# SUMARY POSTGRES SQL

# PHẦN 1: CÁC CÂU LỆNH ĐẶC TRƯNG

#### 1.1 Tạo một cơ sở dữ liệu (p.24)

Để tạo một cơ sở dữ liệu mới, trong ví dụ này tên là mydb, bạn sử dụng lệnh sau:

\$ createdb mydb

#### **1.2 Tạo một bảng mới** (p.29)

```
CREATE TABLE weather (
city varchar(80),
temp_lo int,
temp_hi int,
prcp real,
date date
);
```

#### 1.3 Xoá một bảng (p.29)

**DROP TABLE tablename**;

#### 1.4 Chèn dữ liệu vào bảng (p.29)

INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25, '1994-11-27');

Có thể sử dụng câu lệnh để chép dữ liệu khổng lồ vào bảng dữ liệu thông qua câu lệnh COPY

#### COPY weather FROM '/home/user/weather.txt';

\* Lưu ý: Sử dụng file nằm trên hệ thống nguồn của máy chủ chứ không phải là máy trạm.

#### **1.5 Dạng dữ liệu** (p.29)

PostgreSQL hỗ trợ các dạng SQL tiêu chuẩn int, smallint, real, double, precision, char(N), varchar(N), date, time, timestamp, interval, point.

#### 1.6 Truy vấn dữ liệu (p.30)

Sử dụng các câu lệnh truy vấn dựa trên câu lệnh SELECT

| SELECT * FROM weather  | Lấy tất cả các cột   |
|--|--|
| SELECT city, (temp_hi+temp_lo)/2<br>AS temp_avg, date FROM weather | Sử dụng biểu thức. Mệnh đề AS<br>được sử dụng để gắn lại nhãn cho<br>cột đầu ra.                       |
| SELECT DISTINCT city FROM weather, city                            | Loại bỏ kết quả trùng nhau khỏi kết<br>quả của một truy vấn.   |
| SELECT DISTINCT city FROM weather ORDER BY city                    | Bạn có thể đảm bảo các kết quả<br>nhất quán bằng việc sử dụng cùng<br>một lúc cả DISTINCT và ORDER BY2 |

#### 1.7 Liên kết giữa các bảng dữ liệu (p.32-33)

Bạn cần so sánh cột thành phố (city) của từng hàng trong bảng thời tiết (weather) với cột tên (name).

```
SELECT *
FROM weather, cities
WHERE city = name;
```

| city                           | temp_lo |    |      |            | name<br>                         | location  |
|--------------------------------|---------|----|------|------------|----------------------------------|-----------|
| San Francisco<br>San Francisco | 46      | 50 | 0.25 | 1994-11-27 | San Francisco<br>  San Francisco | (-194,53) |
| (2 rows)                       |         |    |      |            |                                  |           |

Có thể sử dụng câu lệnh:

SELECT city, temp\_lo, temp\_hi, prcp, date, location FROM weather, cities WHERE city = name;

#### **1.8 Các hàm tổng hợp** (p.34)

Các hàm tổng hợp để tính toán như count, sum, avg (average - trung bình), max (maximum - tối đa) và min (minimum - tối thiểu) đối với một tập hợp các hàng.

Ví dụ, chúng ta có thể tìm nhiệt độ thấp cao nhất ở đâu đó với:

SELECT max(temp\_lo) FROM weather;

Tuy nhiên có một vài hạn chế các hàm này không thể đứng sau mệnh đề WHERE nên tối ưu hơn cả là sử dụng câu lệnh truy vấn con:

SELECT city FROM weather WHERE temp\_lo = (SELECT max(temp\_lo) FROM weather);

Truy vấn con là một tính toán độc lập.

Các tổng hợp cũng rất hữu dụng trong sự kết hợp với các mệnh đề GROUP BY. Ví dụ, chúng ta có thể có được nhiệt độ thấp nhất được quan sát thấy trong từng thành phố với:

SELECT city, max(temp\_lo) FROM weather GROUP BY city;

Chúng ta có thể lọc các hàng được nhóm lại đó bằng việc sử dụng HAVING

SELECT city, max(temp\_lo) FROM weather GROUP BY city HAVING max(temp\_lo) < 40;

#### 1.9 Câu lệnh cập nhật - Update (p.36)

Bạn có thể cập nhật các hàng đang tồn tại bằng việc sử dụng lệnh cập nhật.

