# 

[**Tuần 2: Kiến thức cơ bản về SQL 1**](#_jlf1901uew7v)

[**Ngày 1: Các dạng chuẩn hóa database 1**](#_3g1pduxtebj)

[I. Mục đích 1](#_k9n8sy3l454)

[II. Phụ thuộc 1](#_lwtiqkufd285)

[1. Phụ thuộc hàm 1](#_cm01oaxx0vm8)

[2. Phụ thuộc đầy đủ, bộ phận 2](#_e839cm38ueoe)

[3. Phụ thuộc bắc cầu 2](#_vyp3o7ji94wd)

[III. Các dạng chuẩn hóa 2](#_4ck4v59to5sj)

[1. Chuẩn hóa 1NF 2](#_o8inqt39dmmu)

[2. Chuẩn hóa 2NF 3](#_tr43tpucy8go)

[3. Chuẩn hóa 3NF 4](#_wc0sd2z388q1)

[4. Dạng chuẩn Boyce-Codd 4](#_j1xie5s4g4mq)

[**Ngày 2: Truy vấn cơ bản 5**](#_4p73ol7fmakv)

[I. Cấu trúc lệnh SQL: SELECT, INSERT, UPDATE, DELETE 5](#_b0dk3crmerc9)

[1. SELECT 5](#_w4fuj8ubsnsq)

[2. INSERT 7](#_kpjiawc6497f)

[3. UPDATE 8](#_6o7xg1tzfyrz)

[4. DELETE 9](#_gr5iszwhzwhm)

[II. Các kiểu dữ liệu trong SQL 10](#_3r4ibd7rdq1d)

[1. Boolean 10](#_6bm6o1scq3fc)

[2. Character 11](#_vm40fp9nl5x)

[3. Numeric: 2 loại số nguyên và số thực 12](#_8cncxymrfcah)

[4. Kiểu dữ liệu thời gian: 5 kiểu chính 13](#_vpawlrx5r4sz)

[5. Array 21](#_y22qjl4rca2w)

[6. JSON: 2 kiểu JSON và JSONB để lưu trữ dữ liệu JSON. 23](#_l37r7xf7bh4k)

[7. Mã UUID 25](#_gwggx62l8lag)

[8. Các kiểu dữ liệu đặc biệt 25](#_ue9a92abddec)

[III. Sử dụng điều kiện WHERE để lọc dữ liệu 26](#_comvdpk9pthc)

[IV. Sử dụng các toán tử (AND, OR, NOT, BETWEEN, IN, ALL, LIKE) 30](#_7id3vnqym0uh)

[1. Toán tử AND 30](#_nrp8mavuwohn)

[2. Toán tử OR 31](#_pju72ordy8c7)

[3. Toán tử NOT 31](#_ncnxv4enxxnb)

[4. Toán tử BETWEEN 33](#_7w8wt87tvq4a)

[5. Toán tử IN 34](#_xo3bbrx16i0g)

[6. Toán tử ALL 35](#_72872jhj6ng6)

[7. Toán tử LIKE 36](#_611if6j495c8)

[V. Câu lệnh JOIN (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN) 38](#_j49z1lzgqiev)

[1. INNER JOIN 38](#_y4t45k2z7nsv)

[2. LEFT JOIN 39](#_plzrna3pijd8)

[3. RIGHT JOIN 40](#_4dkqc0e20863)

[4. FULL JOIN 41](#_eudwqlb2vlob)

[**Ngày 3: Group By, Order By 42**](#_vz70supxy7cu)

[I. Aggregate functions: COUNT, SUM, AVG, MIN, MAX 42](#_j9g15uek362)

[1. COUNT 42](#_4xk12v7y885s)

[2. SUM 44](#_j85gn1p4ii4x)

[3. AVG 46](#_2x541zxjubra)

[4. MIN 48](#_esvplv6yiga9)

[5. MAX 49](#_lwqaeghywk8m)

[II. GROUP BY 51](#_lzc5r9at4uiq)

[III. ORDER BY 53](#_bt5bvvcirhts)

[IV. CASE WHEN 56](#_gdoonc4ono19)

[V. Union và Union All 58](#_z205l23mzts5)

[1. Union 58](#_q8xcdi567zqo)

[2. Union All 58](#_wxd0ns6t0x34)

[VI. Subqueries 59](#_xnqo6x3b1pkt)

[**Ngày 4: Function Date, Function String 61**](#_xcp8dj5l4aw2)

[I. Function Date 61](#_rdcnrtubbasy)

[1. CURRENT\_DATE 61](#_bu6f9xyjsb5z)

[2. CURRENT\_TIME 62](#_691t4pjt25v0)

[3. CURRENT\_TIMESTAMP 63](#_q77fnxgyz7np)

[4. CLOCK\_TIMESTAMP() 63](#_u5io7g7zqv4q)

[5. STATEMENT\_TIMESTAMP 64](#_mgzrq6p7to8)

[6. NOW 65](#_culrt746nk7b)

[7. LOCALTIME 65](#_m2m6vnjxymqw)

[8. LOCALTIMESTAMP 65](#_mr59lfx34m0h)

[9. DATE\_PART 66](#_r5kqv71yjkyj)

[10. Extract 69](#_dn41yzqprnmf)

[11. TO\_DATE 72](#_2mjzbbnsqmma)

[12. TO\_TIMESTAMP 75](#_7kkcjxsklv33)

[13. MAKE\_DATE 75](#_7lvy67z8mgef)

[14. MAKE\_TIME 76](#_ygn54ss3qo1)

[15. AGE 76](#_y7z5eu3f8i09)

[16. JUSTIFY\_DAYS 76](#_2ns0o31qc78)

[17. JUSTIFY\_HOURS 77](#_wq1m50kmjczt)

[21. DATE\_TRUNC 80](#_lh14u8ug3yss)

[22. ISFINITE 80](#_riycjcmzmre2)

[23. TIMEOFDAY 81](#_s7amj1v09gxh)

[24. PG\_SLEEP 81](#_312va4yyfh9l)

[II. Function String 82](#_pzlpaquwy3dy)

# 

# **Tuần 2: Kiến thức cơ bản về SQL**

## **Ngày 1: Các dạng chuẩn hóa database**

### **Mục đích**

* Trong hầu hết các ứng dụng web và mobile đều cần một bộ cơ sở dữ liệu để xử lý, lưu trữ, kết xuất dữ liệu. Các chức năng của ứng dụng động đều cần phải thao tác với cơ sở dữ liệu. Vì thế, nếu thiết kế cơ sở dữ liệu không đúng chuẩn, có thể gây ra lãng phí không gian tài nguyên lưu trữ của hệ thống, hơn nữa còn gây ảnh hưởng nghiêm trọng đến tốc độ của ứng dụng.

### **Phụ thuộc**

#### **Phụ thuộc hàm**

* Trong một quan hệ, nếu giá trị bộ thuộc tính của A có thể suy ra được từ giá trị bộ thuộc tính của B, thì A là phụ thuộc hàm của B.

#### **Phụ thuộc đầy đủ, bộ phận**

* Trong một quan hệ, nếu bộ thuộc tính của A phụ thuộc hàm vào bộ thuộc tính của B, nếu loại bỏ thuộc tính trong bộ thuộc tính của B mà A vẫn phụ thuộc hàm vào B, thì A phụ thuộc đầy đủ vào B. Phụ thuộc bộ phận ngược lại.

#### **Phụ thuộc bắc cầu**

* Trong một quan hệ, nếu thuộc tính của A phụ thuộc hàm vào thuộc tính của B, thuộc tính của B phụ thuộc hàm vào thuộc tính của C, thì thuộc tính của A phụ thuộc hàm vào thuộc tính của C qua B.

### **Các dạng chuẩn hóa**

#### **Chuẩn hóa 1NF**

* Miền giá trị của mỗi thuộc tính chỉ chứa giá trị đơn nguyên tử, không thể phân tách ra được.
* Giá trị của mỗi thuộc tính trong mỗi bộ phải là giá trị đơn.
* Cách chuẩn hóa:
* Tách các thuộc tính có miền giá trị đa nguyên tử thành các thuộc tính có miền giá trị đơn nguyên tử.
  + Tách các bảng chứa giá trị đa trị thành bảng riêng.

VD: bảng Student có thuộc tính có miền giá trị đa nguyên tử, Hovaten bao gồm Họ và tên

| Hovaten | Lớp | Điểm |
| --- | --- | --- |

thành

| Họ | Tên | Lớp | Điểm |
| --- | --- | --- | --- |

VD:

* Bảng sự kiện

| Tensukien | Loai | Thoigian |
| --- | --- | --- |
| Beach | 1,2 | 2020-12-3 2020-12-6 |

* Bảng loại sự kiện

| Loai | Ten |
| --- | --- |
| 1 | Holiday |
| 2 | Festival |

* Bảng sự kiện

| Tensukien | Loai | Thoigian |
| --- | --- | --- |
| Beach | 1 | 2020-12-3 |
| Beach | 2 | 2020-12-6 |

#### **Chuẩn hóa 2NF**

* Phải thỏa mãn chuẩn 1NF.
* Mọi thuộc tính không khóa của quan hệ phụ thuộc hàm đầy đủ vào khóa chính.
* Cách chuẩn hóa:
* Tách các thuộc tính không khóa phụ thuộc bộ phận khóa chính thành quan hệ riêng, khóa của quan hệ mới là khóa bộ phận tương ứng ban đầu.
* Loại bỏ thuộc tính không khóa ra khỏi quan hệ gốc.

VD:

bảng MuonSach:

| SoThe | MaSach | Ten | TenSach | NgayMuon |
| --- | --- | --- | --- | --- |

thành

Bảng Nguoi:

| SoThe | Ten |
| --- | --- |

Bảng Sach:

| MaSach | TenSach |
| --- | --- |

Bảng MuonSach

| SoThe | MaSach | NgayMuon |
| --- | --- | --- |

#### **Chuẩn hóa 3NF**

* Phải thỏa mãn chuẩn 2NF.
* Không có thuộc tính không khóa phụ nào phụ thuộc bắc cầu vào khóa chính.
* Cách chuẩn hóa:
* Tách quan hệ mới gồm các thuộc tính phụ thuộc bắc cầu và thuộc tính không khóa mà nó phụ thuộc vào.
* Loại bỏ các thuộc tính phụ thuộc bắc cầu vào khóa chính trong quan hệ ban đầu

VD: có bảng NhanVien có cột TenDV phụ thuộc bắc cầu vào khóa chính thông qua thuộc tính không khóa MaDV

| MaNV | HoTen | MaDV | TenDV |
| --- | --- | --- | --- |

thành

| MaNV | HoTen |
| --- | --- |

| MaDV | TenDV |
| --- | --- |

#### **Dạng chuẩn Boyce-Codd**

* Quan hệ được gọi là chuẩn Boyce-Codd (BCNF) nếu:
* Thỏa mãn dạng chuẩn 3NF.
* Không có thuộc tính khóa phụ thuộc hàm vào thuộc tính không khóa.
* Cách chuẩn hóa:
* Tách các thuộc tính không khóa và thuộc tính khóa phụ thuộc hàm vào thành quan hệ mới, thuộc tính không khóa đó trở thành khóa trong quan hệ mới.
* Loại bỏ thuộc tính khóa ở bước 1 khỏi lược đồ
* Bổ sung các thuộc tính không khóa mà thuộc tính khóa, phụ thuộc vào khóa của quan hệ gốc.

VD: bảng Student

| Stu\_Id | Stu\_Branch | Stu\_Course | Branch\_Number | Stu\_Course\_No |
| --- | --- | --- | --- | --- |

thành

| Stu\_Id | Stu\_Branch |
| --- | --- |

| Stu\_Course | Branch\_Number | Stu\_Course\_No |
| --- | --- | --- |

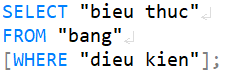
| Stu\_Id | Stu\_Course\_No |
| --- | --- |

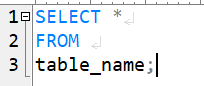
## **Ngày 2: Truy vấn cơ bản**

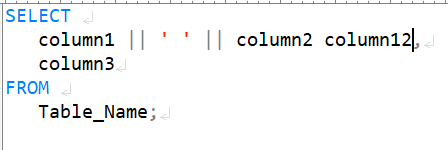
### **Cấu trúc lệnh SQL: SELECT, INSERT, UPDATE, DELETE**

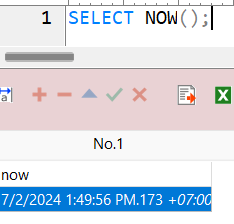
#### **SELECT**

* Cú pháp:



* Để truy xuất tất cả cột từ bảng, **SELECT** được sử dụng như sau (cách này không nên sử dụng nhiều, nên chỉ định rõ ràng tên cột trong mệnh đề bất cứ khi nào có thể):
* Hiệu suất cơ sở dữ liệu. Giả sử có một bảng với nhiều cột và dữ liệu đáng kể, câu **SELECT** có dấu hoa thị ( \*) viết tắt sẽ chọn dữ liệu từ tất cả các cột của bảng, có khả năng truy xuất nhiều dữ liệu hơn mức cần thiết cho ứng dụng.
* Hiệu suất ứng dụng. Việc truy xuất dữ liệu không cần thiết từ cơ sở dữ liệu làm tăng lưu lượng giữa máy chủ PostgreSQL và máy chủ ứng dụng. Do đó, điều này có thể dẫn đến thời gian phản hồi chậm hơn và giảm khả năng mở rộng cho các ứng dụng.
* Có thể sử dụng câu lệnh **SELECT** với biểu thức, toán tử nối || để nối column1, khoảng trắng, column2 (có thể gán bí danh):

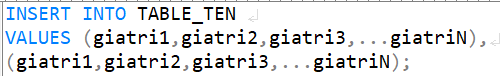
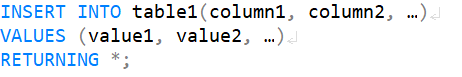
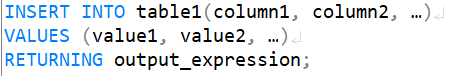


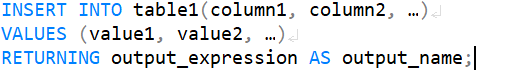
* Sử dụng **SELECT** mà không có mệnh đề **FROM:**
* Dùng **Now()** để trả về ngày và giờ hiện tại của máy chủ PostgreSQL.
* Lưu ý:
* Trong cơ sở dữ liệu tại thời gian thực, lệnh select không được khuyến khích sử dụng vì nó truy xuất dữ liệu nhiều hơn yêu cầu.
* Kết quả khiến ứng dụng chạy chậm.
* Trong trường hợp thêm nhiều cột hơn vào bảng, lệnh select truy xuất toàn bộ cột, bao gồm cả cột mới khiến ứng dụng bi treo.
* Dấu hoa thị (\*) trong mệnh đề SELECT chỉ nên dành riêng cho các truy vấn tùy ý kiểm tra dữ liệu từ cơ sở dữ liệu

#### **INSERT**

* Lệnh **INSERT INTO** được sử dụng để thêm các hàng dữ liệu mới vào một bảng trong Database.
* Có hai dạng cú pháp cho lệnh **INSERT INTO** trong SQL:

và

* Có thể chèn nhiều hơn 1 bản ghi 1 lúc bằng từ khóa VALUES: 
* Có thể dùng từ khóa **DEFAULT VALUES** để chèn bản ghi mới với giá trị mặc định cho các trường:
* Có thể sử dụng từ khóa **RETURNING** để tùy chọn trả về thông tin của hàng được chèn vào. Nếu muốn trả về toàn bộ hàng đã chèn, sử dụng dấu hoa thị \* sau **RETURNING**:
* Hoặc trả về một số thông tin của hàng được chèn, có thể chỉ định một hoặc nhiều cột sau mệnh đề:
* Có thể đổi tên giá trị trả về, sử dụng từ khóa **AS** ngay sau tên đầu ra:



* Chèn chuỗi giá trị có một dấu nháy đơn, sử dụng một dấu nháy đơn bổ sung để thoát chuỗi:

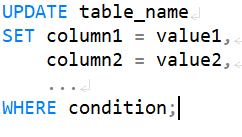


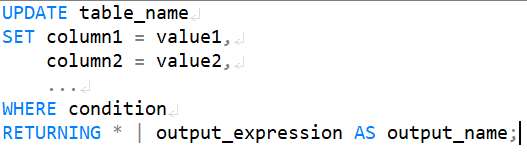
* Chèn giá trị ngày tháng vào cột có type là DATE, sử dụng định dạng YYYY-MM-DD:



* Lưu ý:
* Khi chèn các bản ghi bằng lệnh INSERT, phải cung cấp giá trị cho các cột **NOT NULL**.
* Có thể bỏ cột từ lệnh INSERT, nếu cột cho phép giá trị **NULL**.
* Danh sách cột và giá trị phải theo cùng thứ tự.
* Cung cấp danh sách các giá trị được phân cách bằng dấu phẩy trong ngoặc đơn.

