER TABLE employees

ALTER COLUMN age SET DATA TYPE VARCHAR(3);

Trong câu lệnh ALTER TABLE, ta sử dụng từ khóa ALTER COLUMN để chỉ định cột mà ta muốn thay đổi kiểu dữ liệu. Sau đó, ta sử dụng từ khóa SET DATA TYPE để xác định kiểu dữ liệu mới cho cột "age". Trong ví dụ này, kiểu dữ liệu của cột "age" sẽ được thay đổi thành VARCHAR(3).

6. CREATE TABLE: Sử dụng để tạo bảng mới.

Ví dụ: CREATE TABLE table\_name (column1 data\_type, column2 data\_type, column3 data\_type);

Giả sử chúng ta muốn tạo một bảng có tên "students" với các cột "id", "name", "age" và "grade". Cột "id" là kiểu dữ liệu INTEGER và là khóa chính, cột "name" là kiểu dữ liệu VARCHAR(50), cột "age" là kiểu dữ liệu INTEGER và cột "grade" là kiểu dữ liệu VARCHAR(10). Để tạo bảng này, ta có thể sử dụng câu lệnh CREATE TABLE như sau:

CREATE TABLE students (

id INTEGER PRIMARY KEY,

name VARCHAR(50),

age INTEGER,

grade VARCHAR(10)

);

Trong câu lệnh CREATE TABLE, ta sử dụng từ khóa CREATE TABLE để tạo bảng mới. Sau đó, ta chỉ định tên của bảng, và sau đó, ta liệt kê các cột và kiểu dữ liệu tương ứng của chúng. Trong ví dụ này, chúng ta đã tạo bảng "students" với các cột và kiểu dữ liệu tương ứng. Cột "id" được đặt là khóa chính bằng cách sử dụng từ khóa PRIMARY KEY.

7. DROP TABLE: Sử dụng để xóa bảng.

Ví dụ: DROP TABLE table\_name;

8. TRUNCATE TABLE: Sử dụng để xóa toàn bộ dữ liệu trong bảng, nhưng vẫn giữ cấu trúc của bảng.

Ví dụ: TRUNCATE TABLE table\_name;

Giả sử chúng ta có một bảng có tên "students" và giờ chúng ta muốn xóa toàn bộ dữ liệu trong bảng này nhưng vẫn giữ cấu trúc của bảng. Để làm điều này, ta có thể sử dụng câu lệnh TRUNCATE TABLE như sau:

TRUNCATE TABLE students;

Trong câu lệnh TRUNCATE TABLE, ta sử dụng từ khóa TRUNCATE TABLE để xóa toàn bộ dữ liệu trong bảng. Sau đó, ta chỉ định tên của bảng muốn xóa dữ liệu. Trong ví dụ này, toàn bộ dữ liệu trong bảng "students" sẽ bị xóa, nhưng cấu trúc của bảng vẫn được giữ nguyên. Lưu ý rằng việc xóa dữ liệu bằng TRUNCATE TABLE là nhanh hơn so với câu lệnh DELETE FROM vì nó không ghi lại mỗi dòng dữ liệu bị xóa trong bảng gốc. Tuy nhiên, cũng như khi xóa bảng, hãy chắc chắn rằng bạn đã sao lưu dữ liệu quan trọng trước khi thực hiện thao tác này.

9. DESC: Sử dụng để xem thông tin về cấu trúc của bảng, bao gồm tên cột, kiểu dữ liệu, độ dài, v.v.

Ví dụ: DESC table\_name;

Giả sử chúng ta có một bảng có tên "employees" với các cột "id", "name", "age" và "position". Chúng ta muốn xem thông tin về cấu trúc của bảng này. Để làm điều này, ta có thể sử dụng câu lệnh DESC như sau:

DESC employees;

Trong câu lệnh DESC, ta sử dụng từ khóa DESC để xem thông tin về cấu trúc của bảng. Sau đó, ta chỉ định tên của bảng muốn xem. Trong ví dụ này, chúng ta sẽ xem thông tin về cấu trúc của bảng "employees". Kết quả sẽ hiển thị các cột trong bảng và kiểu dữ liệu tương ứng của chúng. Ngoài ra, nó cũng có thể hiển thị các thông tin khác như khóa chính, khóa ngoại, ràng buộc và các chi tiết khác về cấu trúc của bảng.