Giả sử bạn phát hiện ra việc đọc nhiệt độ tất cả là lệch 2 độ sau ngày 28/11. Bạn có thể sửa các dữ liệu như sau:

UPDATE weather SET temp\_hi = temp\_hi - 2, temp\_lo = temp\_lo - 2 WHERE date > '1994-11-28'

#### **1.20 Câu lệnh xoá - Delete** (p.36)

**DELETE FROM weather WHERE city = 'Hayward'** 

Nên thận trọng đối với các câu lệnh dạng

**DELETE FROM tablename;** 

#### 1.21 Câu lệnh truy vấn vòng ngoài (p.33)

Truy vấn này sử dụng khi mệnh đề WHERE bị bỏ qua. Vì các cột đã có các tên khác nhau, nên trình phân tích cú pháp tự động thấy được bảng nào chúng thuộc về. Nếu có các tên cột trùng nhau trong 2 bảng đó thì bạn cần phải định tính các tên cột để chỉ ra cột nào ban ngu ý, như sau:

SELECT weather.city, weather.temp\_lo, weather.temp\_hi, weather.prcp, weather.date, cities.location

FROM weather, cities WHERE cities.name = weather.city;

Thì viết phải viết lại theo cách truy vấn vòng ngoài.

SELECT \* FROM weather INNER JOIN cities ON (weather.city = cities.name);

| city  | temp_lo            |            |      |                        | name<br> | location |
|---|--------------------|------------|------|------------------------|----------|----------|
| Hayward<br>San Francisco<br>San Francisco<br>(3 rows) | 37<br>  46<br>  43 | 54  <br>50 | 0.25 | 1994-11-<br>  1994-11- |          |          |

#### 1.22 Câu lệnh tự liên kết (p.33)

SELECT W1.city, W1.temp\_lo AS low, W1.temp\_hi AS high, W2.city, W2.temp\_lo AS low, W2.temp\_hi AS high FROM weather W1, weather W2 WHERE W1.temp\_lo < W2.temp\_lo AND W1.temp\_hi > W2.temp\_hi;

| city          | •            | high | •                                |    | high |
|---------------|--------------|------|----------------------------------|----|------|
| San Francisco | 43  <br>  37 | 57   | San Francisco<br>  San Francisco | 46 | 50   |

Ở đây chúng ta đã gắn lại nhãn cho bảng thời tiết như là W1 và W2 để có khả năng phân biệt được phía bên trái và bên phải của liên kết.

**SELECT \* FROM weather w, cities c WHERE w.city = c.name;** 

### PHẦN 2: CÁC TÍNH NĂNG CAO CẤP

#### 2.1 Các khoá ngoại (p.37)

Các khoá ngoại để duy trì tính toàn vẹn tham chiếu, tránh trường hợp dữ liệu đưa vào không đủ điều kiện, cấu trúc và tạo nên lỗ hổng dữ liệu.

Khai báo mới về các bảng có thể trông giống thế này:

CREATE TABLE cities (city varchar(80) primary key, location point);

```
CREATE TABLE weather (
city varchar(80) references cities(city),
temp_lo int,
temp_hi int,
prcp real,
date date
);
```

Bây giờ cố gắng chèn một bản ghi hợp lệ vào:

INSERT INTO weather VALUES ('Berkeley', 45, 53, 0.0, '1994-11-28');

ERROR: insert or update on table "weather" violates foreign key constraint "weather\_city\_DETAIL: Key (city)=(Berkeley) is not present in table "cities".

#### **2.2 Các giao dịch** (p.38)

Các giao dịch là tập hợp nhiều bước trong một bước duy nhất, một hoạt động hoặc tất cả hoặc không có gì xảy ra.

Bảng quyết toán cân bằng thu chi cho các tài khoản khác nhau của người sử dụng, cũng như tổng cân bằng tiền gửi đối với các chi nhánh.

Giả sử là chúng ta muốn ghi lại thanh toán của 100.00 USD từ tài khoản của Alice cho tài khoản của Bob.

UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice';

UPDATE branches SET balance = balance - 100.00 WHERE name = (SELECT branch\_name FROM accounts WHERE name = 'Alice');

```
UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob';
```

```
UPDATE branches SET balance = balance + 100.00
WHERE name = (SELECT branch_name FROM accounts
WHERE name = 'Bob');
```

Cơ sở dữ liệu của một giao dịch đảm bảo rằng tất cả các bản cập nhật được một giao dịch thực hiện bị khóa trong lưu trữ vĩnh cửu (như, trên đĩa cứng) trước khi giao dịch đó được nói là hoàn tất.

Trong PostgreSQL, một giao dịch được thiết lập bằng các lệnh SQL bao quanh giao dịch đó với các lệnh bắt đầu - BEGIN và thực hiện - COMMIT. Vì thế giao dịch ngân hàng của chúng ta có lễ thực sự trông giống như:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
WHERE name = 'Alice';
-- etc etc
COMMIT;
```

Giả thiết chúng ta ghi nợ 100.00 USD từ tài khoản của Alice, và ghi có cho tài khoản của Bob, sẽ chỉ thấy sau này rằng chúng ta nên có tài khoản tin cậy của Wally.

BEGIN;

UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice'; SAVEPOINT my\_savepoint;

UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob';

-- oops ... forget that and use Wally's account ROLLBACK TO my\_savepoint;

UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Wally'; COMMIT;

#### 2.3 Hàm cửa số (p.40)

Một hàm cửa sổ thực hiện một tính toán qua một tập hợp các hàng của bảng mà bằng cách nào đó có liên quan tới hàng hiện hành.

VD: Đây là một ví dụ chỉ ra cách để so sánh từng khoản lương của nhân viên với lương trung bình trong phòng của anh hoặc chị ta.

SELECT depname, empno, salary, avg(salary)
OVER (PARTITION BY depname)
FROM empsalary;

| depname   | empno | salary | avg                    |
|-----------|-------|--------|------------------------|
| develop   | 11    |        | 5020.00000000000000000 |
| develop   | 7     | 4200   | 5020.0000000000000000  |
| develop   | 9     | 4500   | 5020.0000000000000000  |
| develop   | j 8 j | 6000   | 5020.00000000000000000 |
| develop   | 10    | 5200   | 5020.0000000000000000  |
| personnel | 5     | 3500   | 3700.00000000000000000 |
| personnel | j 2 j | 3900   | 3700.0000000000000000  |
| sales     | ] 3   | 4800   | 4866.66666666666666    |
| sales     | 1     | 5000   | 4866.66666666666666    |
| sales     | 4     | 4800   | 4866.66666666666666    |
| (10 rows) |       |        |                        |

SELECT depname, empno, salary, rank() OVER (PARTITION BY depname ORDER BY salary DESC) FROM empsalary

#### **2.4** Sự kế thừa (p.43-45)

Hãy tạo 2 bảng. Một bảng các thành phố - cities và một bảng các thủ phủ - capitals. Một cách tự nhiên, các thủ phủ cũng là các thành phố, nên bạn muốn một số cách để trình bày các thủ phủ một cách ẩn khi bạn liệt kê tất cả các thành phố.

```
CREATE TABLE capitals (
name text,
population real,
altitude int,
-- (in ft) state char(2)
);
CREATE TABLE non_capitals (
name text,
population real,
altitude int -- (in ft)
);
CREATE VIEW cities AS SELECT name, population, altitude
FROM capitals UNION SELECT name, population, altitude
FROM non_capitals;
```

```
CREATE TABLE cities (
name text,
population real,
altitude int -- (in ft) );
CREATE TABLE capitals (
state char(2)
) INHERITS (cities);
```

Trong trường hợp này, một hàng của bảng các thủ phủ kế thừa tất cả các cột (name, population, và altitude) từ bảng cha của nó, bảng các thành phố.

Trong PostgreSQL, một bảng có thể kế thừa từ 0 hoặc nhiều hơn các bảng khác.

#### 2.5 Toán tử (p.55)

```
+-*/<>=~!@#%^&|'?
```

#### 2.6 Ràng buộc kiểm tra (p.73)

Một ràng buộc kiểm tra là dạng ràng buộc phổ biến nhất. Boolean (giá trị đúng – true).

Ví dụ, để yêu cầu các giá thành sản phẩm phải lớn hơn 0, là một số dương, bạn có thể sử dụng.