#### **UPDATE**

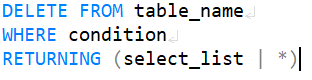
* Cho phép cập nhật dữ liệu trong một hoặc nhiều cột của một hoặc nhiều hàng trong một bảng.
* Cú pháp: 
* Có thể trả về các hàng đã cập nhật:



* Lưu ý:
* Mệnh đề này **WHERE** là tùy chọn, nếu bỏ qua **WHERE** mệnh đề này, **UPDATE** câu lệnh sẽ cập nhật tất cả các hàng trong bảng.
* Khi **UPDATE** câu lệnh được thực thi thành công, nó sẽ trả về thẻ lệnh sau: UPDATE count (count là số hàng được cập nhật bao gồm cả những hàng có giá trị không thay đổi).
* Sử dụng **UPDATE** để cập nhật dữ liệu trong một hoặc nhiều cột của bảng.
* Chỉ định một điều kiện trong mệnh đề **WHERE** để xác định hàng nào sẽ cập nhật dữ liệu.
* Sử dụng **RETURNING** để trả về các hàng đã cập nhật từ **UPDATE**.

#### **DELETE**

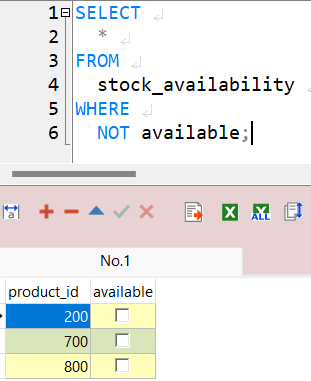
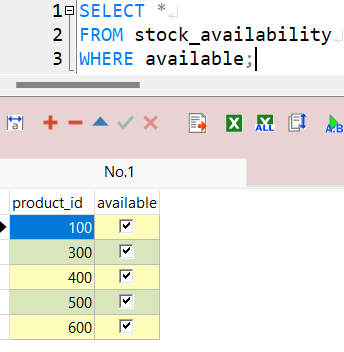
* Cho phép xóa một hoặc nhiều hàng khỏi một bảng.
* Cú pháp:
* Có thể trả về các hàng đã xóa:



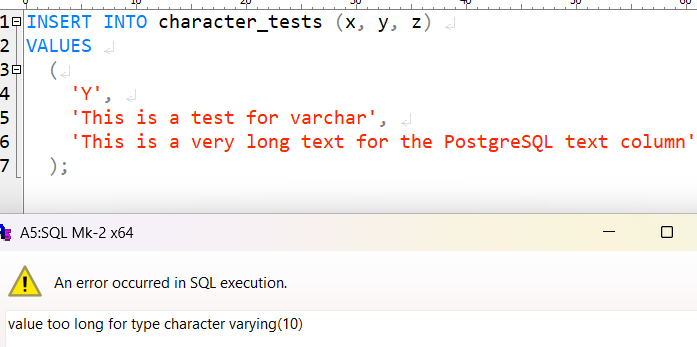
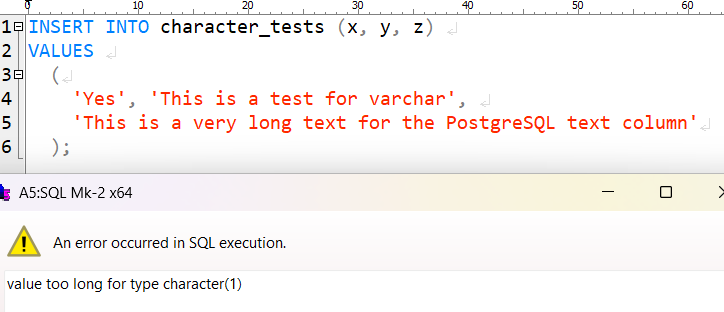
* Lưu ý:
* Mệnh đề này **WHERE** tùy chọn. Nếu bỏ qua mệnh đề **WHERE** này, câu lệnh **DELETE** sẽ xóa tất cả các hàng trong bảng.
* Sử dụng câu lệnh **DELETE FROM** để xóa một hoặc nhiều hàng khỏi bảng.
* Sử dụng mệnh đề **WHERE** này để chỉ định những hàng nào sẽ bị xóa.
* Sử dụng mệnh đề **RETURNING** này để trả về các hàng đã xóa.

### **Các kiểu dữ liệu trong SQL**

#### **Boolean**

* Kiểu dữ liệu Boolean có thể chứa một trong ba giá trị có thể: true, false hoặc null. Sử dụng từ khóa **boolean** hoặc **bool** để khai báo một cột có kiểu dữ liệu Boolean.
* Khi chèn dữ liệu vào một cột Boolean, PostgreSQL sẽ chuyển đổi nó thành một giá trị Boolean:
* 1, yes, y, t, true các giá trị được chuyển đổi thành **true**
* 0, no, false, f các giá trị được chuyển đổi thành **false**.
* Khi chọn dữ liệu từ một cột Boolean, PostgreSQL sẽ chuyển đổi các giá trị trở lại ví dụ t thành **true**, f thành **false** và space thành **null**.
* Sử dụng cột Boolean để truy vấn trả về tất cả các sản phẩm có sẵn:
* Có thể sử dụng toán tử **NOT** để kiểm tra xem các giá trị trong cột Boolean có trả về **FALSE**:
* Chú ý:
* Khi thêm một hàng mới, nếu giá trị của cột đó không được thêm thì mặc định sẽ là **FALSE** theo mặc định.
* Sử dụng kiểu dữ liệu PostgreSQL BOOLEAN để lưu trữ dữ liệu **boolean**.

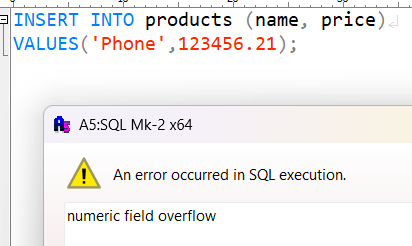
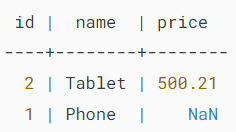
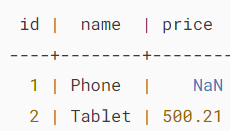
#### **Character**

* PostgreSQL cung cấp 3 kiểu dữ liệu ký tự: **CHAR(n), VARCHAR(n), TEXT**.
* **CHAR(n)** là ký tự có độ dài cố định với khoảng trắng được đệm. Nếu chèn một chuỗi ngắn hơn độ dài của cột, PostgreSQL sẽ đệm khoảng trắng. Nếu chèn một chuỗi dài hơn độ dài của cột, PostgreSQL sẽ báo lỗi.
* **VARCHAR(n)** là chuỗi ký tự có độ dài thay đổi. VARCHAR(n). Cho phép lưu trữ tối đa ký tự. PostgreSQL không chèn khoảng trắng khi chuỗi được lưu trữ ngắn hơn độ dài của cột.
* **TEXT** là chuỗi ký tự có độ dài thay đổi, dữ liệu văn bản là chuỗi ký tự có độ dài không giới hạn.
* Chú ý:
* Việc chỉ định độ dài cho **VARCHAR** kiểu dữ liệu là PostgreSQL sẽ báo lỗi nếu cố chèn một chuỗi có nhiều hơn ký tự vào **VARCHAR(n)** cột.
* Nếu không chỉ định số nguyên n cho kiểu dữ liệu **VARCHAR**, nó sẽ hoạt động giống như kiểu dữ liệu **TEXT**.
* Trong PostgreSQL, không có sự khác biệt về hiệu suất giữa ba kiểu ký tự.

#### **Numeric: 2 loại số nguyên và số thực**

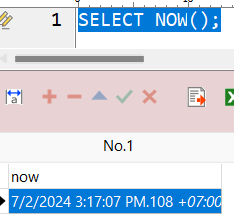
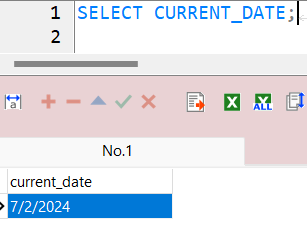
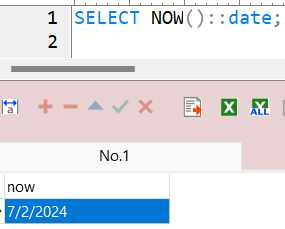
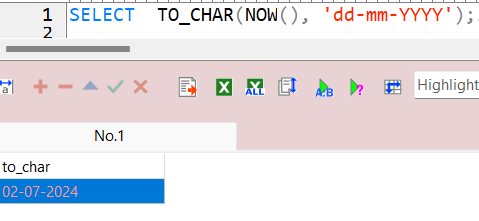
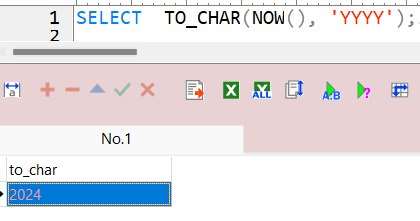
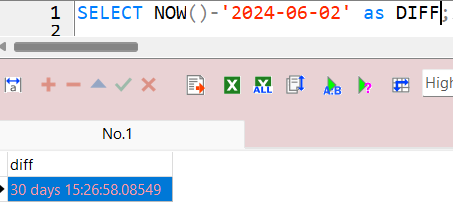
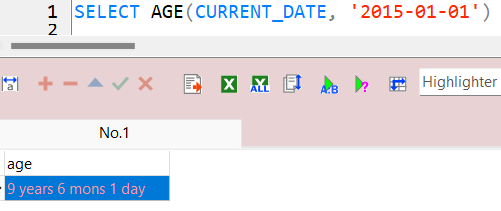
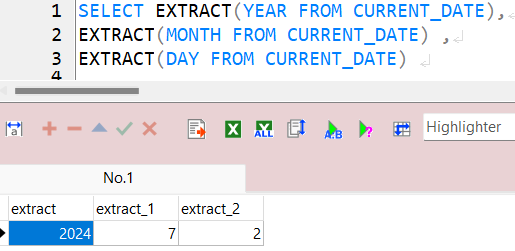
* **Cú pháp:**
* **precision** là tổng số chữ số.
* **scale** là số chữ số trong phần phân số.
* Các kiểu **NUMERIC** và **DECIMAL** từ đồng nghĩa nên có thể sử dụng chúng thay thế cho nhau.
* **Số nguyên có 3 loại:**
* **SMALLINT**: số nguyên có dấu 2 byte có phạm vi từ -32,768 đến 32,767.
* **INTEGER(INT):** số nguyên 4 byte có phạm vi từ -2.147.483.648 đến 2.147.483.647.
* **Serial** giống như integer ngoại trừ PostgreSQL sẽ tự động tạo và điền giá trị vào cột SERIAL. Điều này tương tự như AUTO\_INCREMENT column trong MySQL hoặc AUTOINCREMENT column trong SQLite.
* **Số thực có 3 loại chính:**
* **float(n)**: là số dấu phẩy động có độ chính xác ít nhất là n và tối đa là 8 byte.
* **real** hoặc **float8**: là số dấu phẩy động 4 byte.
* **numeric** hoặc **numeric(p,s)**: là một số thực có p chữ số với s số sau dấu phẩy. Đây **numeric(p,s)**là số chính xác.
* **Các giá trị đặc biệt:**
* Infinity
* -Infinity
* NaN

- Lưu ý:

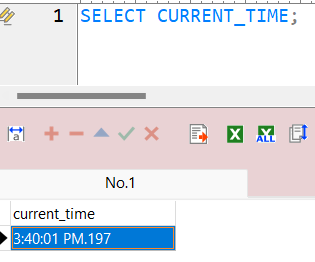
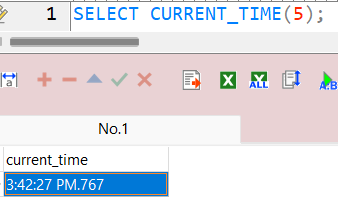
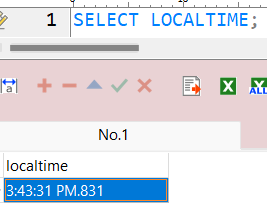
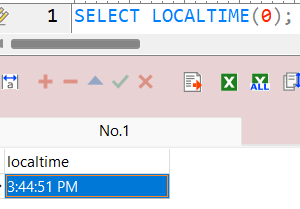
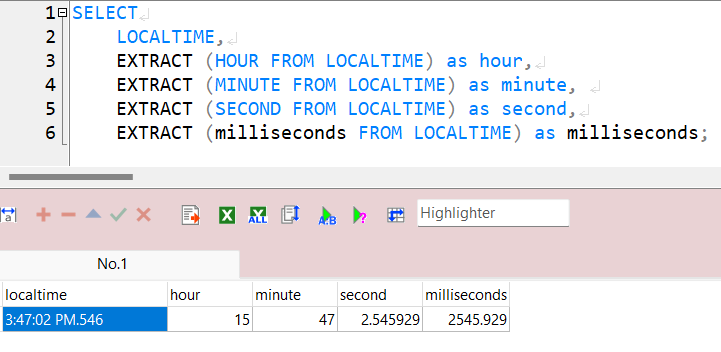
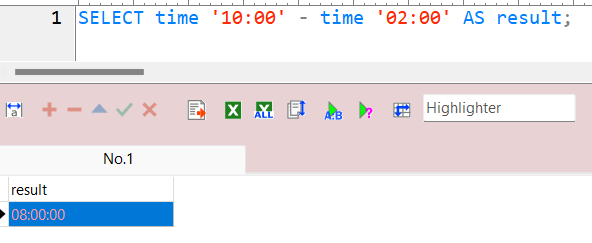
* Để lưu trữ các số yêu cầu độ chính xác như số tiền hoặc số lượng.
* Nếu lưu trữ một giá trị có độ chính xác vượt quá độ chính xác đã khai báo, PostgreSQL sẽ đưa ra lỗi.
* Kiểu dữ liệu **NUMERIC** này cũng có thể lưu trữ một giá trị đặc biệt được gọi là **NaN**.
* Các giá trị **NaN** lớn hơn các số thông thường như 1, 2, 3.