10. ORDER BY: Sử dụng để sắp xếp dữ liệu theo một hoặc nhiều cột trong bảng.

Ví dụ: SELECT \* FROM table\_name ORDER BY column\_name ASC;

Giả sử chúng ta có một bảng có tên "students" với các cột "id", "name" và "grade". Chúng ta muốn lấy dữ liệu từ bảng này và sắp xếp theo tên của học sinh theo thứ tự từ A đến Z. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh ORDER BY như sau:

SELECT \* FROM students

ORDER BY name ASC;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "students". Sau đó, ta sử dụng câu lệnh ORDER BY để sắp xếp dữ liệu theo cột "name". Trong ví dụ này, chúng ta sử dụng ASC để sắp xếp theo thứ tự tăng dần từ A đến Z. Kết quả sẽ hiển thị các dòng dữ liệu từ bảng "students" được sắp xếp theo tên học sinh theo thứ tự từ A đến Z. Nếu muốn sắp xếp theo thứ tự ngược lại (Z đến A), ta có thể sử dụng DESC thay vì ASC.

11. GROUP BY: Sử dụng để nhóm các dòng dữ liệu dựa trên giá trị của một cột trong bảng.

Ví dụ: SELECT column\_name, COUNT(\*) FROM table\_name GROUP BY column\_name;

Giả sử chúng ta có một bảng có tên "orders" với các cột "order\_id", "customer\_id" và "total\_amount". Chúng ta muốn tính tổng số tiền đã đặt hàng của mỗi khách hàng. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh GROUP BY như sau:

SELECT customer\_id, SUM(total\_amount) as total\_order\_amount

FROM orders

GROUP BY customer\_id;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "orders". Sau đó, ta chỉ định các cột mà chúng ta muốn hiển thị trong kết quả, trong trường hợp này là "customer\_id" và SUM(total\_amount) (tổng số tiền đặt hàng). Ta sử dụng câu lệnh GROUP BY để nhóm các dòng dữ liệu theo cột "customer\_id". Kết quả sẽ hiển thị tổng số tiền đặt hàng của mỗi khách hàng, được nhóm theo "customer\_id".

12. JOIN: Sử dụng để kết hợp các bảng dựa trên một điều kiện chung.

Ví dụ: SELECT \* FROM table1 JOIN table2 ON table1.column\_name = table2.column\_name;

Trong ví dụ này, chúng ta sử dụng câu lệnh SELECT để lấy tất cả dữ liệu từ hai bảng "table1" và "table2" và kết hợp chúng bằng cách sử dụng câu lệnh JOIN.

Câu lệnh JOIN được sử dụng để kết hợp các dòng dữ liệu từ hai bảng dựa trên một điều kiện kết nối. Trong trường hợp này, chúng ta sử dụng câu lệnh ON để chỉ định điều kiện kết nối, trong đó "table1.column\_name" đề cập đến cột trong "table1" và "table2.column\_name" đề cập đến cột trong "table2".

Khi câu lệnh JOIN được thực thi, nó sẽ tạo ra một bảng tạm thời chứa các dòng dữ liệu từ cả hai bảng "table1" và "table2" mà thoả mãn điều kiện kết nối. Kết quả của câu lệnh SELECT sẽ hiển thị tất cả các cột và dòng từ bảng tạm thời này.

Điều này cho phép chúng ta kết hợp thông tin từ hai bảng khác nhau dựa trên điều kiện kết nối, giúp chúng ta lấy được dữ liệu liên quan từ các bảng khác nhau trong cùng một câu lệnh SELECT.

Giả sử chúng ta có hai bảng, "customers" và "orders", với các cột tương ứng là "customer\_id" và "order\_id". Chúng ta muốn lấy thông tin về tên khách hàng và số đơn hàng đã đặt. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh JOIN như sau:

SELECT customers.name, COUNT(orders.order\_id) as total\_orders

FROM customers

JOIN orders ON customers.customer\_id = orders.customer\_id

GROUP BY customers.name;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "customers". Sau đó, ta chỉ định các cột mà chúng ta muốn hiển thị trong kết quả, trong trường hợp này là "customers.name" (tên khách hàng) và COUNT(orders.order\_id) (số đơn hàng). Ta sử dụng câu lệnh JOIN để kết hợp các dòng dữ liệu từ hai bảng "customers" và "orders" dựa trên điều kiện customers.customer\_id = orders.customer\_id. Cuối cùng, ta sử dụng câu lệnh GROUP BY để nhóm các dòng dữ liệu theo tên khách hàng. Kết quả sẽ hiển thị tên khách hàng và số đơn hàng đã đặt của mỗi khách hàng.