```
CREATE TABLE products (
product_no integer,
name text,
price numeric CHECK (price > 0)
);
```

#### **2.7** Khoá ngoại (p.78)

Một ràng buộc khóa ngoại (Foreign Key) chỉ định rằng các giá trị trong một cột (hoặc một nhóm các cột) phải khớp với các giá trị xuất hiện trong một số hàng của bảng khác.

```
CREATE TABLE products (
product_no integer PRIMARY KEY,
name text,
price numeric
);
```

Hãy cũng giả thiết bạn có một bảng lưu trữ các đơn hàng của các sản phẩm đó. Chúng ta muốn đảm bảo rằng bảng các đơn hàng chỉ chứa các đơn hàng của các sản phẩm mà thực sự tồn tại.

```
CREATE TABLE orders (
order_id integer PRIMARY KEY,
product_no integer REFERENCES products (product_no),
quantity integer
);
```

Một khóa chính cũng có thể ràng buộc và tham chiếu tới một nhóm các cột. Như thường lệ, nó sau đó cần phải được viết ở dạng ràng buộc bảng. Đây là một ví dụ cú pháp được trù tính trước:

```
CREATE TABLE t1 (
a integer PRIMARY KEY,
b integer,
c integer,
FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

#### 2.8 Sửa đổi cấu trúc bảng (p.83)

// Sửa bảng sản phẩm và thêm cột văn bản mô tả

#### **ALTER TABLE products ADD COLUMN description text**

#### 2.9 Loại bỏ cột (p.83)

// Để loại bỏ một cột, hãy sử dụng một lệnh giống như là:

#### **ALTER TABLE products DROP COLUMN description**

#### **2.10** Thêm ràng buộc (p.84)

Để thêm một ràng buộc, cú pháp ràng buộc bảng được sử dụng. Ví dụ:

ALTER TABLE products ADD CHECK (name <> ");

ALTER TABLE products ADD CONSTRAINT some\_name UNIQUE (product\_no);

ALTER TABLE products ADD FOREIGN KEY (product\_group\_id) REFERENCES product\_groups;

Để thêm một ràng buộc không null, mà nó không thể được viết như một ràng buộc bảng, hãy sử dụng cú pháp này.

ALTER TABLE products ALTER COLUMN product\_no SET NOT NULL

Ràng buộc đó sẽ được kiểm tra ngay lập tức, nên dữ liệu của bảng phải làm thỏa mãn ràng buộc đó trước khi nó có thể được thêm vào.

#### 2.11 Loại bỏ ràng buộc (p.84-86)

Để loại bỏ một ràng buộc thì bạn cần biết tên của nó.

Lệnh **psql \d tablename** có thể tìm thấy tên của danh sách các bảng.

ALTER TABLE products DROP CONSTRAINT some\_name;

# 2.12 Sơ đồ SCHEMA (p.87) A Tạo sơ đồ CREATE SCHEMA myschema; CREATE TABLE myschema.mytable ( ... ); B Loại bỏ schema khi dữ liệu liên quan đều trống DROP SCHEMA myschema CASCADE;

C Loại bỏ schema khi dữ liệu liên quan đều trống

**CREATE SCHEMA schemaname AUTHORIZATION username;** 

**DROP SCHEMA myschema CASCADE;** 

# PHẦN 3 : DẠNG DỮ LIỆU

## 1.1 Một số dạng dữ liệu (p.129)

PostgreSQL có một tập hợp giàu có các dạng dữ liệu bẩm sinh sẵn có cho người sử dụng. Người sử dụng có thể thêm các dạng mới cho PostgreSQL bằng việc sử dụng lệnh tạo dạng CREATE TYPE.