#### **Kiểu dữ liệu thời gian: 5 kiểu chính**

##### **DATE** dùng đểchỉ lưu trữ ngày tháng:

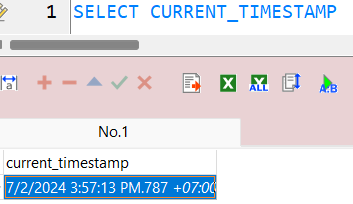
* Cú pháp:
* Các hàm của **DATE**:
* **NOW()**: Lấy về ngày giờ hiện tại.
* Kết quả có định dạng yyyy-mm-dd, có thể custom bằng **TO\_CHAR():**
* Để lấy khoảng thời gian giữa hai ngày, sử dụng - **toán tử trừ** ():
* Tính tuổi tại thời điểm hiện tại theo năm, tháng và ngày, sử dụng hàm **AGE()**:
* Trích xuất (**EXTRACT**) năm, quý, tháng, tuần và ngày từ giá trị ngày:

##### **TIME** lưu trữ giá trị thời gian trong ngày (yêu cầu 8 byte).

* Cú pháp: 
* kiểu dữ liệu TIME WITH TIME ZONE cho phép lưu trữ và xử lý thời gian trong ngày theo múi giờ.
* Các hàm của **TIME**:
* Để lấy thời gian hiện tại theo múi giờ, sử dụng **CURRENT\_TIME**:
* Để lấy thời gian hiện tại với độ chính xác cụ thể, sử dụng hàm **CURRENT\_TIME(precision)**:
* Để lấy giờ địa phương, sử dụng hàm **LOCALTIME**:
* Để lấy giờ địa phương với độ chính xác cụ thể, sử dụng hàm **LOCALTIME(precision)**:
* Chuyển đổi thời gian sang múi giờ khác:
* Trích xuất giờ, phút và giây từ giá trị thời gian:
* Các phép toán số học trên giá trị thời gian:

cộng thêm 2 giờ vào giờ địa phương:

##### **TIMESTAMP** lưu trữ cả giá trị ngày tháng và thời gian.

* Các hàm dấu thời gian của PostgreSQL:
* Để lấy dấu thời gian hiện tại có thể sử dụng hàm CURRENT\_TIMESTAMP:
* Để lấy thời gian trong ngày theo định dạng chuỗi, sử dụng hàm timeofday():

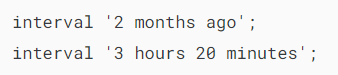
##### **TIMESTAMPTZ** là kiểu dữ liệu dấu thời gian nhận biết múi giờ. Đây là chữ viết tắt của dấu thời gian có múi giờ.

* Các hàm của TIMESTAMPTZ:
* Chuyển đổi giữa các múi giờ



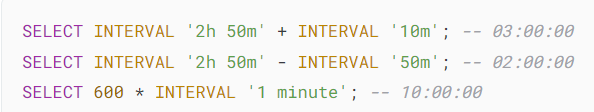
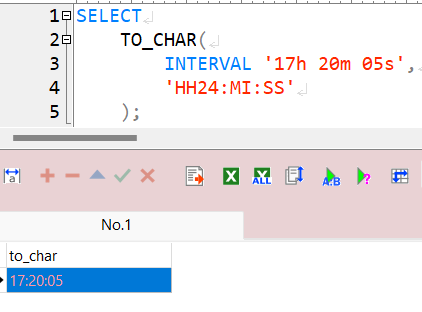
##### **INTERVAL** lưu trữ khoảng thời gian theo năm, tháng, ngày, giờ, phút và giây.

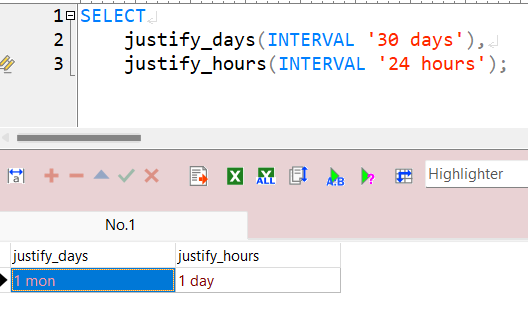
* Cú pháp:

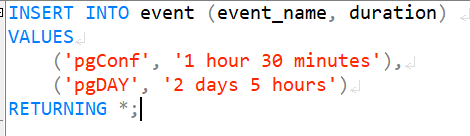
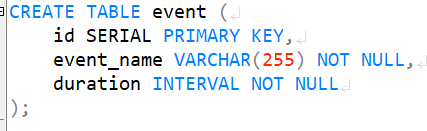
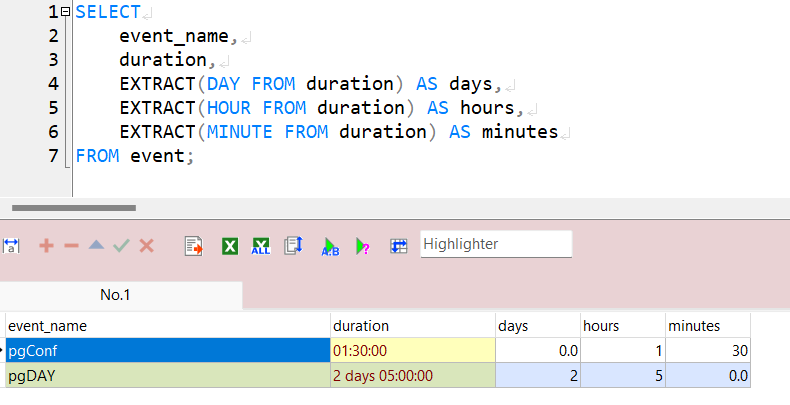


* **PostgreSQL** cung cấp bốn định dạng đầu ra:
* sql standard.
* postgres.
* postgres verbose.
* iso\_8601.



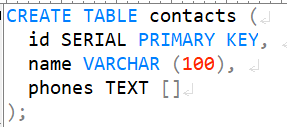
* Toán tử khoảng: 
* Chuyển đổi khoảng thời gian PostgreSQL thành chuỗi:
* Hai chức năng justifydays, justifyhours cho phép điều chỉnh khoảng thời gian 30 ngày thành một tháng và khoảng thời gian 24 giờ thành một ngày:

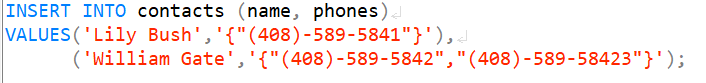
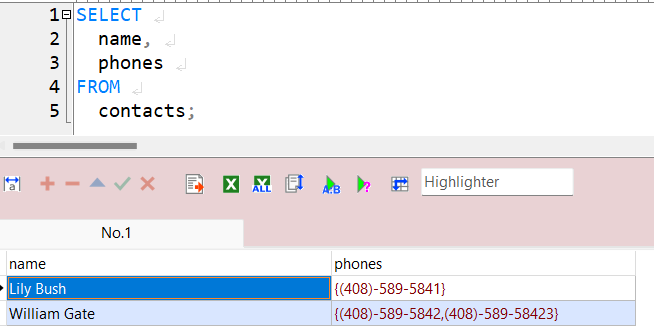


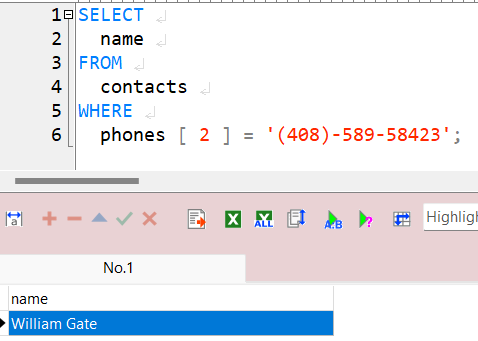
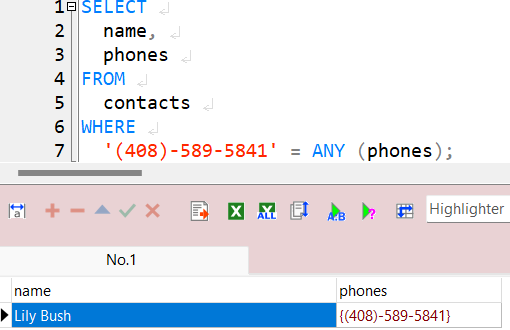
* Sử dụng kiểu **INTERVAL** trong bảng:
* Lưu ý:
* **timestamp**: dấu thời gian không có múi giờ.
* **timestamptz**: dấu thời gian có múi giờ.
* PostgreSQL lưu trữ các giá trị **timestamptz** trong cơ sở dữ liệu dưới dạng giá trị UTC.

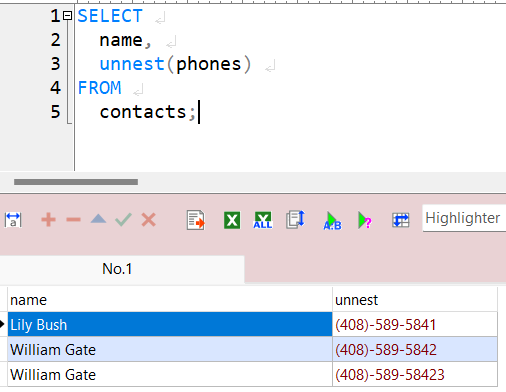
#### **Array**

* Trong PostgreSQL, có thể lưu trữ một chuỗi, một mảng số nguyên, v.v. trong các cột mảng.
* Mảng có thể là mảng một chiều, mảng nhiều chiều hoặc thậm chí là mảng lồng nhau.
* Cú pháp: 





* Có thể sử dụng phần tử mảng kết hợp với mệnh đề WHERE làm điều kiện để lọc các hàng.
* Các hàm có thể sử dụng trong mảng:
* Có thể sử dụng hàm **ANY()**:
* PostgreSQL cung cấphàm **unnest()** để mở rộng một mảng thành một danh sách các hàng.



* Lưu ý:
* Trong PostgreSQL, mảng là tập hợp các phần tử có cùng kiểu dữ liệu.
* Sử dụng **data\_type []**để xác định mảng một chiều cho một cột.
* Sử dụng cú pháp **[index]** để truy cập phần tử index của mảng. Phần tử đầu tiên có chỉ số là một.
* Sử dụng hàm **unnest()** này để mở rộng mảng thành danh sách các hàng.

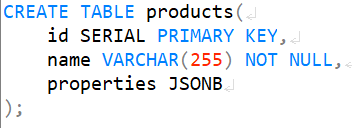
#### **JSON: 2 kiểu JSON và JSONB để lưu trữ dữ liệu JSON.**

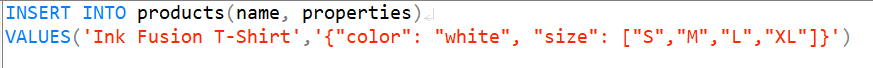
##### **JSON**

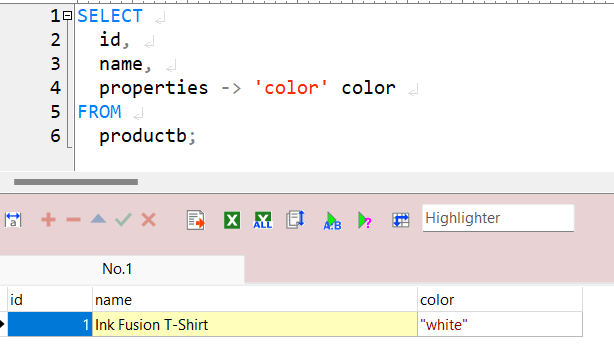
* Viết tắt của JavaScript Object Notation. **JSON** là định dạng trao đổi dữ liệu nhẹ, dễ đọc đối với con người và dễ phân tích đối với máy tính.
* Dựa trên hai cấu trúc dữ liệu chính là **đối tượng** và **mảng** :
* Một **đối tượng** được định nghĩa là một tập hợp không có thứ tự các cặp khóa-giá trị được bao trong dấu ngoặc nhọn {}.
* **Mảng** là danh sách có thứ tự các giá trị được đặt trong dấu ngoặc vuông []. Các giá trị không nhất thiết phải cùng loại. Ngoài ra, mảng có thể chứa các giá trị của bất kỳ loại dữ liệu **JSON** hợp lệ nào bao gồm **đối tượng** và **mảng**.
* **JSON** hỗ trợ một số kiểu dữ liệu bao gồm:
* Chuỗi: “Joe”
* Số: 100, 9.99, …
* Boolean: đúng và sai.
* Null null.

##### **JSONB** – lưu trữ dữ liệu **JSON** ở định dạng nhị phân.

| **Tính năng** | **JSON** | **JSONB** |
| --- | --- | --- |
| Kho | Biểu diễn văn bản (nguyên văn) | Định dạng lưu trữ nhị phân |
| Kích cỡ | Thông thường lớn hơn vì nó giữ lại khoảng trắng trong dữ liệu JSON | Thông thường nhỏ hơn |
| Lập chỉ mục | Chỉ mục tìm kiếm toàn văn | Chỉ số nhị phân |
| Hiệu suất | Hơi chậm hơn | Nói chung là nhanh hơn |
| Hiệu suất truy vấn | Chậm hơn do phân tích cú pháp | Nhanh hơn do lưu trữ nhị phân |
| Phân tích cú pháp | Phân tích mỗi lần | Phân tích cú pháp một lần, lưu trữ ở định dạng nhị phân |
| Thao tác dữ liệu | Đơn giản và dễ dàng | Phức tạp hơn |
| Sắp xếp chìa khóa | Bảo quản | Không được bảo quản |
| Các phím trùng lặp | Cho phép khóa trùng lặp, giá trị cuối cùng được giữ lại | Không cho phép sử dụng chìa khóa trùng lặp. |
| Trường hợp sử dụng | Lưu trữ dữ liệu cấu hình, dữ liệu nhật ký, tài liệu JSON đơn giản | Lưu trữ các tài liệu JSON khi cần truy vấn và lập chỉ mục nhanh |

* VD: 



* Lưu ý:
* Toán tử -> trích xuất một trường đối tượng **JSON** bằng một khóa.
* Toán tử ->> để trích xuất các phần tử đầu tiên trong mảng dưới dạng văn bản.
* **JSON** là viết tắt của **JavaScript Object Notation**.
* Sử dụng kiểu dữ liệu **JSONB** để lưu trữ dữ liệu **JSON**.

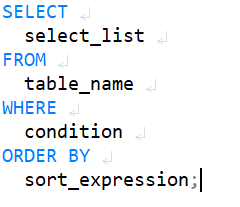
#### **Mã UUID**

* Kiểu dữ liệu này **UUID** cho phép lưu trữ Universal Unique Identifiers được định nghĩa bởi RFC 4122[.](https://tools.ietf.org/html/rfc4122) Các UUID giá trị đảm bảo tính duy nhất tốt hơn SERIAL và có thể được sử dụng để ẩn dữ liệu nhạy cảm được công khai như các giá trị id trong URL.