13. LIMIT: Sử dụng để giới hạn số lượng dòng dữ liệu được trả về từ câu truy vấn.

Ví dụ: SELECT \* FROM table\_name LIMIT 10;

Giả sử chúng ta có một bảng có tên "products" với các cột "product\_id" và "product\_name". Chúng ta muốn lấy chỉ 5 sản phẩm đầu tiên từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh LIMIT như sau:

SELECT product\_id, product\_name

FROM products

LIMIT 5;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "products". Sau đó, ta chỉ định các cột mà chúng ta muốn hiển thị trong kết quả, trong trường hợp này là "product\_id" và "product\_name". Ta sử dụng câu lệnh LIMIT để chỉ định số lượng dòng dữ liệu mà chúng ta muốn lấy, trong trường hợp này là 5. Kết quả sẽ hiển thị 5 sản phẩm đầu tiên từ bảng "products".

14. DISTINCT: Sử dụng để loại bỏ các giá trị trùng lặp từ kết quả truy vấn.

Ví dụ: SELECT DISTINCT column\_name FROM table\_name;

Giả sử chúng ta có một bảng có tên "customers" với cột "city". Chúng ta muốn lấy danh sách các thành phố duy nhất từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh DISTINCT như sau:

SELECT DISTINCT city

FROM customers;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "customers". Sau đó, ta chỉ định cột "city" mà chúng ta muốn hiển thị trong kết quả. Ta sử dụng câu lệnh DISTINCT để chỉ lấy các giá trị duy nhất từ cột "city". Kết quả sẽ hiển thị danh sách các thành phố duy nhất từ bảng "customers".

15. LIKE: Sử dụng để tìm kiếm các giá trị phù hợp với một mẫu nhất định trong một cột.

Ví dụ: SELECT \* FROM table\_name WHERE column\_name LIKE 'abc%';

Giả sử chúng ta có một bảng có tên "customers" với các cột "customer\_id" và "customer\_name". Chúng ta muốn lấy tất cả các khách hàng có tên bắt đầu bằng "abc" từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh WHERE và LIKE như sau:

SELECT \*

FROM customers

WHERE customer\_name LIKE 'abc%';

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "customers". Ta sử dụng dấu "\*" để lấy tất cả các cột trong bảng. Trong câu lệnh WHERE, ta sử dụng từ khóa WHERE để chỉ định điều kiện lọc dữ liệu. Trong trường hợp này, ta sử dụng câu lệnh LIKE để tìm kiếm các giá trị trong cột "customer\_name" mà bắt đầu bằng "abc". Kết quả sẽ hiển thị tất cả các dòng dữ liệu từ bảng "customers" mà tên khách hàng bắt đầu bằng "abc".

16. COUNT: Sử dụng để đếm số lượng dòng dữ liệu hoặc giá trị duy nhất trong một cột.

Ví dụ: SELECT COUNT(\*) FROM table\_name;

Giả sử chúng ta có một bảng có tên "orders" với các cột "order\_id" và "order\_date". Chúng ta muốn lấy số lượng đơn hàng tổng cộng từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh COUNT như sau:

SELECT COUNT(\*)

FROM orders;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "orders". Trong câu lệnh COUNT(\*), ta sử dụng câu lệnh COUNT để đếm số lượng dòng dữ liệu trong bảng. Dấu "\*" được sử dụng để chỉ định rằng ta muốn đếm tất cả các dòng dữ liệu. Kết quả sẽ hiển thị số lượng đơn hàng tổng cộng trong bảng "orders".

17. SUM: Sử dụng để tính tổng các giá trị trong một cột.

Ví dụ: SELECT SUM(column\_name) FROM table\_name;

Giả sử chúng ta có một bảng có tên "sales" với các cột "product\_id" và "sales\_amount". Chúng ta muốn tính tổng số lượng bán hàng từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh SUM như sau:

SELECT SUM(sales\_amount)

FROM sales;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "sales". Trong câu lệnh SUM(sales\_amount), ta sử dụng câu lệnh SUM để tính tổng giá trị của cột "sales\_amount". Kết quả sẽ hiển thị tổng số lượng bán hàng từ bảng "sales".

18. AVG: Sử dụng để tính trung bình các giá trị trong một cột.

Ví dụ: SELECT AVG(column\_name) FROM table\_name;

Giả sử chúng ta có một bảng có tên "students" với các cột "student\_id" và "score". Chúng ta muốn tính điểm trung bình của các học sinh từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh AVG như sau:

SELECT AVG(score)

FROM students;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "students". Trong câu lệnh AVG(score), ta sử dụng câu lệnh AVG để tính giá trị trung bình của cột "score". Kết quả sẽ hiển thị điểm trung bình của các học sinh từ bảng "students".

19. MAX: Sử dụng để tìm giá trị lớn nhất trong một cột.

Ví dụ: SELECT MAX(column\_name) FROM table\_name;

Giả sử chúng ta có một bảng có tên "products" với các cột "product\_id" và "price". Chúng ta muốn tìm sản phẩm có giá cao nhất từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh MAX như sau:

SELECT MAX(price)

FROM products;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "products". Trong câu lệnh MAX(price), ta sử dụng câu lệnh MAX để tìm giá trị lớn nhất trong cột "price". Kết quả sẽ hiển thị giá cao nhất của sản phẩm từ bảng "products".

