13 câu lệnh SQL quan trọng Programmer nào cũng cần biết

Cơ sở dữ liệu là một phần không thể thiếu của những trang web hiện đại. Trang web lớn hoặc web động đều sử dụng database theo một cách nào đó và khi được kết hợp với Structured Query Language (SQL) thì khả năng thao tác dữ liệu thực sự là vô tận. Nếu đã biết SQL mà lại còn là lập trình viên thì bạn hãy chắc chắn rằng mình đã nắm chắc 13 câu lệnh SQL quan trọng mà chúng tôi đề cập đến trong bài viết này nhé.

Có rất nhiều tên dữ liệu được trả về từ bảng dữ liệu. Dữ liệu thường được gọi là Rows (hàng), Records (bản ghi) hoặc Tuples. Những thuật ngữ vừa liệt kê sẽ được sử dụng thay thế cho nhau trong suốt bài viết này.

Lời nói đầu

Tất cả các ví dụ ngày hôm nay sẽ được dựa trên bốn bảng giả định.
Bảng **customers** có tên và tuổi của khách hàng:

Name	Age
Joe	99
James	78
Ryan	101

Bảng **heights** có chứa tên và chiều cao của bất kỳ người nào:

Name	Height
Joe	101
James	102
Ryan	103

Bảng **staff** có tên và tuổi của nhân viên - chính xác như **customers**:

Name	Age
Joe	17
James	24
Ryan	18

Bảng cuối cùng được gọi là **people** có tên và tuổi của người, giống như bảng **customers**và **staff**:

Name	Age
Joe	98
James	99
Ryan	100

1. SELECT

Câu lệnh SELECT là đơn giản nhất, và bạn cần phải hiểu nó vì nó làm cơ sở cho khá nhiều lệnh khác. Hãy cân nhắc việc luyện tập viết các lệnh SQL bằng chữ hoa, vì nó làm cho câu lệnh dễ đọc và dễ hiểu hơn.

Như tên của nó ngụ ý, SELECT được sử dụng để chọn dữ liệu từ cơ sở dữ liệu. Đây là cách sử dụng đơn giản nhất:

```
SELECT * FROM table;
```

Câu lệnh trên có hai phần:

- **SELECT ***: xác định cột bạn muốn chọn, dấu * ở đây hiểu là bạn muốn chọn tất cả các cột trong bảng.
- **FROM table**: phần này nói với công cụ cơ sở dữ liệu nơi bạn muốn trích xuất dữ liệu, thay thế "table" bằng tên của bảng cơ sở dữ liệu cần lấy.

Câu lệnh SELECT này được gọi là "select star", sử dụng dấu * là một phương pháp khá hay giúp tìm, tính toán dữ liệu trong bảng, nhưng không phải lúc nào cũng

dùng câu lệnh này. Khi sử dụng select star, việc trình bày dữ liệu trả về như thế nào hoàn toàn phụ thuộc vào engine của database, bạn không thể kiểm soát thứ tự dữ liệu được trả về, vì vậy, nếu có ai đó thêm cột mới vào bảng, bạn thấy các biến trong ngôn ngữ lập trình của mình không hiển thị dữ liệu đúng. May mắn là có một giải pháp khác cho vấn đề này.

Bạn có thể nói rõ các cột muốn truy xuất, như sau:

```
SELECT age, name FROM people;
```

Truy vấn này sẽ trích xuất cột **name** và **age** từ bảng **people**. Việc này có vẻ hơi nhàm chán nếu bạn có quá nhiều dữ liệu, nhưng làm vậy sẽ giúp giảm nhiều vấn đề có thể xảy ra trong tương lai, cũng như làm cho SQL dễ hiểu hơn với các lập trình viên mới sau này.

Nếu bạn muốn chọn thêm dữ liệu bổ sung, nhưng nó không được lưu trữ trong bất kỳ bảng nào, thì có thể làm như sau:

```
SELECT age, '1234' FROM people;
```

Age	?column?
98	1234
99	1234
100	1234

Bất kỳ chuỗi nào bên trong dấu nháy đơn sẽ được trả về thay vì tên cột phù hợp.