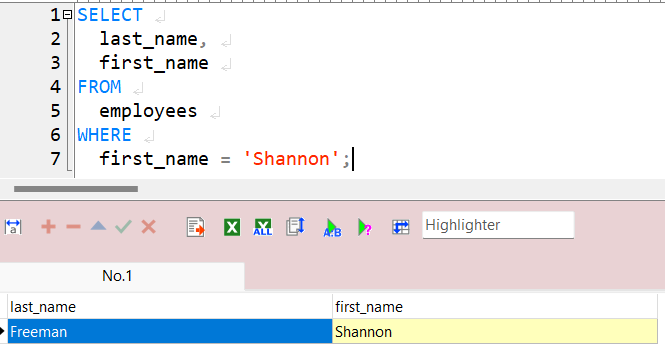
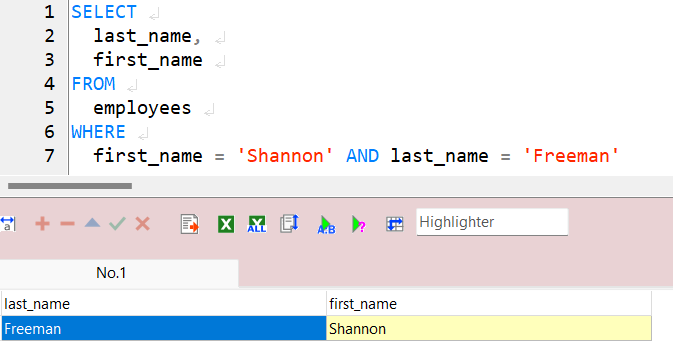
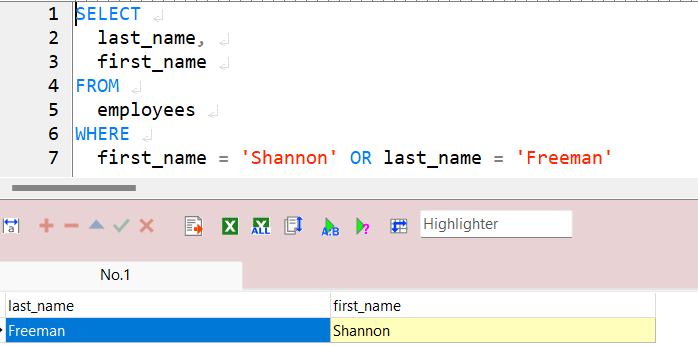
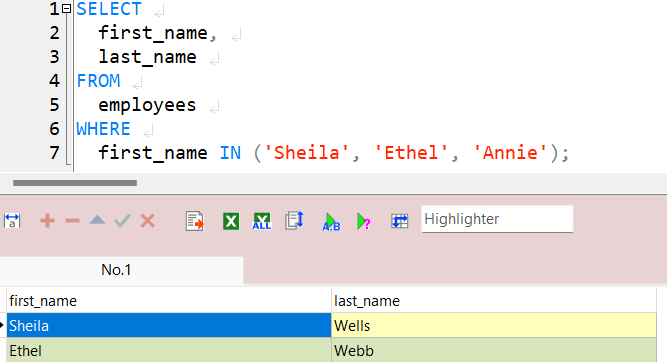
#### **Các kiểu dữ liệu đặc biệt**

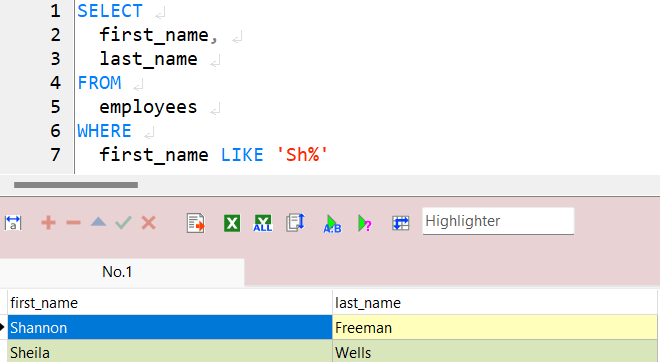
* Liên quan đến hình học và mạng.
* box– một hình hộp chữ nhật.
* line – một tập hợp các điểm.
* point– một cặp số hình học.
* lseg– một đoạn thẳng.
* polygon– hình học khép kín.
* inet– địa chỉ IP4.
* macaddr– địa chỉ MAC.

### **Sử dụng điều kiện WHERE để lọc dữ liệu**

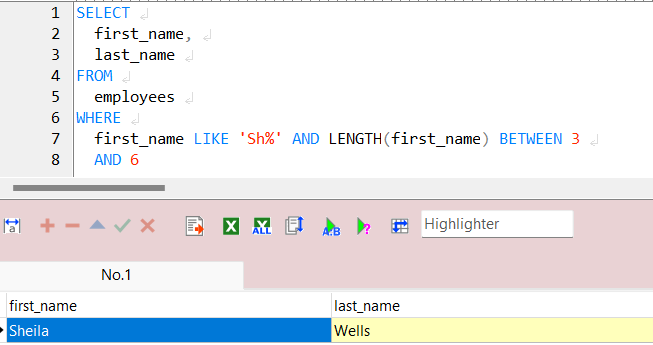
* Câu lệnh **SELECT** trả về tất cả các hàng từ một hoặc nhiều cột trong một bảng. Để truy xuất các hàng thỏa mãn một điều kiện đã chỉ định, sử dụng một mệnh đề **WHERE**.
* Cú pháp:
* Mệnh đề này **WHERE** sử dụng condition để lọc các hàng trả về từ mệnh đề **SELECT**.
* Để hình thành điều kiện trong mệnh đề **WHERE**, sử dụng toán tử so sánh và logic:

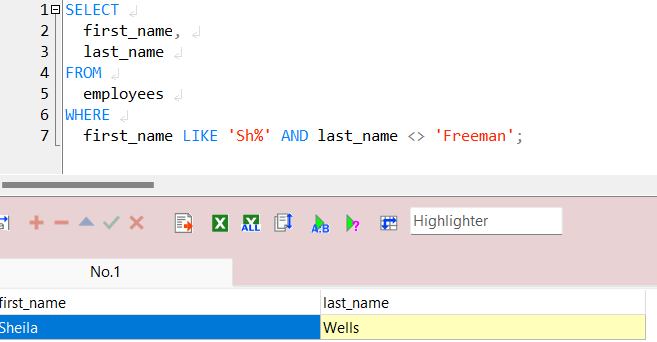
| **Operator** | **Description** |
| --- | --- |
| = | bằng |
| > | Lớn hơn |
| < | Ít hơn |
| >= | Lớn hơn hoặc bằng |
| <= | Nhỏ hơn hoặc bằng |
| <> hoặc != | Không bằng |
| AND | Toán tử logic AND |
| OR | Toán tử logic OR |
| IN | Trả về true nếu giá trị khớp với bất kỳ giá trị nào trong danh sách |
| BETWEEN | Trả về true nếu giá trị nằm trong phạm vi giá trị |
| LIKE | Trả về true nếu giá trị khớp với mẫu |
| IS NULL | Trả về true nếu giá trị là NULL |
| NOT | Phủ định kết quả của các toán tử khác |

* Sử dụng mệnh đề **WHERE** với toán tử **=**:
* Sử dụng mệnh đề **WHERE** với ví dụ toán tử **AND:**
* Sử dụng mệnh đề **WHERE** với ví dụ toán tử **OR:**
* Sử dụng mệnh đề **WHERE** với ví dụ toán tử **IN** để tìm một giá trị trong danh sách các giá trị:
* Sử dụng mệnh đề **WHERE** với ví dụ toán tử **LIKE** để tìm một chuỗi ký tự khớp với một mẫu đã chỉ định:



* Sử dụng mệnh đề **WHERE** với ví dụ toán tử **BETWEEN**



* Sử dụng mệnh đề **WHERE** với toán tử **không bằng (<>)** ví dụ
* Chú ý:
* Sử dụng mệnh đề **WHERE** trong câu lệnh **SELECT** đểlọc các hàng của truy vấn dựa trên một hoặc nhiều điều kiện.

### **Sử dụng các toán tử (AND, OR, NOT, BETWEEN, IN, ALL, LIKE)**

#### **Toán tử AND**

* Toán tử **AND** là toán tử logic kết hợp hai biểu thức boolean.
* Chú ý:
* Sử dụng toán tử **AND** để kết hợp nhiều biểu thức boolean.
* Cú pháp:
* Trong cú pháp này, expression1 và expression2 là các biểu thức **boolean** đánh giá là true, false, hoặc null.

| **AND** | **True** | **False** | **Null** |
| --- | --- | --- | --- |
| **True** | True | False | Null |
| **False** | False | False | False |
| **Null** | Null | False | Null |

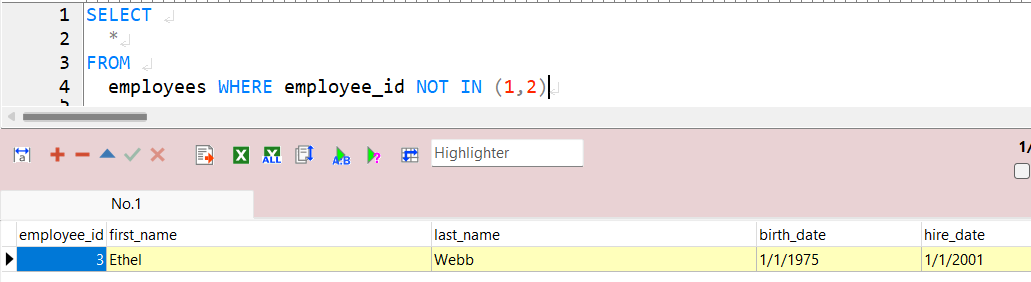
#### **Toán tử OR**

* Toán tử **OR** là toán tử logic kết hợp nhiều biểu thức **boolean**.
* Chú ý:
* Sử dụng toán tử **OR** để kết hợp nhiều biểu thức boolean.
* Cú pháp:
* Trong cú pháp này, expression1 và expression2 là các biểu thức boolean đánh giá là true, false, hoặc null.

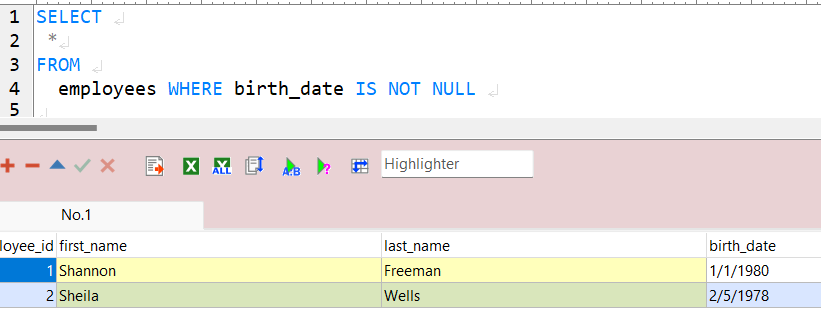
| **OR** | **True** | **False** | **Null** |
| --- | --- | --- | --- |
| **True** | True | True | True |
| **False** | True | False | Null |
| **Null** | True | Null | Null |

#### **Toán tử NOT**

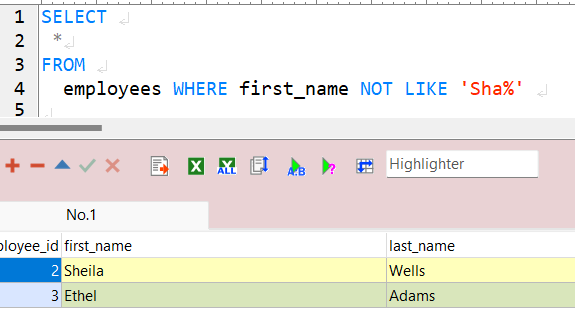
* Điều kiện **NOT** của PostgreSQL (còn gọi là Toán tử **NOT**) được sử dụng để phủ định một điều kiện trong câu lệnh **SELECT, INSERT, UPDATE** hoặc **DELETE**.
* Cú pháp:
* Có thể kết hợp với điều kiện **IN**

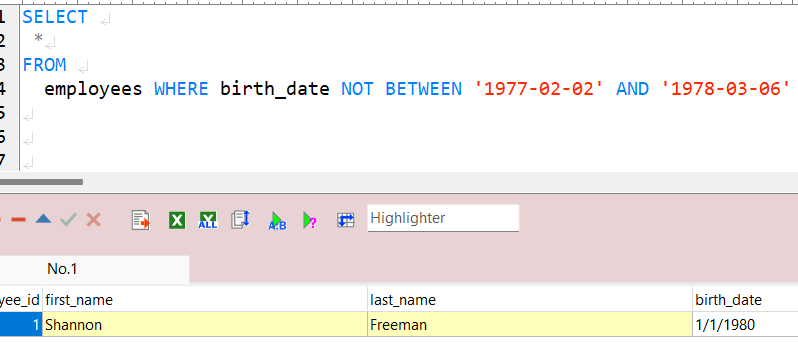


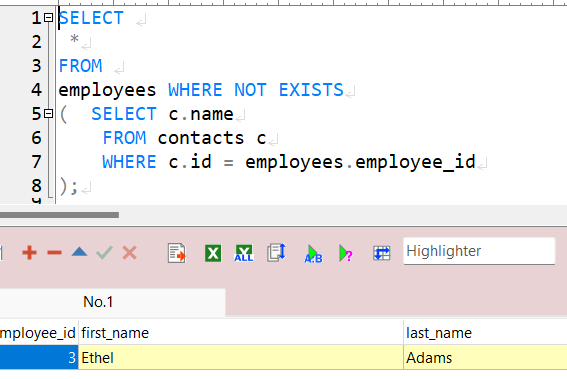
* Có thể kết hợp với điều kiện **IS NULL**



* Có thể kết hợp với điều kiện **LIKE**



* Có thể kết hợp với điều kiện **BETWEEN**
* Có thể kết hợp với điều kiện **EXISTS**

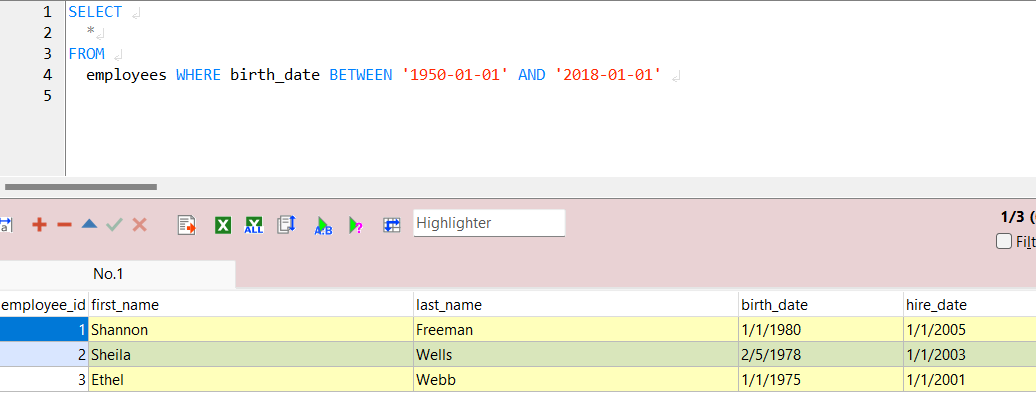


* Chú ý:
* Điều kiện **NOT** của PostgreSQL yêu cầu điều ngược lại của điều *kiện* phải được đáp ứng để bản ghi được đưa vào tập kết quả.

#### **Toán tử BETWEEN**

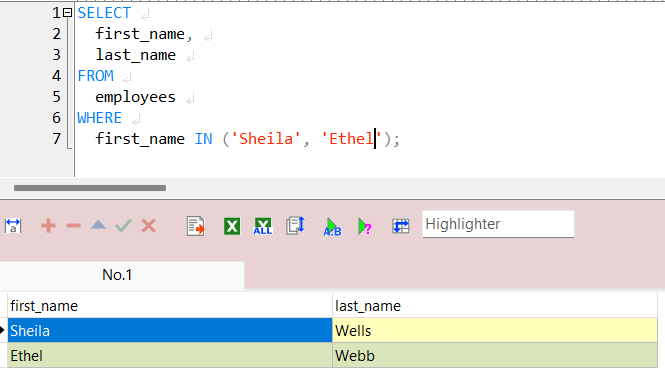
* Toán tử **BETWEEN** cho phép kiểm tra xem một giá trị có nằm trong một phạm vi giá trị hay không.
* Chú ý:
* Sử dụng toán tử **NOT BETWEEN** để phủ định toán tử **BETWEEN**.
* Cú pháp:



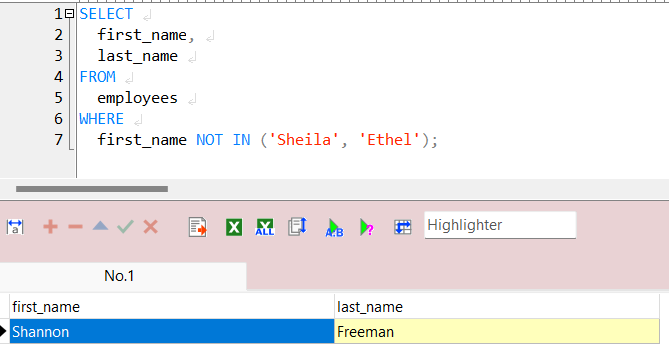


#### **Toán tử IN**

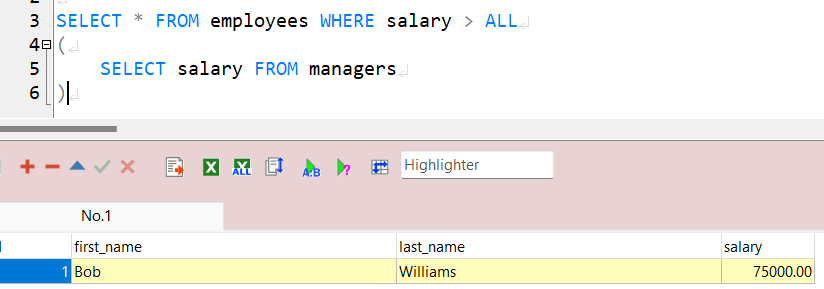
* Toán tử **IN** cho phép kiểm tra xem một giá trị có khớp với bất kỳ giá trị nào trong danh sách giá trị hay không.
* Chú ý:
* Sử dụng toán tử **IN** để kiểm tra xem một giá trị có khớp với bất kỳ giá trị nào trong danh sách các giá trị hay không.
* Sử dụng toán tử **NOT** để phủ định toán tử **IN**.
* Cú pháp:
* Toán tử **IN** trả về giá trị true nếu value bằng bất kỳ giá trị nào trong danh sách như value1 và value2.
* Về mặt chức năng, toán tử **IN** này tương đương với việc kết hợp nhiều biểu thức boolean với toán tử **OR**.