20. MIN: Sử dụng để tìm giá trị nhỏ nhất trong một cột.

Ví dụ: SELECT MIN(column\_name) FROM table\_name;

Giả sử chúng ta có một bảng có tên "products" với các cột "product\_id" và "price". Chúng ta muốn tìm sản phẩm có giá thấp nhất từ bảng này. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với câu lệnh MIN như sau:

SELECT MIN(price)

FROM products;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "products". Trong câu lệnh MIN(price), ta sử dụng câu lệnh MIN để tìm giá trị nhỏ nhất trong cột "price". Kết quả sẽ hiển thị giá thấp nhất của sản phẩm từ bảng "products".

21. HAVING: Sử dụng để áp dụng điều kiện cho các hàm tổng hợp sau câu lệnh GROUP BY.

Ví dụ: SELECT column\_name, COUNT(\*) FROM table\_name GROUP BY column\_name HAVING COUNT(\*) > 5;

Giả sử chúng ta có một bảng có tên "orders" với các cột "customer\_id" và "order\_id". Chúng ta muốn tìm các khách hàng có số lượng đơn hàng lớn hơn 5. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với GROUP BY và HAVING như sau:

SELECT customer\_id, COUNT(\*)

FROM orders

GROUP BY customer\_id

HAVING COUNT(\*) > 5;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "orders". Trong câu lệnh GROUP BY customer\_id, ta nhóm dữ liệu theo cột "customer\_id" để tính số lượng đơn hàng cho mỗi khách hàng. Trong câu lệnh HAVING COUNT(\*) > 5, ta chỉ lấy những khách hàng có số lượng đơn hàng lớn hơn 5. Kết quả sẽ hiển thị các khách hàng và số lượng đơn hàng tương ứng của họ, chỉ bao gồm những khách hàng có số lượng đơn hàng lớn hơn 5.

22. CONCAT: Sử dụng để kết hợp giá trị của hai hoặc nhiều cột thành một chuỗi.

Ví dụ: SELECT CONCAT(column1, ' ', column2) AS full\_name FROM table\_name;

Giả sử chúng ta có một bảng có tên "employees" với các cột "first\_name" và "last\_name". Chúng ta muốn tạo ra một cột mới là "full\_name" bằng cách kết hợp cột "first\_name" và "last\_name". Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với CONCAT và AS như sau:

SELECT CONCAT(first\_name, ' ', last\_name) AS full\_name

FROM employees;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "employees". Trong câu lệnh CONCAT(first\_name, ' ', last\_name), ta sử dụng câu lệnh CONCAT để kết hợp cột "first\_name" và "last\_name", và ' ' để thêm một khoảng trắng giữa hai cột. Bằng việc sử dụng AS full\_name, ta đặt tên cột mới là "full\_name". Kết quả sẽ hiển thị cột "full\_name" chứa tên đầy đủ của nhân viên từ bảng "employees".

23. DATE\_FORMAT: Sử dụng để định dạng ngày tháng trong câu truy vấn.

Ví dụ: SELECT DATE\_FORMAT(date\_column, '%Y-%m-%d') FROM table\_name;

Giả sử chúng ta có một bảng có tên "orders" với cột "order\_date" có kiểu dữ liệu là DATE. Chúng ta muốn hiển thị ngày đặt hàng dưới dạng "YYYY-MM-DD". Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với DATE\_FORMAT như sau:

SELECT DATE\_FORMAT(order\_date, '%Y-%m-%d')

FROM orders;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "orders". Trong câu lệnh DATE\_FORMAT(order\_date, '%Y-%m-%d'), ta sử dụng câu lệnh DATE\_FORMAT để định dạng lại cột "order\_date" dưới dạng "YYYY-MM-DD". Kết quả sẽ hiển thị ngày đặt hàng theo định dạng "YYYY-MM-DD" từ bảng "orders".

24. IFNULL: Sử dụng để xác định giá trị mặc định cho một cột nếu nó là NULL.

Ví dụ: SELECT IFNULL(column\_name, 'N/A') FROM table\_name;

Giả sử chúng ta có một bảng có tên "students" với cột "name" và một số sinh viên trong bảng này có giá trị NULL trong cột "name". Chúng ta muốn hiển thị tên sinh viên và nếu giá trị là NULL, thay thế bằng 'N/A'. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với IFNULL như sau:

SELECT IFNULL(name, 'N/A')

FROM students;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "students". Trong câu lệnh IFNULL(name, 'N/A'), ta sử dụng câu lệnh IFNULL để kiểm tra giá trị của cột "name". Nếu giá trị là NULL, ta thay thế bằng 'N/A'. Kết quả sẽ hiển thị tên của sinh viên và nếu giá trị là NULL, sẽ hiển thị 'N/A' từ bảng "students".