2. WHERE

Câu lệnh SELECT là lựa chọn tuyệt vời để lấy dữ liệu, nhưng nếu bạn muốn lọc kết quả kỹ hơn chút nữa, ví như, chỉ muốn trích xuất ra những người có màu mắt xanh, người sinh tháng 1 và làm thợ cơ khí thì phải làm sao? Đây chính là lúc sử dụng câu lệnh WHERE. WHERE cho phép áp dụng thêm các điều kiện vào SELECT, bạn chỉ cần nối nó vào cuối cùng của câu lệnh là được:

```
SELECT age, name FROM people WHERE age > 10;
```

Age	Name
98	Joe
99	James
100	Ryan

Truy vấn này được giới hạn cho những người có tuổi lớn hơn 10. Bạn có thể kết hợp nhiều điều kiện bằng cách sử dụng toán tử AND:

```
SELECT age, name FROM people WHERE age > 10 AND age < 20;
```

Lệnh AND làm việc chính xác như nghĩa của nó trong tiếng Anh: Nó áp dụng những điều kiện khác nhau cho câu lệnh. Trong ví dụ trên, dữ liệu được trả về sẽ là bất kỳ bản ghi nào có tuổi nằm giữa 10 và 20. Do không có kết quả nào phù hợp nên không có dữ liệu nào được trả lại.

Một lệnh khác có thể được sử dụng để kết hợp điều kiện là OR. Đây là ví dụ:

```
SELECT age, name FROM people WHERE age > 10 OR name = 'Joe';
```

Age	Name
98	Joe
99	James
100	Ryan

Truy vấn này yêu cầu trả về những bản ghi có tuổi lớn hơn 10 hoặc tên là Joe. Chú ý, ở đây chỉ có một dấu "=", nhưng nhiều ngôn ngữ lập trình sử dụng 2 dấu bằng (==) để kiểm tra sự tương đương, điều này không cần thiết cho phần lớn các engine của database, xong bạn vẫn nên kiểm tra kỹ trên môi trường làm việc của cơ sở dữ liệu.

3. ORDER

Lệnh ORDER được sử dụng để sắp xếp kết quả trả về, sử dụng ORDER khá đơn giản, chỉ cần thêm ORDER vào cuối câu lệnh như ví dụ dưới đây:

```
SELECT name, age FROM people ORDER BY age DESC;
```

Name	Age
Ryan	100
James	99
Joe	98

Nếu cần chọn cột và thứ tự cụ thể, bạn có thể làm như sau (ASC là tăng dần, DESC là giảm dần):

```
SELECT name, age FROM people ORDER BY name ASC, age DESC;
```

Name	Age
James	99
Joe	98
Ryan	100

ORDER BY có lẽ là hữu ích nhất khi kết hợp với các lệnh khác. Không phải tất cả các truy vấn sẽ trả về dữ liệu một cách hợp lý hoặc có trật tự - lệnh này cho phép bạn thay đổi điều đó.

4. JOIN

Lệnh JOIN được sử dụng để kết hợp các dữ liệu liên quan được lưu trữ trong một hoặc nhiều bảng. Bạn có thể nối bảng thứ hai vào bảng đầu tiên, và chỉ định cách dữ liệu được kết nối. Dưới đây là ví dụ cơ bản:

```
SELECT age, name, height FROM people LEFT JOIN heights USING (name);
```

Có một vài chú ý ở đây. Bạn phải bắt đầu với cú pháp "LEFT JOIN", hiểu rằng bạn muốn nối một bảng bằng cách sử dụng một kiểu nối LEFT. Tiếp theo, xác định bảng mà bạn muốn nối (heights). Cú pháp USING (name) cho biết cột "name" có thể được tìm thấy trong cả hai bảng và cột này sẽ được sử dụng như một chìa khóa để kết hợp các bảng với nhau.