Sử dụng toán tử **NOT** để phủ định toán tử **IN**



#### **Toán tử ALL**

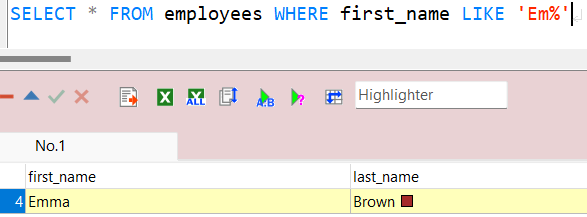
* Toán tử **ALL** cho phép so sánh một giá trị với tất cả các giá trị trong một tập hợp được trả về bởi một truy vấn phụ.
* Cú pháp:
* Toán tử **ALL** phải được đặt trước bởi **một toán tử so sánh** như bằng (=), không bằng (<>), lớn hơn (>), lớn hơn hoặc bằng (>=), nhỏ hơn (<), và nhỏ hơn hoặc bằng (<=).
* Toán tử **ALL** phải được theo sau bởi **một truy vấn phụ**, truy vấn phụ này cũng phải được bao quanh bởi dấu ngoặc đơn.
* Sử dụng toán tử **ALL** với **toán tử lớn hơn** (>)

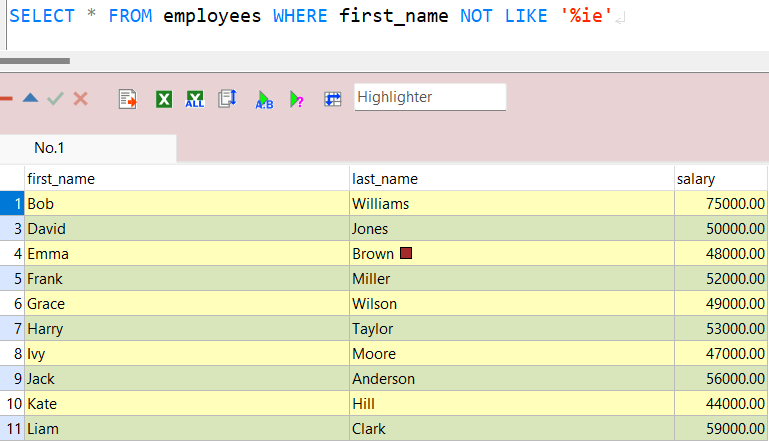
### 

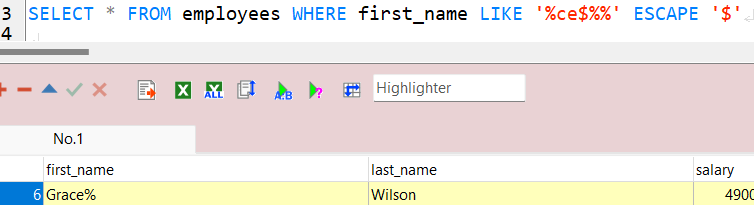
* Sử dụng toán tử **ALL** với **toán tử nhỏ hơn** (<)

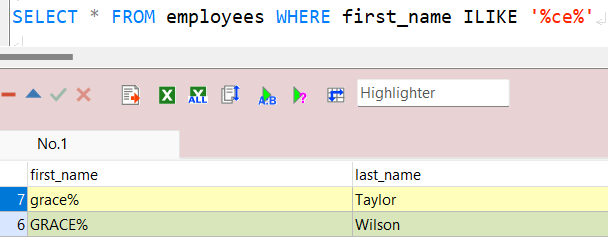


#### **Toán tử LIKE**

* Sử dụng để lọc các bản ghi theo một mẫu chỉ định.
* PostgreSQL cung cấp hai ký tự đại diện:
* **Dấu phần trăm** ( **%**) khớp với bất kỳ chuỗi nào có từ 0 ký tự trở lên.
* **Dấu gạch dưới** (**\_**) khớp với bất kỳ ký tự đơn nào.
* Cú pháp:
* Sử dụng toán tử **LIKE** để lọc các bản ghi có chuỗi ký tự bắt đầu là “Em”: 
* Sử dụng toán tử **NOT LIKE** để lọc các bản ghi không có chuỗi ký tự kết thúc là ”ie”:



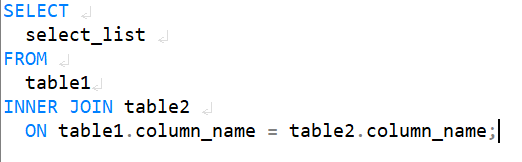
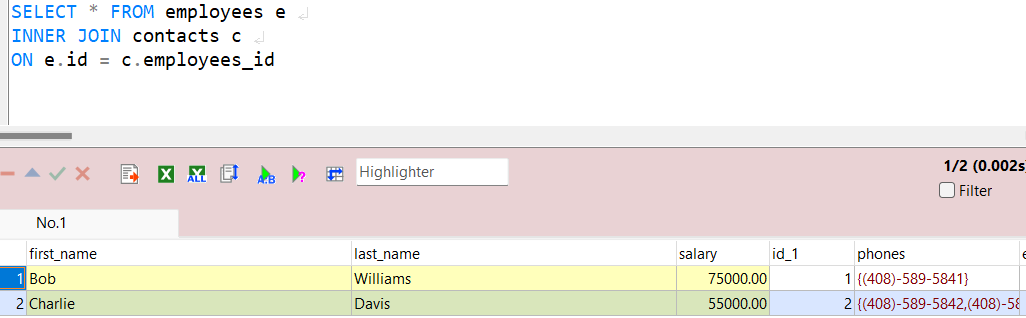
* Toán tử **LIKE** của PostgreSQL với tùy chọn **ESCAPE**:
* Toán tử **ILIKE** không phân biệt hoa thường:



* Chú ý:
* Sử dụng toán tử **LIKE** để khớp dữ liệu theo mẫu.
* Sử dụng toán tử **NOT LIKE** để phủ định toán tử **LIKE**.
* Sử dụng ký tự đại diện **%** để khớp với không hoặc nhiều ký tự.
* Sử dụng ký tự đại diện **\_** để khớp với một ký tự đơn.
* Sử dụng tùy chọn **ESCAPE** để chỉ định ký tự thoát.
* Sử dụng toán tử **ILIKE** để so khớp dữ liệu không phân biệt chữ hoa chữ thường.

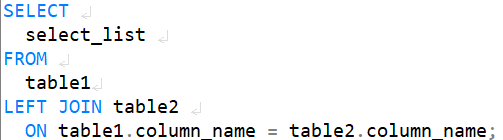
### **Câu lệnh JOIN (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN)**

#### **INNER JOIN**

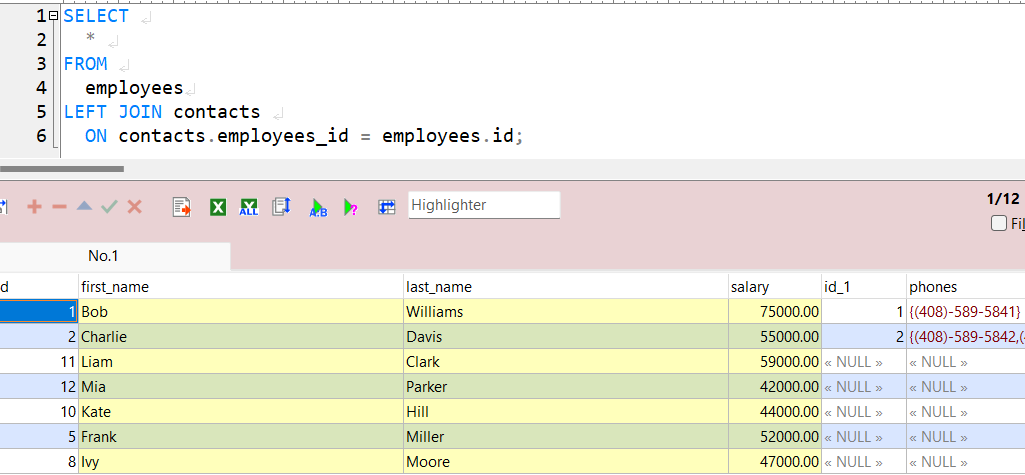
* Sử dụng mệnh đề **INNER JOIN** để chọn dữ liệu từ hai hoặc nhiều bảng có liên quan và trả về các hàng có giá trị trùng khớp trong tất cả các bảng.
* Cú pháp:
* Đầu tiên, chỉ định các cột từ cả hai bảng trong danh sách chọn của mệnh đề **SELECT**.
* Thứ hai, chỉ định bảng chính ( table1) mà muốn chọn dữ liệu trong mệnh đề **FROM**.
* Thứ ba, chỉ định bảng thứ hai ( table2) mà muốn tham gia bằng cách sử dụng từ khóa **INNER JOIN**.
* Cuối cùng, xác định điều kiện cho phép nối. Điều kiện này chỉ ra cột nào (column\_name) trong mỗi bảng phải có giá trị khớp với phép nối.
* Câu lệnh sau sử dụng mệnh đề này **INNER JOIN** để chọn dữ liệu **trùng khớp** từ cả hai bảng:
* Lưu ý:
* Để làm cho truy vấn ngắn hơn, có thể sử dụng **bí danh** bảng.
* Có thể sử dụng **USING** nối khóa chính và khóa ngoại có tên trùng nhau.

#### **LEFT JOIN**

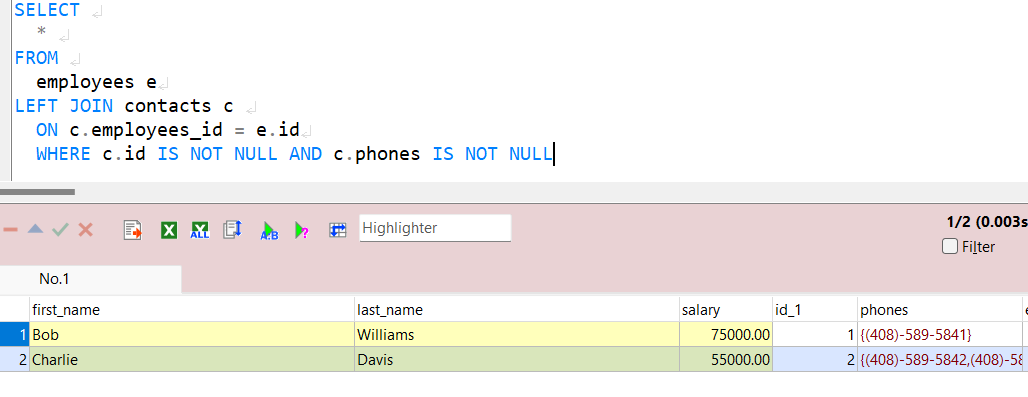
* Mệnh đề này nối bảng bên trái với bảng bên phải và trả về các hàng từ bảng bên trái có thể có hoặc không có các hàng tương ứng trong bảng bên phải.
* Cú pháp:



* Đầu tiên, chỉ định các cột từ cả hai bảng trong danh sách chọn ( select\_list) của mệnh đề **SELECT**.
* Thứ hai, chỉ định bảng bên trái ( table1) mà muốn chọn dữ liệu trong mệnh đề **FROM**.
* Thứ ba, chỉ định đúng bảng ( table2) mà muốn tham gia bằng cách sử dụng từ khóa **LEFT JOIN**.
* Cuối cùng, hãy xác định điều kiện cho phép nối ( table1.column\_name = table2.column\_name), điều kiện này chỉ ra rằng cột ( column\_name) trong mỗi bảng phải có giá trị khớp nhau.
* Sử dụng **LEFT JOIN** để lấy các bản ghi từ bảng bên trái employees và các bản ghi trùng khớp từ bên phải (không trùng khớp trả về giá trị **NULL**):



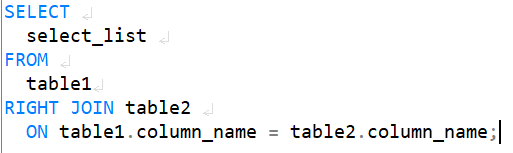
* Có thể kết hợp với mệnh đề **WHERE** để lọc các bản ghi không trùng khớp từ bên phải:



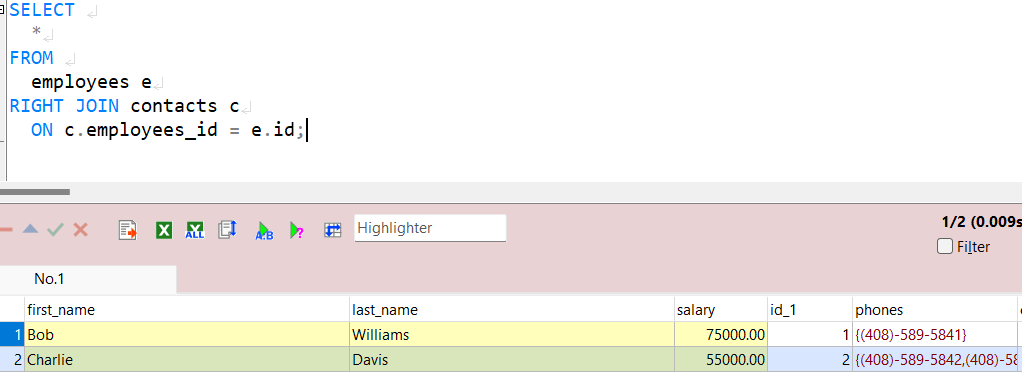
* Chú ý:
* Nếu các cột để nối hai bảng có cùng tên, bạn có thể sử dụng cú pháp **USING.**
* Sử dụng mệnh đề PostgreSQL **LEFT JOIN** để chọn các hàng từ một bảng có thể có hoặc không có các hàng tương ứng trong các bảng khác.
* **LEFT JOIN c**ònđược gọi là **LEFT OUTER JOIN.**

#### **RIGHT JOIN**

* Mệnh đề này nối một bảng bên phải với một bảng bên trái và trả về các hàng từ bảng bên phải có thể có hoặc không có các hàng khớp với nhau trong bảng bên trái.
* Cú pháp:

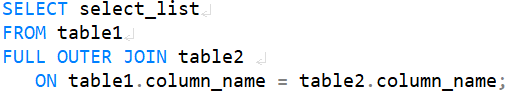


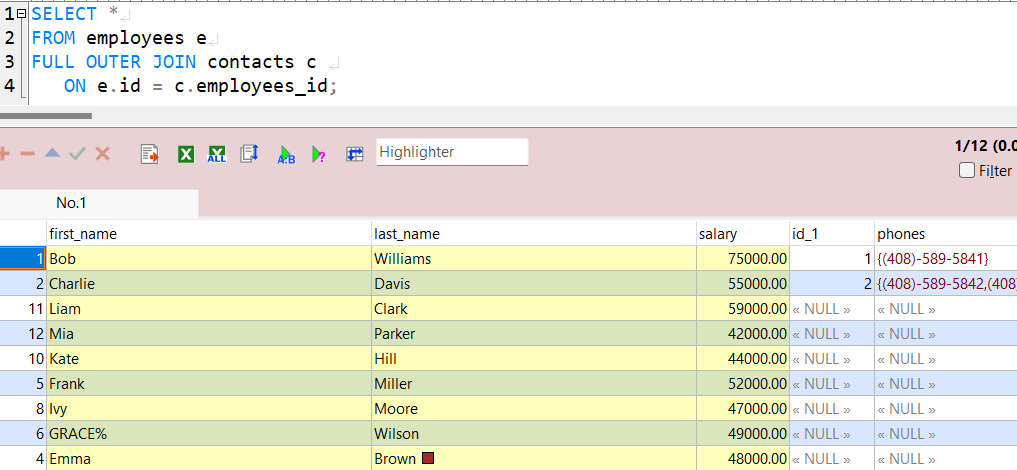
* Đầu tiên, chỉ định các cột từ cả hai bảng trong select\_list mệnh đề **SELECT**.
* Thứ hai, cung cấp bảng bên trái ( table1) mà muốn chọn dữ liệu trong mệnh đề **FROM**.
* Thứ ba, chỉ định bảng bên phải ( table2) mà muốn nối với bảng bên trái trong mệnh đề **RIGHT JOIN**.
* Cuối cùng, hãy xác định điều kiện để nối hai bảng ( table1.column\_name = table2.column\_name), điều kiện này chỉ ra rằng column\_nametrong mỗi bảng phải có các hàng khớp nhau.
* Sử dụng **RIGHT JOIN** để lấy các bản ghi từ bảng bên phải Contacts và các bản ghi trùng khớp từ bên trái (không trùng khớp trả về giá trị **NULL**):



* Chú ý:
* Sử dụng mệnh đề PostgreSQL **RIGHT JOIN** để nối một bảng bên phải với một bảng bên trái và trả về các hàng từ bảng bên phải có thể có hoặc không có các hàng tương ứng trong bảng bên trái.
* **RIGHT JOIN c**ònđược gọi là **RIGHT OUTER JOIN.**

#### **FULL JOIN**

* Kết hợp **FULL OUTER JOIN** dữ liệu từ hai bảng và trả về tất cả các hàng từ cả hai bảng, bao gồm các hàng khớp và không khớp từ cả hai bên.
* Cú pháp:
* Đầu tiên, chỉ định các cột từ table1 và table2 trong select\_list.
* Thứ hai, chỉ rõ dữ liệu từ table1 muốn lấy trong mệnh đề **FROM**.
* Thứ ba, chỉ rõ dữ liệu từ table 2 mà muốn nối với table1 trong mệnh đề **FULL OUTER JOIN**.
* Cuối cùng, xác định điều kiện để nối hai bảng.
* Sử dụng **FULL JOIN** để lấy các bản ghi từ cả hai bảng bao gồm bản ghi trùng khớp và không trùng khớp giá trị:

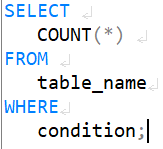
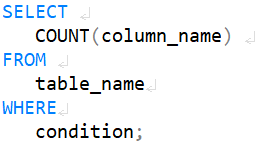
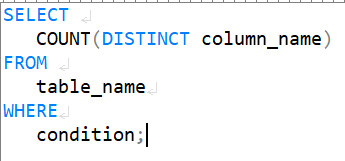
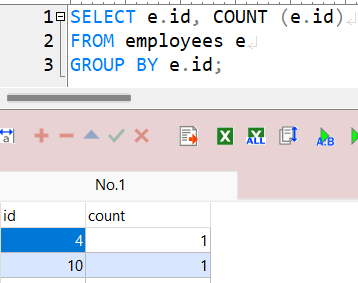


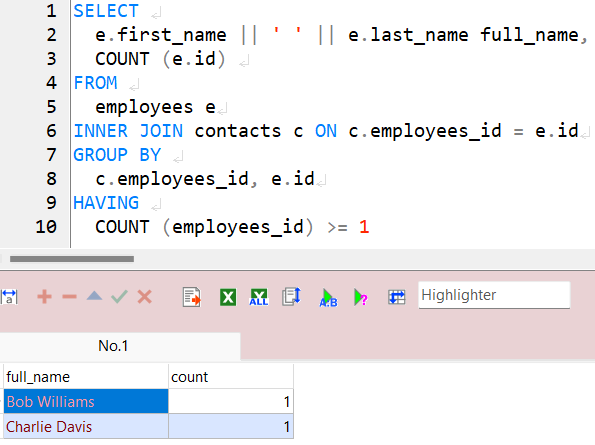
* Chú ý:
* **FULL OUTER JOIN** còn được gọi là **FULL JOIN.**
* Sử dụng mệnh đề **FULL OUTER JOIN** để kết hợp dữ liệu từ cả hai bảng, đảm bảo rằng các hàng khớp nhau được bao gồm từ cả bảng bên trái và bên phải, cũng như các hàng không khớp từ một trong hai bảng.

## **Ngày 3: Group By, Order By**

### **Aggregate functions: COUNT, SUM, AVG, MIN, MAX**

#### **COUNT**

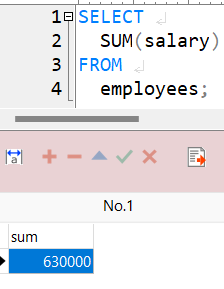
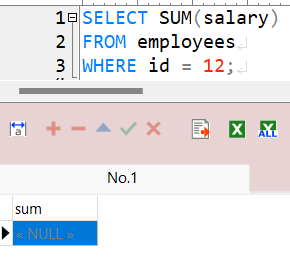
* Hàm này **COUNT()** là hàm tổng hợp cho phép lấy được số hàng khớp với một điều kiện cụ thể.
* Sử dụng hàm PostgreSQL **COUNT()** để trả về số hàng trong một bảng.
* **COUNT(\*)**:
* Hàm này **COUNT(\*)** trả về số hàng được trả về bởi một câu lệnh **SELECT**, bao gồm cả NULL và các hàng trùng lặp.
* **COUNT(column\_name):**
* hàm trả về số hàng được trả về bởi một mệnh đề **SELECT**. Tuy nhiên, nó không xem xét các giá trị **NULL** trong column\_name.
* **COUNT(DISTINCT column\_name)**
* trả về số lượng giá trị không null duy nhất trong column\_name.
* Sử dụng hàm **COUNT()** của PostgreSQL với mệnh đề **GROUP BY**
* Sử dụng hàm **COUNT**() của PostgreSQL với mệnh đề **HAVING**

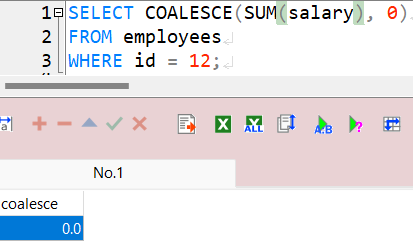


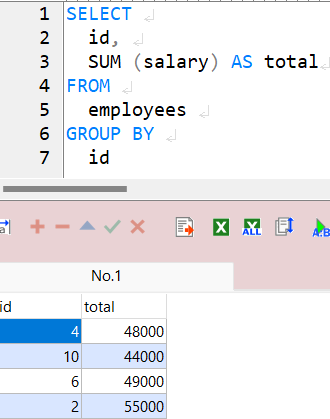
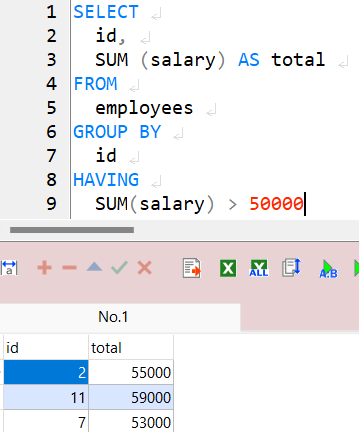
#### **SUM**

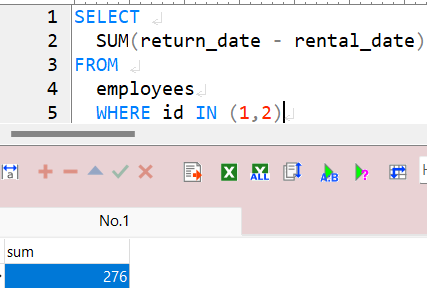
* PostgreSQL **SUM()** là một [hàm tổng hợp](https://www.postgresqltutorial.com/postgresql-aggregate-functions/) trả về tổng các giá trị trong một tập hợp.
* Cú pháp:



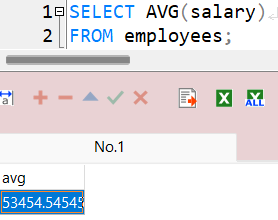
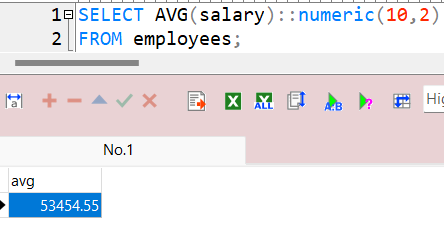
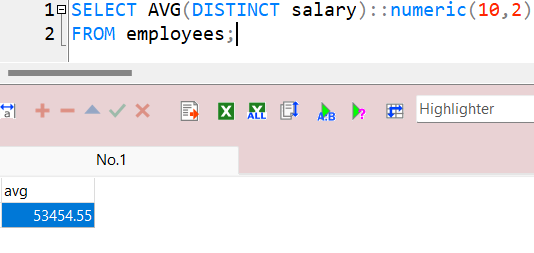
* Chú ý:
* Hàm này **SUM()** bỏ qua **NULL**.
* Nếu sử dụng **DISTINCT**, hàm **SUM()** sẽ chỉ tính tổng của các giá trị riêng biệt.
* Kết quả **SUM()** của một tập hợp rỗng sẽ trả về **NULL**, không phải 0.
* Sử dụng hàm **SUM()** để tính tổng các giá trị trong một tập hợp.
* Sử dụng **DISTINCT** trong hàm **SUM()** để tính tổng các giá trị khác biệt.
* Sử dụng hàm **SUM()** với mệnh đề **GROUP BY** để tính tổng cho mỗi nhóm.
* Sử dụng hàm **SUM()** của PostgreSQL trong ví dụ câu lệnh **SELECT**
* Sử dụng hàm **SUM()** của PostgreSQL với **tập kết quả rỗng**
* Sử dụng hàm **SUM()** với ví dụ hàm **COALESCE()**
* Nếu muốn hàm **SUM()** trả về giá trị 0 thay vì **NULL** trong trường hợp không có hàng nào khớp, sử dụng hàm **COALESCE()**.

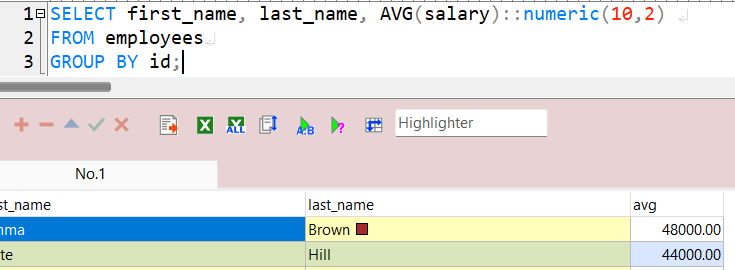


* Sử dụng hàm **SUM()** của PostgreSQL với ví dụ về mệnh đề **GROUP BY**
* Sử dụng hàm **SUM()** của PostgreSQL với mệnh đề **HAVING**
* Sử dụng hàm **SUM()** của PostgreSQL với **một biểu thức**

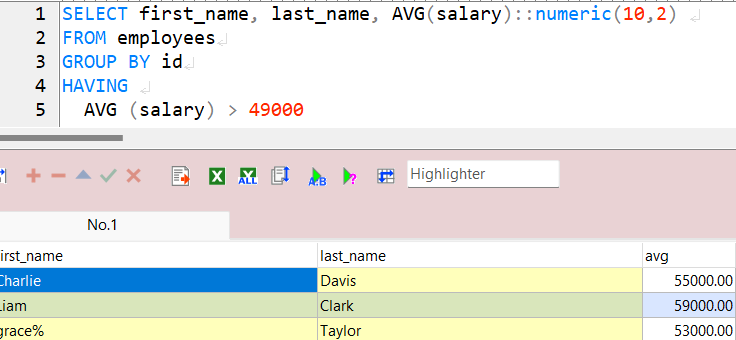


#### **AVG**

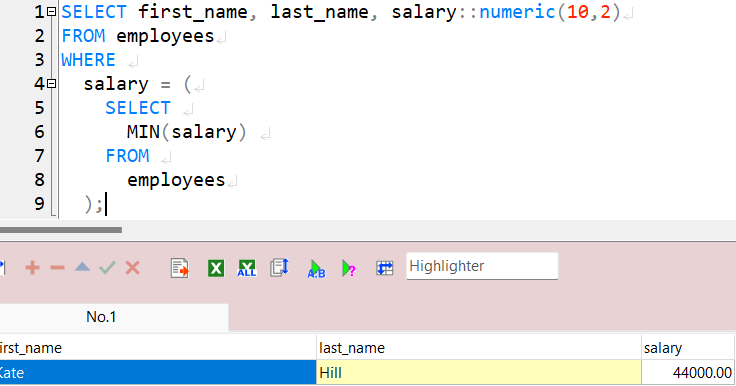
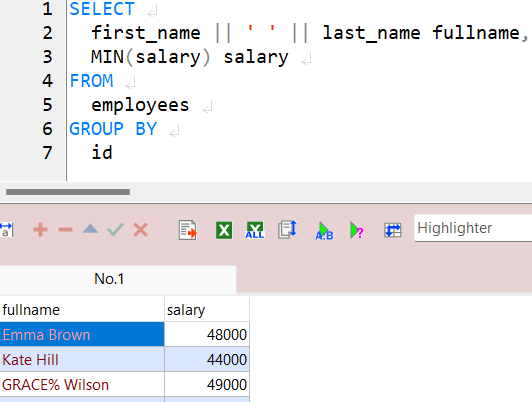
* Hàm này cho phép bạn tính giá trị trung bình của một tập hợp.
* Cú pháp:
* Lưu ý:
* Hàm **AVG**() bỏ qua **NULL**. Nếu cột không có giá trị, hàm **AVG**() trả về **NULL**.
* Có thể sử dụng hàm **AVG**() trong mệnh đề **SELECT** and **HAVING**.
* Sử dụng **AVG()** tính trung bình salary:
* Có thể ép kiểu:
* Sử dụng hàm **AVG()** với ví dụ toán tử **DISTINCT**
* Sử dụng hàm **AVG()** của với mệnh đề **GROUP BY**