25. CASE WHEN: Sử dụng để thực hiện các điều kiện logic trong câu truy vấn.

Ví dụ: SELECT column\_name, CASE WHEN column\_name = 'value1' THEN 'result1' WHEN column\_name = 'value2' THEN 'result2' ELSE 'result3' END FROM table\_name;

Giả sử chúng ta có một bảng có tên "products" với cột "product\_name". Chúng ta muốn hiển thị tên sản phẩm và dựa trên giá trị của cột "product\_name", ta sẽ trả về kết quả tương ứng. Nếu giá trị là 'value1', ta trả về 'result1'. Nếu giá trị là 'value2', ta trả về 'result2'. Nếu không phù hợp với bất kỳ giá trị nào, ta trả về 'result3'. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với CASE WHEN như sau:

SELECT product\_name,

CASE

WHEN product\_name = 'value1' THEN 'result1'

WHEN product\_name = 'value2' THEN 'result2'

ELSE 'result3'

END

FROM products;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy dữ liệu từ bảng "products". Trong câu lệnh CASE WHEN, ta sử dụng các điều kiện để kiểm tra giá trị của cột "product\_name". Nếu giá trị là 'value1', ta trả về 'result1'. Nếu giá trị là 'value2', ta trả về 'result2'. Nếu không phù hợp với bất kỳ giá trị nào, ta trả về 'result3'. Kết quả sẽ hiển thị tên sản phẩm và kết quả tương ứng từ bảng "products".

26. IN: Sử dụng để kiểm tra xem giá trị của một cột có nằm trong danh sách giá trị cho trước hay không.

Ví dụ: SELECT \* FROM table\_name WHERE column\_name IN ('value1', 'value2', 'value3');

Giả sử chúng ta có một bảng có tên "customers" với cột "country". Chúng ta muốn lấy tất cả các dòng trong bảng "customers" mà cột "country" có giá trị là 'value1', 'value2' hoặc 'value3'. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với IN như sau:

SELECT \*

FROM customers

WHERE country IN ('value1', 'value2', 'value3');

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy tất cả các cột từ bảng "customers". Trong câu lệnh WHERE, ta sử dụng IN để kiểm tra giá trị của cột "country". Nếu giá trị của cột "country" là 'value1', 'value2' hoặc 'value3', dòng đó sẽ được lấy. Kết quả sẽ hiển thị tất cả các dòng trong bảng "customers" mà cột "country" có giá trị là 'value1', 'value2' hoặc 'value3'.

27. NOT IN: Sử dụng để kiểm tra xem giá trị của một cột có không nằm trong danh sách giá trị cho trước hay không.

Ví dụ: SELECT \* FROM table\_name WHERE column\_name NOT IN ('value1', 'value2', 'value3');

Giả sử chúng ta có một bảng có tên "products" với cột "category". Chúng ta muốn lấy tất cả các dòng trong bảng "products" mà cột "category" không có giá trị là 'value1', 'value2' hoặc 'value3'. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với NOT IN như sau:

SELECT \*

FROM products

WHERE category NOT IN ('value1', 'value2', 'value3');

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy tất cả các cột từ bảng "products". Trong câu lệnh WHERE, ta sử dụng NOT IN để kiểm tra giá trị của cột "category". Nếu giá trị của cột "category" không phải là 'value1', 'value2' hoặc 'value3', dòng đó sẽ được lấy. Kết quả sẽ hiển thị tất cả các dòng trong bảng "products" mà cột "category" không có giá trị là 'value1', 'value2' hoặc 'value3'.

28. EXISTS: Sử dụng để kiểm tra xem một câu truy vấn con trả về ít nhất một dòng dữ liệu hay không.

Ví dụ: SELECT \* FROM table1 WHERE EXISTS (SELECT \* FROM table2 WHERE table1.column\_name = table2.column\_name);

Giả sử chúng ta có hai bảng "customers" và "orders", mỗi bảng có một cột "customer\_id". Chúng ta muốn lấy tất cả các dòng từ bảng "customers" mà có tồn tại ít nhất một dòng trong bảng "orders" có cùng giá trị trong cột "customer\_id". Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với EXISTS như sau:

SELECT \*

FROM customers

WHERE EXISTS (SELECT \*

FROM orders

WHERE customers.customer\_id = orders.customer\_id);

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy tất cả các cột từ bảng "customers". Trong câu lệnh WHERE, ta sử dụng EXISTS để kiểm tra sự tồn tại của các dòng trong bảng "orders". Nếu tồn tại ít nhất một dòng trong bảng "orders" có cùng giá trị trong cột "customer\_id" với bảng "customers", dòng đó sẽ được lấy. Kết quả sẽ hiển thị tất cả các dòng từ bảng "customers" mà có tồn tại ít nhất một dòng trong bảng "orders" có cùng giá trị trong cột "customer\_id".