| Tên                        | Tên hiệu        | Mô tả                                   |
|----------------------------|-----------------|---|
| bigint                     | int8            | số nguyên 8 byte được ký                |
| bigserial                  | serial8         | số nguyên 8 byte tự động tăng           |
| bit [ (n) ]                |                 | chuỗi bit độ dài cố định                |
| bit varying [ (n) ]        | varbit          | chuỗi bit độ dài biến đổi               |
| boolean                    | bool            | Boolean logic (đúng/sai - true/false)   |
| box                        | -               | hộp chữ nhật trên một mặt phẳng         |
| bytea                      |                 | dữ liệu nhị phân ("mảng theo byte")     |
| character varying [ (n)]   | varchar [ (n) ] | chuỗi ký tự độ dài biến đổi             |
| character [ (n) ]          | char [ (n) ]    | chuỗi ký tự độ dài cố định              |
| cidr                       |                 | địa chỉ mạng IPv4 hoặc IPv6             |
| circle                     |                 | mạch trên mặt phẳng                     |
| date                       |                 | ngày tháng theo lịch (năm, tháng, ngày) |
| double precision           | float8          | số các chấm động chính xác đúp (8 byte) |
| inet                       |                 | địa chỉ máy chủ theo IPv4 hoặc IPv6     |
| integer                    | int, int4       | số nguyên 4 byte được ký                |
| interval [ fields ] [(p) ] |                 | khoảng thời gian                        |
| line                       |                 | đường vô cực trên một mặt phẳng         |

| Iseg               |                    | đoạn thẳng trên mặt phẳng  |
|--------------------|--------------------|--|
| macaddr            |                    | địa chỉ MAC (Kiểm soát Truy cập Phương tiện - Media Access<br>Control) |
| money              |                    | lượng hiện hành  |
| numeric [ (p, s) ] | decimal [ (p, s) ] | số chính xác của độ chính xác có khả năng chọn được                    |
| path               |                    | đường địa lý trên mặt phẳng  |
| point              |                    | điểm địa lý trên mặt phẳng   |
| polygon            |                    | đường địa lý khép kín trên mặt phẳng                                   |
| real               | float4             | điểm chấm động chính xác duy nhất (4 byte)                             |
| smallint           | int2               | số nguyên 2 byte được ký   |
| serial             | serial4            | số nguyên 4 byte tự động tăng  |
|                    |                    |  |

| Tên  | Tên hiệu    | Mô tả   |
|--|-------------|---|
| text                                       |             | chuỗi ký tự độ dài biến đổi                       |
| time [ (p) ] [ without<br>time zone ]      |             | thời gian trong ngày (không có vùng thời gian)    |
| time [ (p) ] with time<br>zone             | timetz      | thời gian trong ngày, có vùng thời gian           |
| timestamp [ (p) ] [<br>without time zone ] |             | ngày tháng và thời gian (không có vùng thời gian) |
| timestamp [ (p) ] with<br>time zone        | timestamptz | ngày tháng và thời gian, có vùng thời gian        |
| tsquery                                    |             | truy vấn tìm kiếm văn bản                         |
| tsvector                                   |             | tài liệu tìm kiếm văn bản                         |
| txid_snapshot                              |             | hình chụp ID giao dịch mức người sử dụng          |
| uuid                                       |             | mã định danh độc nhất vạn năng                    |
| xml  |             | dữ liệu XML                                       |

# PHẦN 4: JSON & JSON B

Kiểu dữ liệu được lưu dưới dạng Json là kiểu key/value. Với tốc độ xử lý nhanh chóng, cấu trúc đơn giản.

#### 4.1 Tạo bảng với HSTORE

```
CREATE TABLE hstore_data (data HSTORE);
```

#### 4.2 Thêm dữ liệu vào HSTORE

```
INSERT INTO hstore_data (data)VALUES ('cost"=>"500",
"product"=>"iphone",
"provider"=>"apple"
');
```

#### 4.3 Truy vấn dữ liệu HSTORE

```
SELECT data FROM hstore_data;
```

```
--Dữ liệu hiển thị --
"cost"=>"500","product"=>"iphone","provider"=>"Apple"
```

#### 4.4 Tạo bảng với JSON B

```
CREATE TABLE json_data (data JSONB);
```

#### 4.5 Thêm dữ liệu vào JSON B

```
INSERT INTO json_data (data) VALUES ('
{
"name": "Apple Phone",
"type": "phone",
"brand": "ACME",
"price": 200,
"available": true,
"warranty_years": 1
}
')
```