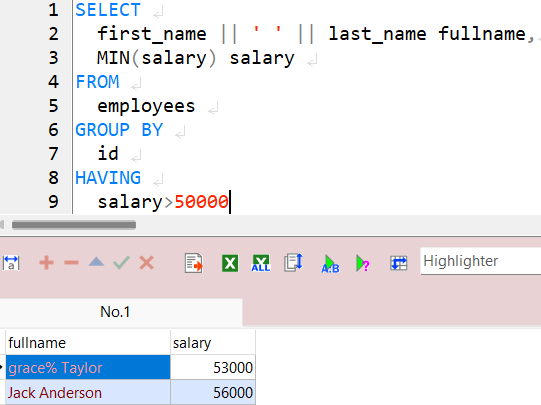


* Sử dụng hàm **AVG()** với mệnh đề **HAVING**

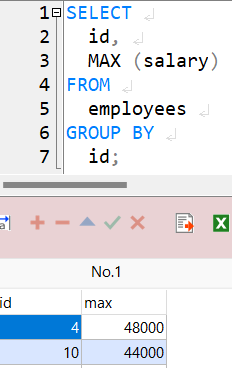
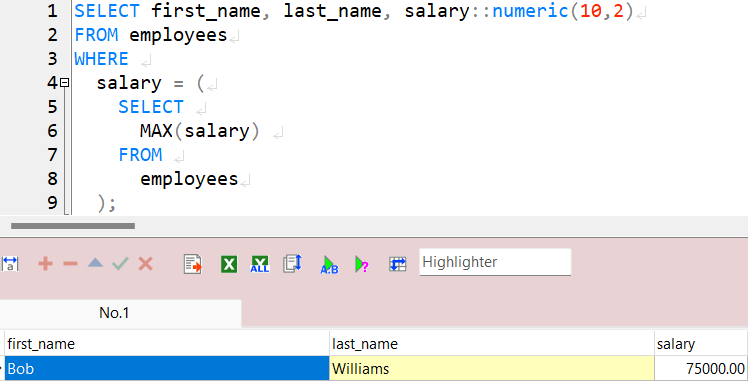


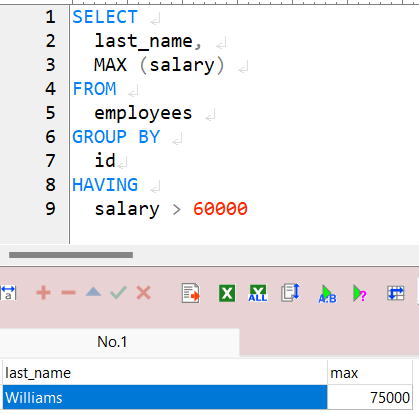
#### **MIN**

* Hàm **MIN()** là [hàm tổng hợp](https://www.postgresqltutorial.com/postgresql-aggregate-functions/) trả về giá trị nhỏ nhất trong một tập hợp các giá trị.
* Cú pháp:
* Lưu ý:
* **DISTINCT** không có bất kỳ tác động nào đến hàm **MIN()**.
* Sử dụng hàm **MIN()** để tìm giá trị thấp nhất trong một tập hợp các giá trị.
* Sử dụng mệnh đề **MIN()** với **GROUP BY** để tìm giá trị thấp nhất trong một nhóm giá trị.
* Sử dụng hàm **MIN()** trong ví dụ truy vấn phụ
* Sử dụng hàm **MIN()** với mệnh đề **GROUP BY**
* Sử dụng hàm **MIN()** với ví dụ mệnh đề **HAVING**

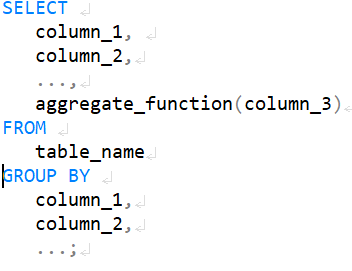
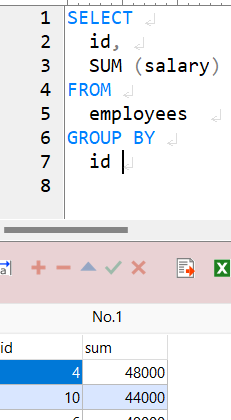
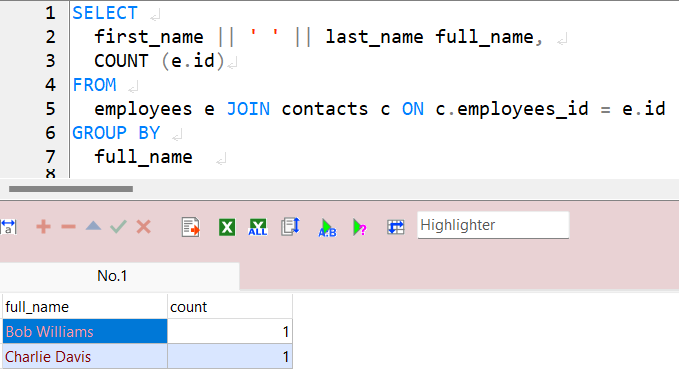
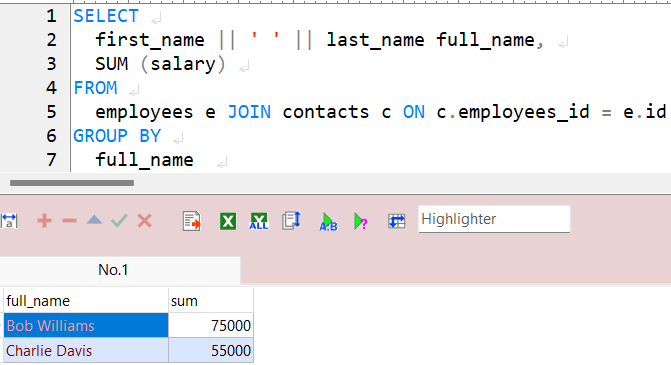


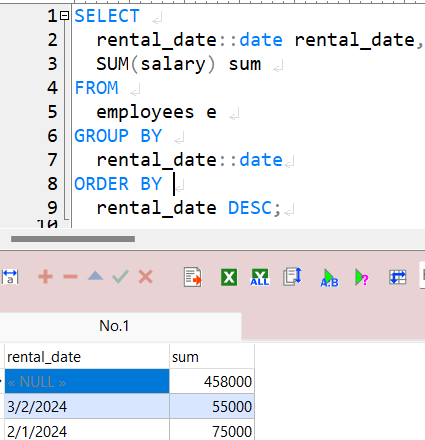
#### **MAX**

* Hàm **MAX()** là [hàm tổng hợp](https://www.postgresqltutorial.com/postgresql-aggregate-functions/) trả về giá trị lớn nhất trong một tập hợp các giá trị.
* Cú pháp:
* Lưu ý:
* Có thể sử dụng hàm **MAX()** không chỉ trong mệnh đề **SELECT** mà còn trong mệnh đề **WHERE** và **HAVING.**
* Sử dụng hàm **MAX()**để tìm giá trị lớn nhất của một tập hợp.
* Sử dụng hàm **MAX()** trong **truy vấn phụ**
* Sử dụng hàm **MAX()** với mệnh đề **GROUP BY**
* Sử dụng hàm **MAX()** với mệnh đề **HAVING**



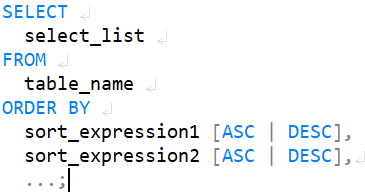
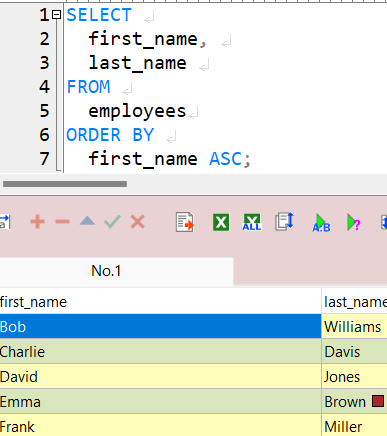
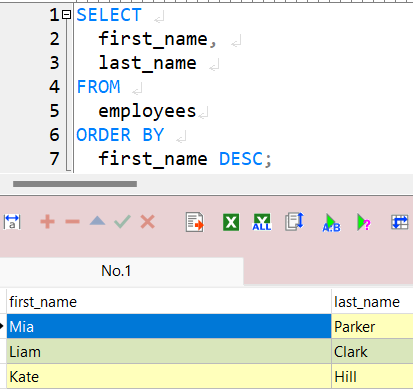
### **GROUP BY**

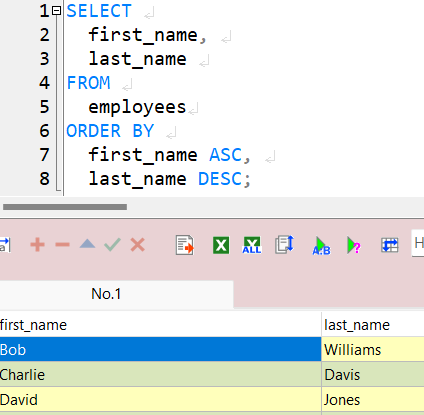
* Mệnh đề **GROUP BY** chia các hàng trả về từcâu lệnh **SELECT** thành các nhóm.
* Cú pháp:
* Đầu tiên, chọn các cột mà muốn nhóm như column1 và column2, và cột mà muốn áp dụng hàm tổng hợp ( column3).
* Thứ hai, liệt kê các cột mà muốn nhóm trong mệnh đề **GROUP BY**.
* Sử dụng **GROUP BY** với ví dụ về hàm **SUM():**
* Sử dụng mệnh đề **GROUP BY** với mệnh đề **JOIN**
* Sử dụng **GROUP BY** với ví dụ hàm **COUNT()**
* Sử dụng mệnh đề **GROUP BY** với cột ngày

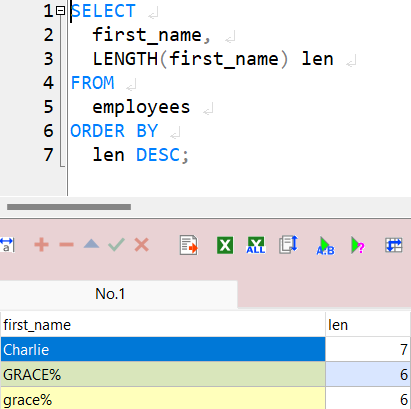


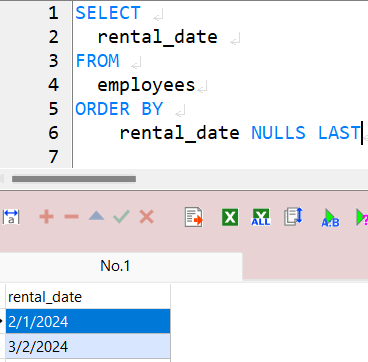
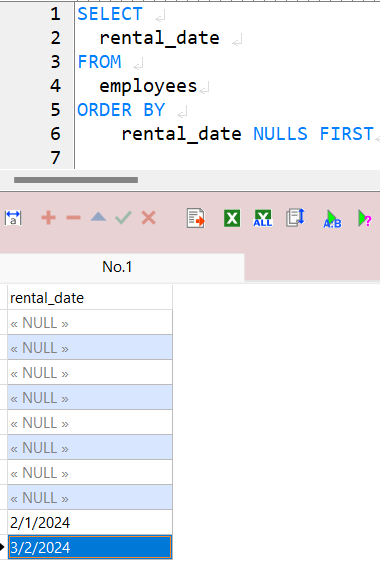
* Chú ý:
* PostgreSQL đánh giá mệnh đề **GROUP BY** sau mệnh đề **FROM** và **WHERE**, và trước mệnh đề **HAVING** [**SELECT**](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-select/), **DISTINCT**, **ORDER B**[**Y**](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-order-by/) và **LIMIT**.
* Sử dụng mệnh đề **GROUP BY** để chia các hàng thành **các nhóm** và áp dụng **hàm tổng hợp** cho **mỗi nhóm**.

### **ORDER BY**

* Để sắp xếp các hàng của tập kết quả, bạn sử dụng mệnh **ORDER BY** đề trong câu lệnh **SELECT**.
* Mệnh đề **ORDER BY** cho phép sắp xếp các hàng trả về bởi một mệnh đề **SELECT** theo thứ tự tăng dần hoặc giảm dần dựa trên biểu thức sắp xếp.
* Cú pháp:
* Đầu tiên, chỉ định một biểu thức sắp xếp, có thể là một cột hoặc một biểu thức, mà muốn sắp xếp sau các từ khóa **ORDER BY**. Nếu muốn sắp xếp tập kết quả dựa trên nhiều cột hoặc biểu thức, cần đặt dấu phẩy ( ,) giữa hai cột hoặc biểu thức để phân tách chúng.
* Thứ hai, sử dụng **ASC** để sắp xếp các hàng theo thứ tự tăng dần và **DESC** để sắp xếp các hàng theo thứ tự giảm dần. Nếu bỏ qua tùy chọn **ASC** hoặc **DESC**, thì mặc định **ORDER BY** sẽ sử dụng **ASC**.
* Sử dụng mệnh đề **ORDER BY** để sắp xếp các hàng theo **một cột** theo thứ tự **tăng dần**
* Sử dụng mệnh đề **ORDER BY** để sắp xếp các hàng theo **một cột** theo thứ tự **giảm dần**
* Sử dụng mệnh đề **ORDER BY** để sắp xếp các hàng theo **nhiều cột**

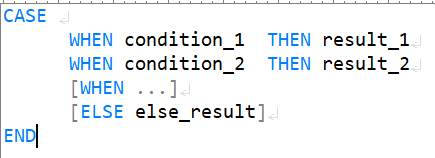
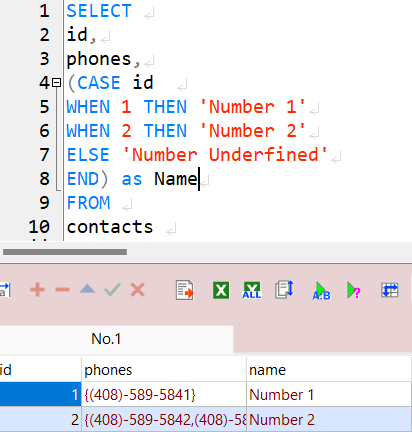


* Sử dụng mệnh đề **ORDER BY** để sắp xếp các hàng theo **biểu thức**
* Mệnh đề **ORDER BY** và **NULL**
* **NULL FIRST**: Sắp xếp các giá trị NULL lên trước.
* **NULL LAST**: Sắp xếp các giá trị NOT NULL lên trước.



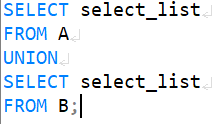
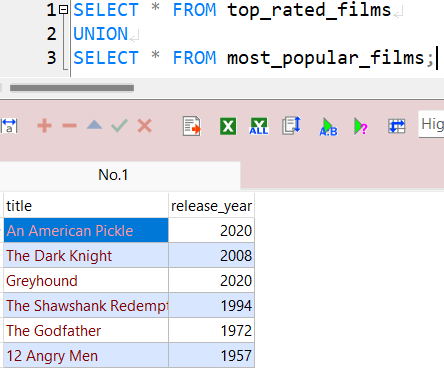
* Lưu ý:
* **ASC, DESC** tùy chọn này là mặc định nên bạn có thể bỏ nó trong **ORDER BY**.
* Mặc định **ORDER BY** là **ASC**.

### **CASE WHEN**

* Biểu thức **CASE** giống như câu lệnh **IF/ELSE** trong các ngôn ngữ lập trình khác.
* Biểu thức **CASE** có hai dạng:
* Tổng quan.
* Đơn giản.
* Cú pháp:
* Trong cú pháp này, mỗi điều kiện ( condition\_1, condition\_2…) là một biểu thức **boolean** trả về **true** hoặc **false**.
* Khi một điều kiện được đánh giá là false, biểu thức **CASE** sẽ đánh giá điều kiện tiếp theo từ trên xuống dưới cho đến khi tìm thấy điều kiện được đánh giá là **true**.
* Nếu một điều kiện được đánh giá là **true**, biểu thức **CASE** sẽ trả về kết quả tương ứng theo sau điều kiện.
* Ví dụ, nếu condition\_2 đánh giá là **true**, biểu thức **CASE** trả về result\_2. Ngoài ra, nó ngay lập tức dừng đánh giá các biểu thức còn lại.
* Nếu tất cả các điều kiện là false, biểu thức **CASE** trả về kết quả ( else\_result) theo sau từ khóa **ELSE**. Nếu bỏ qua mệnh đề **ELSE**, biểu thức **CASE** trả về **NULL**.
* Lưu ý:
* Vì **CASE** là một biểu thức, bạn có thể sử dụng nó ở bất kỳ nơi nào sử dụng biểu thức như **SELECT**, **WHERE**, **GROUP** [**BY**](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-group-by/), và **HAVING** các mệnh đề.