29. UNION: Sử dụng để kết hợp kết quả của hai hoặc nhiều câu truy vấn thành một kết quả duy nhất.

Ví dụ: SELECT column1 FROM table1 UNION SELECT column2 FROM table2;

Giả sử chúng ta có hai bảng "employees" và "customers", mỗi bảng có một cột "name". Chúng ta muốn lấy tất cả các giá trị từ cột "name" trong bảng "employees" và bảng "customers" và kết hợp chúng thành một danh sách duy nhất. Để làm điều này, ta có thể sử dụng câu lệnh SELECT kết hợp với UNION như sau:

SELECT name

FROM employees

UNION

SELECT name

FROM customers;

Trong câu lệnh SELECT, ta sử dụng từ khóa SELECT để lấy cột "name" từ bảng "employees". Sau đó, ta sử dụng UNION để kết hợp với câu lệnh SELECT tiếp theo. Trong câu lệnh SELECT thứ hai, ta lấy cột "name" từ bảng "customers". Kết quả sẽ hiển thị tất cả các giá trị từ cột "name" trong bảng "employees" và bảng "customers", và loại bỏ các giá trị trùng lặp, tạo thành một danh sách duy nhất.

30. BACKUP: Sử dụng để sao lưu dữ liệu của bảng hoặc cơ sở dữ liệu.

Ví dụ: BACKUP TABLE table\_name TO '/path/to/backup/file.sql';

31. SHOW: Sử dụng để hiển thị thông tin về cơ sở dữ liệu, bảng và cột.

Ví dụ: SHOW DATABASES; SHOW TABLES; SHOW COLUMNS FROM table\_name;

SHOW DATABASES;

- Câu lệnh này sẽ hiển thị danh sách tất cả các cơ sở dữ liệu trong hệ thống quản trị cơ sở dữ liệu.

SHOW TABLES;

- Câu lệnh này sẽ hiển thị danh sách tất cả các bảng trong cơ sở dữ liệu hiện tại.

SHOW COLUMNS FROM table\_name;

- Câu lệnh này sẽ hiển thị thông tin về các cột trong bảng được chỉ định.

- Bạn cần thay thế "table\_name" bằng tên bảng mà bạn muốn hiển thị thông tin.

32. GRANT: Sử dụng để cấp quyền truy cập và quản lý người dùng trong cơ sở dữ liệu.

Ví dụ: GRANT SELECT, INSERT, UPDATE, DELETE ON table\_name TO 'username'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON database\_name.table\_name TO 'username'@'localhost';

Trong đó:

- database\_name là tên của cơ sở dữ liệu chứa bảng bạn muốn cấp quyền.

- table\_name là tên bảng bạn muốn cấp quyền.

- username là tên người dùng mà bạn muốn cấp quyền.

- localhost là địa chỉ máy chủ mà người dùng được phép truy cập từ.

Ví dụ cụ thể:

GRANT SELECT, INSERT, UPDATE, DELETE ON mydatabase.customers TO 'john'@'localhost';

Trong ví dụ này, chúng ta cấp quyền SELECT, INSERT, UPDATE, DELETE cho người dùng 'john' trên bảng 'customers' trong cơ sở dữ liệu 'mydatabase'. Người dùng 'john' chỉ được phép truy cập từ máy chủ 'localhost'.

33. REVOKE: Sử dụng để thu hồi quyền truy cập đã được cấp cho người dùng trong cơ sở dữ liệu.

Ví dụ: REVOKE SELECT, INSERT, UPDATE, DELETE ON table\_name FROM 'username'@'localhost';

Đây là một ví dụ về câu lệnh REVOKE để thu hồi quyền của một người dùng trên một bảng cụ thể trong MySQL:

REVOKE SELECT, INSERT, UPDATE, DELETE ON database\_name.table\_name FROM 'username'@'localhost';

Trong đó:

- database\_name là tên của cơ sở dữ liệu chứa bảng bạn muốn thu hồi quyền.

- table\_name là tên bảng bạn muốn thu hồi quyền.

- username là tên người dùng mà bạn muốn thu hồi quyền.

- localhost là địa chỉ máy chủ mà người dùng được phép truy cập từ.