Đừng lo lắng nếu các cột của bạn có tên khác nhau trong mỗi bảng. Bạn có thể sử dụng "ON" thay vì "USING":

```
SELECT age, name, height FROM people LEFT JOIN heights ON (namea = nameb);
```

Age	Name	Height
98	Joe	101
99	James	102
100	Ryan	103

Lệnh ON sẽ xác định rõ cột nào là chìa khóa để nối. Có rất nhiều kiểu nối mà bạn sẽ cần chút thời gian để tìm hiểu chi tiết, đây là một bản tóm tắt nhanh:

- **(INNER) JOIN:** Trả về các hàng có trong cả hai bảng.
- **LEFT (OUTTER) JOIN:** Trả về tất cả các hàng từ bảng bên trái cùng với những bản ghi phù hợp ở bảng bên phải. Nếu không có bản ghi nào phù hợp thì những bản ghi ở bảng bên trái vẫn được trả về.
- **RIGHT (OUTER) JOIN:** Trái ngược với kiểu nối bên trên, tất cả các hàng của bảng bên phải sẽ được trả về cùng với những hàng phù hợp của bảng bên trái.
- **FULL (OUTER) JOIN:** Trả về tất cả những bản ghi phù hợp ở trong hai bảng.

Cú pháp INNER hay OUTER là tùy chọn, nó làm cho mọi thứ dễ hiểu hơn nhưng không nhất thiết lúc nào bạn cũng bắt buộc phải dùng đến chúng.

5. ALIAS

Bây giờ bạn đã biết những câu lệnh cơ bản rồi, thử tiếp với lệnh ALIAS xem sao nhé.

Câu lệnh này được sử dụng để tạm thời đổi tên một bảng, tên mới này chỉ tồn tại bên trong tiến trình xử lý (transaction) bạn đang chạy. Đây là cách sử dụng:

```
SELECT A.age FROM people A;
```

Có thể sử dụng bất kỳ tên phù hợp nào bạn muốn, trong ví dụ này tôi sử dụng các chữ cái trong bảng chữ cái. Trước mỗi tên cột, ALIAS sẽ được đặt trước. ALIAS này được gán cho bảng ngay sau khi khai báo. Tương tự:

```
SELECT people.age FROM people;
```

Thay vì phải nhập tên bảng dài, bạn chỉ cần nhập chữ cái đơn giản, dễ nhớ. Nhưng ở đây có một vấn đề nhỏ, nếu bạn chọn từ nhiều bảng, rất dễ bị nhầm lẫn giữa các cột trong bảng. Trong trường hợp các bảng đó có những cột giống tên nhau, truy vấn cơ sở dữ liệu có thể bị lỗi vì không tham chiếu chính xác được đến tên bảng hoặc ALIAS. Đây là ví dụ với hai bảng:

```
SELECT staff.age, staff.name, customers.age, customers.name FROM staff, customers;
```

Và đây là truy vấn tương tự với các ALIAS:

```
SELECT A.age, A.name, B.age, B.name FROM staff A, customers B;
```

Bảng **staff** được gán tên mới là A, bảng **customers** được gán tên mới là B. Các bảng này giúp code dễ hiểu hơn và giảm số lượng chữ cần phải gõ.

Nếu muốn đổi tên cột với ALIAS, bạn sử dụng lệnh AS:

```
SELECT age AS person_age FROM people;
```

Person age
98
99
100

Khi truy vấn này được thực hiện, cột sẽ được gọi là "person_age" thay vì "age".