#### 4.6 Dữ liệu JSON B sắp xếp

```
{"name": "Apple Phone", "type": "phone", "brand": "ACME", "price": 200, "available": true, "warranty_years": 1}
```

#### 4.7 Dữ liệu JSON B kiến trúc tổ kén

```
{"full name": "John Joseph Carl Salinger",
    "names": [
    {"type": "firstname", "value": "John"},
    {"type": "middlename", "value": "Joseph"},
    {"type": "middlename", "value": "Carl"},
    {"type": "lastname", "value": "Salinger"}
    ]}
```

#### 4.8 Sự kết nối giữa SQL and JSON

Truy vấn trả về tập dữ liệu chuẩn

```
SELECT * FROM products;
id | product_name
------
1 | iPhone
2 | Samsung
3 | Nokia
```

Chọn truy vấn trả về cùng một kết quả dưới dạng tập dữ liệu JSON

```
SELECT ROW_TO_JSON(products) FROM products;

{"id":1,"product_name":"iPhone"}

{"id":2,"product_name":"Samsung"}

{"id":3,"product_name":"Nokia"}
```

# PHÂN 5: PostgreSQL JSON

#### 5.1 Tạo bảng

```
CREATE TABLE orders (
id serial NOT NULL PRIMARY KEY,
info json NOT NULL
);
```

Bảng **orders** có 2 cột:

Cột **id** là khoá chính được xác định. Cột **info** dùng để lưu trữ định dạng JSON.

#### 5.2 Thêm mới dữ liệu JSON data

```
INSERT INTO orders (info)
VALUES('{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}');
Trong bảng này John Doe có 6 hộp beers.
```

```
INSERT INTO orders (info)
VALUES
('{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}'),
('{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}'),
('{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}');
```

#### 5.3 Truy vấn dữ liệu JSON data

#### **SELECT info FROM orders**;

```
info

{ "customer": "John Doe", "items": {"product": "Beer", "qty": 6}}

{ "customer": "Lily Bush", "items": {"product": "Diaper", "qty": 24}}

{ "customer": "Josh William", "items": {"product": "Toy Car", "qty": 1}}

{ "customer": "Mary Clark", "items": {"product": "Toy Train", "qty": 2}}
```

PostgreSQL cung cấp hai hình thức truy vấn JSON ( -> và ->>)

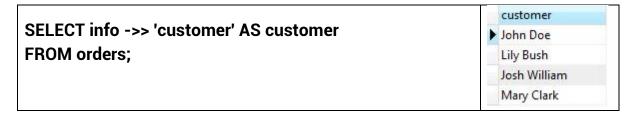
Dấu -> trả về giá trị JSON bằng khoá key.

Dấu ->> trả về JSON dữ liệu trường text.

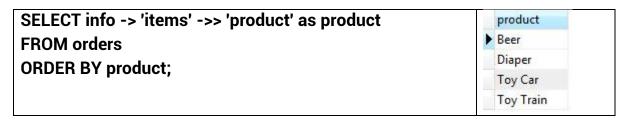
Câu lệnh sử dụng dấu -> lấy tất cả customers trong form JSON:



Câu lệnh sử dụng ->> lấy tất cả customers trong file text:



Bởi vì -> trả về một JSON object, bạn có thể liên kết nó với toán tử - >> để truy xuất một nút cụ thể. Ví dụ: câu lệnh sau trả về tất cả các sản phẩm đã bán



info -> 'items' trả về giá trị là JSON objects. Sau đó info->'items'->>'product' trả về tất cả all products như là text.