Ví dụ cụ thể:

REVOKE SELECT, INSERT, UPDATE, DELETE ON mydatabase.customers FROM 'john'@'localhost';

Trong ví dụ này, chúng ta thu hồi quyền SELECT, INSERT, UPDATE, DELETE của người dùng 'john' trên bảng 'customers' trong cơ sở dữ liệu 'mydatabase'. Người dùng 'john' chỉ được phép truy cập từ máy chủ 'localhost'.

34. COMMIT: Sử dụng để lưu các thay đổi đã thực hiện trong giao dịch.

Ví dụ: COMMIT;

Ví dụ:

BEGIN TRANSACTION;

UPDATE customers SET status = 'active' WHERE id = 1;

INSERT INTO orders (customer\_id, total) VALUES (1, 100);

COMMIT;

Trong ví dụ này, chúng ta bắt đầu một giao dịch bằng câu lệnh BEGIN TRANSACTION. Sau đó, chúng ta thực hiện các thay đổi trong cơ sở dữ liệu như cập nhật trạng thái của khách hàng và thêm một đơn hàng mới. Cuối cùng, chúng ta sử dụng câu lệnh COMMIT để lưu các thay đổi đã được thực hiện vào cơ sở dữ liệu.

Lưu ý rằng câu lệnh COMMIT chỉ được sử dụng trong các hệ quản trị cơ sở dữ liệu hỗ trợ giao dịch, chẳng hạn như MySQL, PostgreSQL, Oracle, và SQL Server.

35. ROLLBACK: Sử dụng để hủy bỏ các thay đổi chưa được lưu trong giao dịch.

Ví dụ: ROLLBACK;

Câu lệnh ROLLBACK được sử dụng để hủy bỏ các thay đổi đã được thực hiện trong một giao dịch (transaction) và khôi phục cơ sở dữ liệu về trạng thái trước khi giao dịch bắt đầu. Dưới đây là một ví dụ về việc sử dụng câu lệnh ROLLBACK trong SQL:

Ví dụ:

BEGIN TRANSACTION;

UPDATE customers SET status = 'active' WHERE id = 1;

INSERT INTO orders (customer\_id, total) VALUES (1, 100);

ROLLBACK;

Trong ví dụ này, chúng ta bắt đầu một giao dịch bằng câu lệnh BEGIN TRANSACTION. Sau đó, chúng ta thực hiện các thay đổi trong cơ sở dữ liệu như cập nhật trạng thái của khách hàng và thêm một đơn hàng mới. Tuy nhiên, thay vì sử dụng câu lệnh COMMIT để lưu các thay đổi, chúng ta sử dụng câu lệnh ROLLBACK để hủy bỏ các thay đổi và khôi phục cơ sở dữ liệu về trạng thái ban đầu.

Lưu ý rằng câu lệnh ROLLBACK chỉ được sử dụng trong các hệ quản trị cơ sở dữ liệu hỗ trợ giao dịch, chẳng hạn như MySQL, PostgreSQL, Oracle, và SQL Server.

36. SET: Sử dụng để thiết lập các biến hệ thống và biến phiên làm việc.

Ví dụ: SET @variable\_name = value;

SET @greeting = 'Hello, World!';

Trong ví dụ này, chúng ta sử dụng câu lệnh SET để gán giá trị 'Hello, World!' cho biến @greeting. Bây giờ, biến @greeting chứa giá trị 'Hello, World!' và có thể được sử dụng trong các câu lệnh SQL khác.

Lưu ý rằng biến trong MySQL bắt đầu bằng ký tự '@' và có thể được sử dụng trong các câu lệnh SQL để lưu trữ và truy xuất giá trị tạm thời.

Ví dụ khác:

SET @num1 = 10;

SET @num2 = 5;

SET @sum = @num1 + @num2;

Trong ví dụ này, chúng ta sử dụng câu lệnh SET để gán giá trị 10 cho biến @num1 và giá trị 5 cho biến @num2. Sau đó, chúng ta sử dụng câu lệnh SET để tính tổng của hai biến và gán kết quả cho biến @sum. Bây giờ, biến @sum chứa giá trị 15.