6. UNION

UNION là một lệnh tuyệt vời. Nó cho phép bạn nối các hàng với nhau. Không giống như lệnh JOIN chỉ nối thêm các cột phù hợp, UNION có thể nối các hàng không liên quan với nhau nếu có cùng một số lượng cột và tên cột. Đây là cách bạn sử dụng nó:

```
SELECT age, name FROM customersUNION SELECT age, name FROM staff;
```

Age	Name
24	James
101	Ryan
99	Joe
78	James
17	Joe
18	Ryan

Một câu lệnh UNION sẽ chỉ trả về những kết quả là hàng duy nhất giữa 2 truy vấn, bạn có thể sử dụng cú pháp UNION ALL để trả lại tất cả dữ liệu, kể cả những cái trùng nhau.

```
SELECT age, name FROM customersUNION ALLSELECT age, name FROM staff;
```

Age	Name
99	Joe
78	James
101	Ryan
17	Joe
24	James
18	Ryan

Dù kết quả trả về của 2 câu lệnh trên giống nhau, nhưng bạn nhận thấy thứ tự của các hàng có sự thay đổi, đúng không? UNION hoạt động theo cách hiệu quả nhất, vì vậy dữ liệu trả về có thể khác nhau theo thứ tự.

Một trường hợp nữa có thể sử dụng UNION là tính tổng số phụ (subtotal), bạn kết hợp một truy vấn của tổng số (sum total) vào truy vấn của các tổng số riêng lẻ (individual total) cho một tình huống cụ thể. Nghe lằng nhằng nhỉ!

7. INSERT

6 câu lệnh bên trên đều giúp bạn trích xuất dữ liệu từ database, nếu muốn chèn thêm dữ liệu vào database thì làm thế nào? Đây là lúc cho lệnh INSERT thể hiện:

```
INSERT INTO people(name, age) VALUES('Joe', 102);
```

Bạn phải chỉ định tên bảng (**people**) và cột bạn muốn sử dụng (name và age). Cú pháp VALUES sau đó được sử dụng để cung cấp các giá trị cần chèn. Thứ tự của giá trị cần chèn phải được đặt đúng như thứ tự của các cột đã được chỉ định trước đó.

Bạn không thể chỉ định WHERE để chèn, và cần đảm bảo rằng đã tuân thủ đúng các ràng buộc giữa các bảng.

8. UPDATE

Sau khi chèn thêm dữ liệu, bạn cần phải thay đổi các hàng cụ thể. Đây là cú pháp của lệnh UPDATE:

UPDATE people SET name = 'Joe', age = 101;

Bạn phải chỉ định bảng muốn thay đổi, sau đó sử dụng cú pháp SET để xác định các cột và các giá trị mới của chúng. Câu lệnh trong ví dụ này sẽ cập nhật tất cả bản ghi riêng lẻ.

Để cụ thể hơn, bạn có thể sử dụng WHERE giống như khi thực hiện lệnh SELECT:

```
UPDATE people SET name = 'Joe', age = 101 WHERE name = 'James';
```

Thậm chí, có thể sử dụng cả toán tử điều kiện AND, OR:

```
UPDATE people SET name = 'Joe', age = 101 WHERE (name = 'James' AND age = 100) OR name = 'Ryan';
```

Hãy chú ý cách mà dấu ngoặc đơn được sử dụng để bắt buộc tuân theo các điều kiện.

9. UPSERT

UPSERT nghe có vẻ lạ, nhưng đây lại là lệnh khá hữu ích. Giả sử có một hạn chế trên bảng dữ liệu là bạn chỉ lưu những bản ghi với tên duy nhất, bạn không muốn có hai hàng trùng tên nhau xuất hiện trong bảng. Khi đó nếu cố gắng chèn nhiều giá trị "Joe" vào thì engine của database sẽ báo lỗi và từ chối làm điều đó (gần như vậy). Lệnh UPSERT cho phép bạn cập nhật bản ghi nếu nó đã tồn tại. Nếu không có lệnh này, bạn sẽ phải viết rất nhiều logic để kiểm tra như kiểm tra xem nó đã tồn tại chưa, nếu chưa tồn tại thì chèn, nếu đã tồn tại thì trích xuất khóa chính (primary key) chính xác của nó rồi cập nhật. Thật là muốn phát điên luôn mà...