#### 5.4 Truy vấn dữ liệu JSON với mệnh đề WHERE

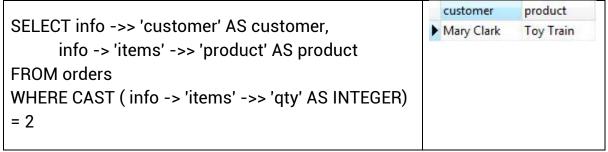
Chúng ta có thể sử dụng các toán tử JSON trong mệnh đề WHERE để lọc các hàng trả về. Ví dụ: để biết ai đã mua Diaper, chúng tôi sử dụng truy vấn sau

```
SELECT info ->> 'customer' AS customer

FROM orders

WHERE info -> 'items' ->> 'product' = 'Diaper';
```

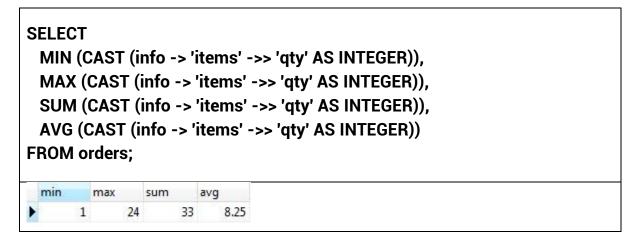
Để biết ai đã mua hai sản phẩm cùng một lúc, chúng tôi sử dụng truy vấn sau:



Lưu ý rằng chúng tôi đã sử dụng kiểu ép kiểu để chuyển đổi trường qty thành INTEGER gõ và so sánh nó với hai

#### 5.5 Áp dụng các hàm tổng hợp cho dữ liệu JSON

Chúng tôi có thể áp dụng các hàm tổng hợp như MIN, MAX, AVERAGE, SUM, v.v., cho dữ liệu JSON. Ví dụ: câu lệnh sau trả về số lượng tối thiểu, số lượng tối đa, số lượng trung bình và tổng số lượng sản phẩm đã bán.



#### 5.6 PostgreSQL JSON functions

PostgreSQL cung cấp cho chúng tôi một số chức năng để giúp bạn xử lý dữ liệu JSON.

#### A. Json\_each Function

Hàm json\_each () cho phép chúng ta mở rộng đối tượng JSON ngoài cùng thành a set of key-value pairs.

```
SELECT json_each (info)
FROM orders;

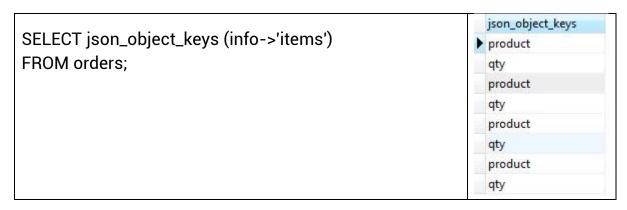
json_each
(customer,"""John Doe""")
(items,"{""product"": ""Beer"",""qty"": 6}")
(customer,"""Lily Bush""")
(items,"{""product"": ""Diaper"",""qty"": 24}")
(customer,"""Josh William""")
(items,"{""product"": ""Toy Car"",""qty"": 1}")
(customer,"""Mary Clark""")
(items,"{""product"": ""Toy Train"",""qty"": 2}")
```

Nếu bạn muốn nhận một tập hợp các cặp key - value dưới dạng văn bản, bạn sử dụng hàm **json\_each\_text** () để thay thế.

#### B. Json\_object\_keys Function

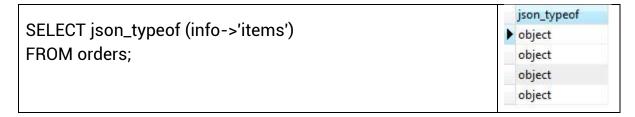
Để lấy một bộ khóa trong đối tượng JSON ngoài cùng, bạn sử dụng hàm json\_object\_keys ()

Truy vấn sau đây trả về tất cả các khóa của đối tượng mục lồng nhau trong cột thông tin.

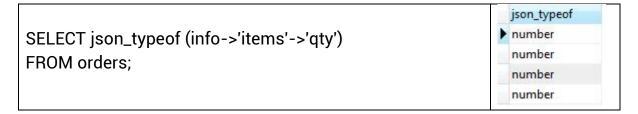


#### C. Json\_typeof Function

The **json\_typeof()** function returns type of the outermost JSON value as a string. It can be number, boolean, null, object, array, and string.



The following query returns the data type of the qty field of the nested items JSON object.



Xem thêm nhiều function ở đây

https://www.postgresql.org/docs/current/functions-json.html