37. FLUSH: Sử dụng để làm mới các bộ đệm và áp dụng các thay đổi đã thực hiện.

Ví dụ: FLUSH PRIVILEGES;

Câu lệnh FLUSH PRIVILEGES được sử dụng trong MySQL để làm mới (refresh) và áp dụng lại các quyền người dùng sau khi đã thay đổi các quyền hoặc cấu hình trong hệ thống. Dưới đây là một ví dụ về việc sử dụng câu lệnh FLUSH PRIVILEGES:

Ví dụ:

GRANT SELECT ON mydatabase.\* TO 'username'@'localhost';

FLUSH PRIVILEGES;

Trong ví dụ này, chúng ta sử dụng câu lệnh GRANT để cấp quyền SELECT cho người dùng 'username' trên tất cả các bảng trong cơ sở dữ liệu 'mydatabase'. Sau đó, chúng ta sử dụng câu lệnh FLUSH PRIVILEGES để làm mới các quyền người dùng và đảm bảo rằng các thay đổi quyền vừa được thực hiện được áp dụng ngay lập tức.

Lưu ý rằng câu lệnh FLUSH PRIVILEGES chỉ cần được sử dụng sau khi đã thay đổi các quyền người dùng hoặc cấu hình trong hệ thống. Nó không được sử dụng sau mỗi câu lệnh GRANT hoặc REVOKE, mà chỉ khi bạn muốn áp dụng lại tất cả các quyền đã thay đổi trong phiên làm việc hiện tại.

38. OPTIMIZE: Sử dụng để tối ưu hóa bảng và cơ sở dữ liệu để cải thiện hiệu suất.

Ví dụ: OPTIMIZE TABLE table\_name;

OPTIMIZE TABLE customers;

Trong ví dụ này, chúng ta sử dụng câu lệnh OPTIMIZE TABLE để tối ưu hóa bảng 'customers'. Quá trình tối ưu hóa sẽ kiểm tra và sắp xếp lại dữ liệu trong bảng để giảm thiểu không gian lưu trữ và cải thiện hiệu suất truy vấn.

Lưu ý rằng câu lệnh OPTIMIZE TABLE chỉ nên được sử dụng khi cần thiết, ví dụ như sau khi đã thực hiện một số thay đổi lớn trong bảng hoặc khi có hiện tượng giảm hiệu suất truy vấn. Nó không nên được sử dụng quá thường xuyên, vì quá trình tối ưu hóa có thể tốn thời gian và tài nguyên hệ thống.

39. ANALYZE: Sử dụng để phân tích và cập nhật thông tin thống kê của bảng để cải thiện hiệu suất truy vấn.

Ví dụ: ANALYZE TABLE table\_name;

ANALYZE TABLE customers;

Trong ví dụ này, chúng ta sử dụng câu lệnh ANALYZE TABLE để phân tích bảng 'customers'. Quá trình phân tích sẽ xem xét cấu trúc của bảng và dữ liệu hiện có để tạo ra thông tin thống kê như số lượng bản ghi, kích thước bảng, phân phối giá trị và các thống kê khác. Thông tin này được sử dụng bởi trình tối ưu truy vấn của MySQL để đưa ra quyết định tốt nhất về cách thực hiện các truy vấn.

Lưu ý rằng câu lệnh ANALYZE TABLE chỉ nên được sử dụng khi cần thiết, ví dụ như sau khi đã thực hiện một số thay đổi lớn trong bảng hoặc khi có hiện tượng giảm hiệu suất truy vấn. Nó không nên được sử dụng quá thường xuyên, vì quá trình phân tích có thể tốn thời gian và tài nguyên hệ thống.

40. SHOW PROCESSLIST: Sử dụng để hiển thị danh sách các tiến trình đang chạy trong cơ sở dữ liệu.

Ví dụ: SHOW PROCESSLIST;

Câu lệnh SHOW PROCESSLIST được sử dụng trong MySQL để hiển thị danh sách các tiến trình (process) đang chạy trên máy chủ cơ sở dữ liệu. Dưới đây là một ví dụ về việc sử dụng câu lệnh SHOW PROCESSLIST:

Ví dụ:

SHOW PROCESSLIST;

Trong ví dụ này, chúng ta sử dụng câu lệnh SHOW PROCESSLIST để hiển thị danh sách các tiến trình đang chạy. Kết quả trả về sẽ bao gồm các thông tin như ID tiến trình, người dùng đã kết nối, thời gian thực hiện, trạng thái của tiến trình và câu lệnh SQL đang được thực thi (nếu có).

Câu lệnh SHOW PROCESSLIST rất hữu ích để kiểm tra trạng thái của các tiến trình trong cơ sở dữ liệu, đặc biệt là trong trường hợp có tiến trình chạy quá lâu hoặc gây ảnh hưởng đến hiệu suất. Nó cũng cho phép bạn xem xét các kết nối đang mở và các truy vấn đang được thực thi để kiểm tra tình trạng hoạt động của máy chủ cơ sở dữ liệu.

Lưu ý rằng câu lệnh SHOW PROCESSLIST cần được thực thi với quyền SUPER hoặc PROCESS để hiển thị tất cả các tiến trình trên máy chủ.