Tiếc là lệnh này được thực hiện khác nhau trên những database khác nhau. PostgreSQL gần đây đã có thêm lệnh này, trong khi MySQL đã có từ rất lâu. Đây là cú pháp lệnh UPSERT trên MySQL để bạn tham khảo:

```
INSERT INTO people(name, age)VALUES('Joe', 101)ON DUPLICATE KEY UPDATE age = 101;
```

Nếu tinh ý, bạn sẽ nhận thấy rằng cách này thực chất là một lệnh cập nhật kết hợp với lệnh chèn, có thể hiểu là "cập nhật nếu chèn không thành công".

10. DELETE

Lệnh DELETE được sử dụng để xóa hoàn toàn các bản ghi, nó có thể khá nguy hiểm nếu bị lạm dụng. Cú pháp của lệnh này khá đơn giản:

```
DELETE FROM people;
```

Câu lệnh trên sẽ xóa mọi thứ từ bảng **people**. Nếu chỉ muốn xóa những bản ghi nhất định hãy sử dụng thêm WHERE:

```
DELETE FROM people WHERE name = 'Joe';
```

Nếu bạn đang phát triển một hệ thống thì cách khôn ngoan hơn là sử dụng một lệnh "soft delete". Cụ thể, bạn không bao giờ thực sự chạy một lệnh DELETE, mà tạo một cột đã xóa (chuyển dữ liệu sang đó), kiểm tra cột một lần nữa để tránh những trường hợp xóa nhầm đáng tiếc. Cách này cũng giúp nhanh chóng lấy lại bản ghi nếu phát hiện lỗi hay vấn đề cần kiểm tra lại. Tất nhiên, đây không phải là lựa chọn sao lưu thích hợp đâu nhé. Hãy cứ thực hiện sao lưu hệ thống của bạn, bởi cẩn tắc vô áy náy mà.

11. CREATE TABLE

Vâng, đúng như tên gọi, lệnh này được sử dụng để tạo bảng, và đây là cú pháp của nó:

```
CREATE TABLE people (name TEXT,age, INTEGER,PRIMARY KEY(name));
```

Chú ý cách các tên cột, ràng buộc nằm trong ngoặc và gán kiểu dữ liệu cho cột được viết như thế nào. Khóa chính cũng cần được chỉ định, đây là yêu cầu đầu tiên của một thiết kế database chuẩn.

12. ALTER TABLE

Lệnh ALTER TABLE được sử dụng để sửa đổi cấu trúc của một bảng. Ở đây có một chút hạn chế, vì cơ sở dữ liệu của bạn sẽ không cho phép thay đổi một bảng nếu dữ liệu đang tồn tại có thể gây ra xung đột, ví dụ, thay đổi một chuỗi thành một số nguyên. Trong những trường hợp này, cần sửa dữ liệu trước, sau đó sửa đổi bảng. Đây là ví dụ:

```
ALTER TABLE people ADD height integer;
```

Ví dụ này thêm một cột được gọi là "height" với kiểu dữ liệu là số nguyên vào bảng **people**. Không có giới hạn về những gì bạn có thể thay đổi.

13. DROP TABLE

Lệnh cuối cùng là DROP TABLE. Lệnh này cũng gần giống với DELETE nhưng thay vì xóa một bản ghi duy nhất, nó xóa mọi bản ghi trong bảng. Đây là cách sử dụng nó:

```
DROP TABLE people;
```

Lệnh này khá nguy hiểm, vì thế nên thực hiện nó bằng tay trong phần lớn các trường hợp, đề phòng những lỗi không mong muốn có thể xảy ra.

Xong rồi, 13 lệnh tất cả, hy vọng bạn đã bỏ túi được một số thủ thuật hữu ích khi làm việc với cơ sở dữ liệu. Hãy chia sẻ với chúng tôi những câu lệnh, thủ thuật SQL khác mà bạn đã khám phá được nhé!