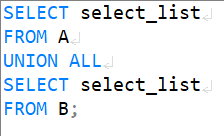
### **Union và Union All**

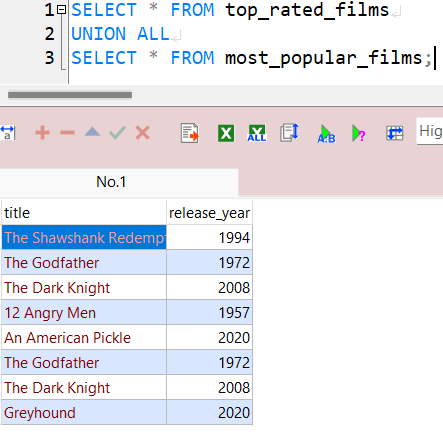
#### **Union**

* Toán tử **UNION** cho phép kết hợp các tập kết quả của hai hoặc nhiều câu lệnh [**SELECT**](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-select/)thành một tập kết quả duy nhất.
* Cú pháp:
* **Số lượng** và **thứ tự** của các cột trong danh sách chọn của cả hai truy vấn phải giống nhau.
* **Kiểu dữ liệu** của các cột trong danh sách truy vấn được chọn phải **tương thích**.
* Lưu ý:
* Sử dụng **UNION** để kết hợp các tập kết quả của hai truy vấn và trả về các **hàng riêng biệt**.

#### **Union All**

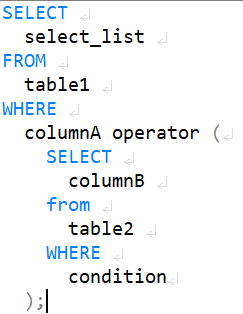
* Toán tử **UNION ALL** cho phép kết hợp các tập kết quả nhưng vẫn giữ nguyên các **hàng trùng lặp**.
* Cú pháp:

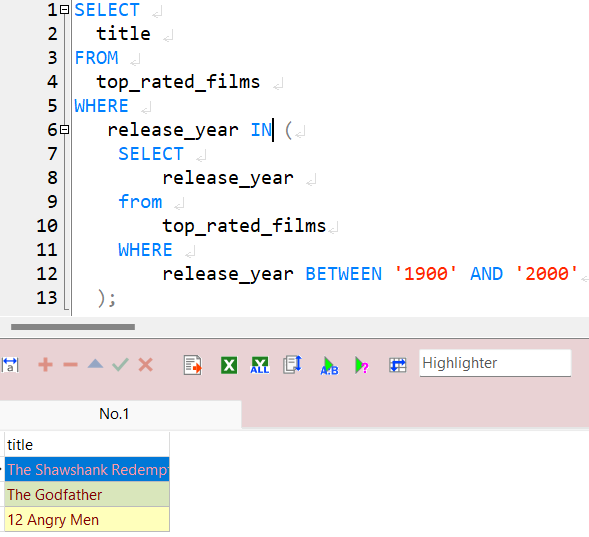


* Lưu ý:
* Sử dụng **UNION ALL** để kết hợp các tập kết quả của hai truy vấn nhưng vẫn giữ nguyên các **hàng trùng lặp**.

### **Subqueries**

* **Truy vấn con** là [truy vấn](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-select/) **lồng nhau** trong một truy vấn khác. Truy vấn con cũng được gọi là **truy vấn bên trong** hoặc truy **vấn lồng nhau**.
* Truy vấn chính sẽ sử dụng kết quả của truy vấn phụ để lọc dữ liệu trong mệnh đề **WHERE**.
* Cú pháp:



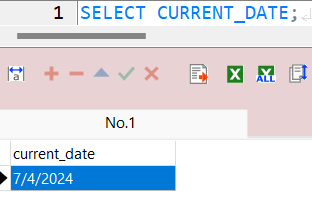
* Sử dụng truy vấn con với toán tử **IN**
* Lưu ý:
* Một **truy vấn con** là một truy vấn được **lồng** bên trong một truy vấn khác
* Truy vấn con còn được gọi là truy vấn bên trong hoặc **truy vấn lồng nhau**.

## **Ngày 4: Function Date, Function String**

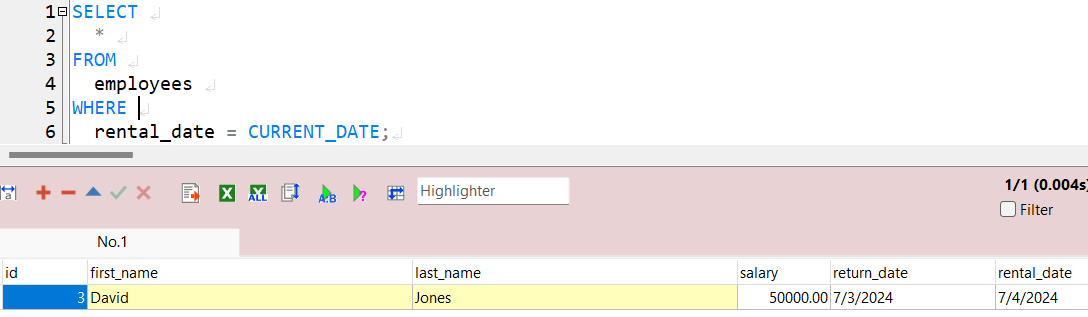
### **Function Date**

#### **CURRENT\_DATE**

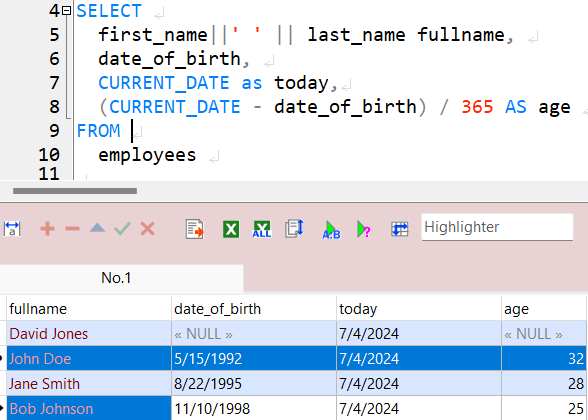
* Hàm này **CURRENT\_DATE** để trả về ngày hiện tại theo múi giờ mặc định của phiên cơ sở dữ liệu.
* Cú pháp:

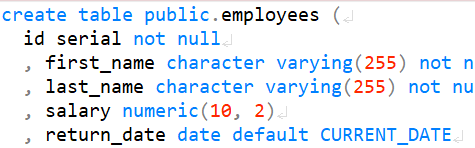


* Sử dụng **CURRENT\_DATE** để lọc dựa trên ngày:



* Tính tuổi

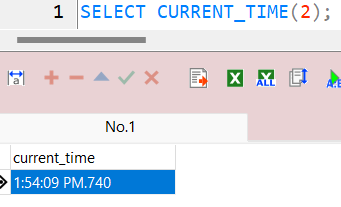


* Dùng làm giá trị mặc định cho một cột

* Lưu ý:
* Lưu ý rằng hàm **CURRENT\_DATE** này trả về ngày hiện tại mà không có thông tin thời gian nào.

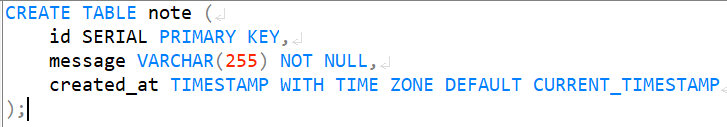
#### **CURRENT\_TIME**

* Sử dụng hàm **CURRENT\_TIME** để lấy thời gian hiện tại theo múi giờ mặc định.
* Cú pháp:
* **precision** định độ chính xác giây phân số được trả về. Nếu bỏ qua đối số **precision** kết quả sẽ bao gồm độ chính xác có sẵn đầy đủ.
* Sử dụng hàm **CURRENT\_TIME** với độ chính xác được đặt thành 2

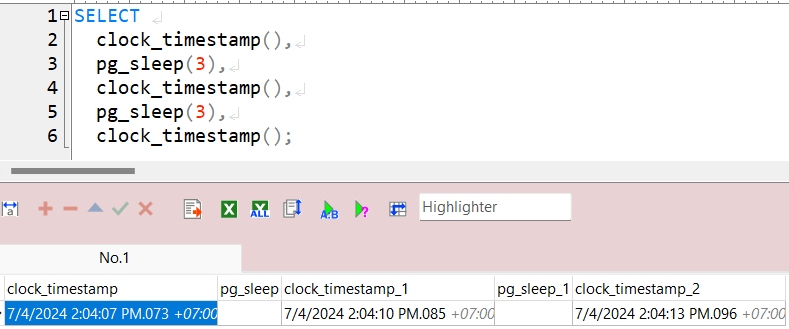


* Sử dụng hàm **CURRENT\_TIME** làm giá trị mặc định của một cột

#### **CURRENT\_TIMESTAMP**

* Để trả về ngày và giờ hiện tại kèm theo múi giờ.
* Cú pháp:
* **precision**: chỉ định số chữ số theo độ chính xác giây phân số trong trường thứ hai của kết quả.
* Sử dụng hàm **CURRENT\_TIMESTAMP** làm giá trị mặc định của một cột
* Chú ý:
* Sử dụng **CURRENT\_TIMESTAMP()** để lấy ngày và giờ giao dịch bắt đầu.

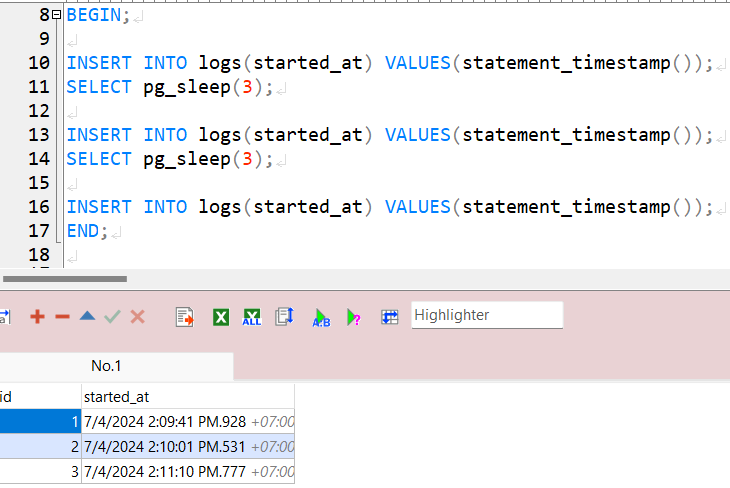
#### **CLOCK\_TIMESTAMP()**

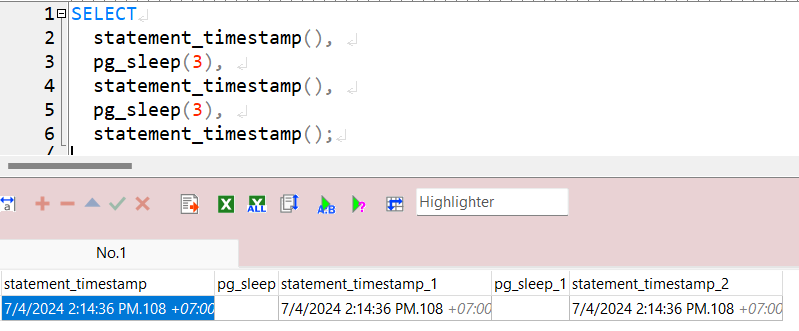
* Để trả về ngày và giờ hiện tại kèm theo múi giờ.
* Cú pháp:
* Gọi hàm **CLOCK\_TIMESTAMP**() nhiều lần trong một câu lệnh
* Chú ý:
* Hàm này **CLOCK\_TIMESTAMP**() không có tham số.
* Hàm này **CLOCK\_TIMESTAMP**() trả về ngày và giờ hiện tại dưới dạng dấu thời gian có múi giờ .
* Khi gọi **CLOCK\_TIMESTAMP**()hàm nhiều lần trong một câu lệnh, bạn sẽ nhận được những kết quả khác nhau.

#### **STATEMENT\_TIMESTAMP**

* Để trả về thời gian bắt đầu của câu lệnh hiện tại.
* Cú pháp:



* Sử dụng **STATEMENT\_TIMESTAMP()**  trong một giao dịch
* Gọi hàm **STATEMENT\_TIMESTAMP()** nhiều lần trong một câu lệnh

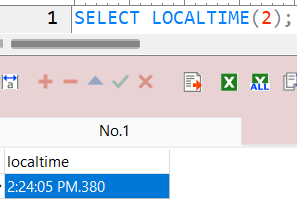


* Chú ý:
* Không chấp nhận bất kỳ đối số nào.
* Trả về một giá trị có kiểu **TIMESTAMP WITH TIME ZONE.**
* Để lấy thời gian bắt đầu của câu lệnh hiện tại.

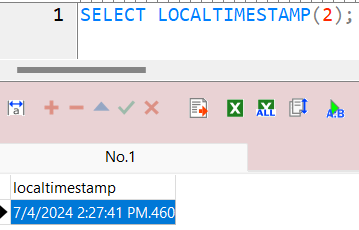
#### **NOW**

* Dùng để trả về ngày và giờ hiện tại theo múi giờ của máy chủ cơ sở dữ liệu.
* Cú pháp:
* Hàm **NOW()** làm giá trị mặc định

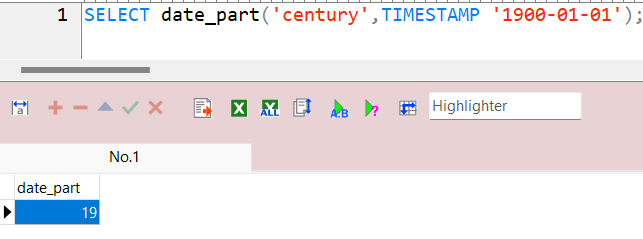
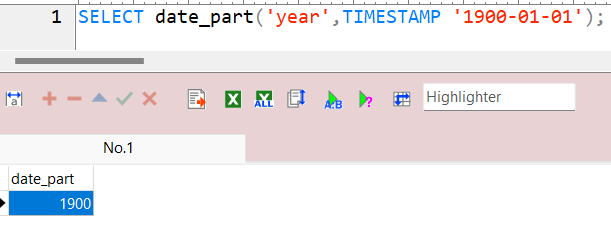
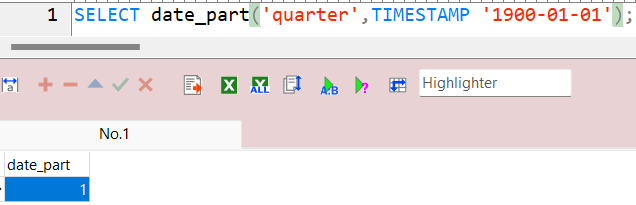
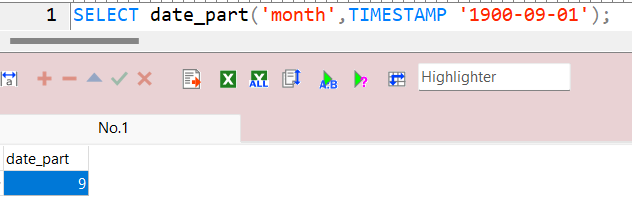
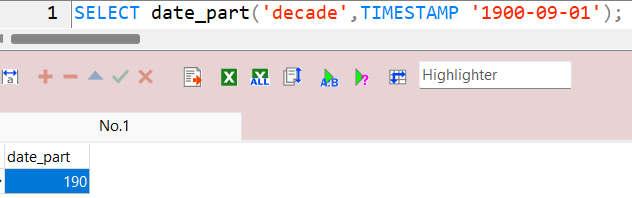
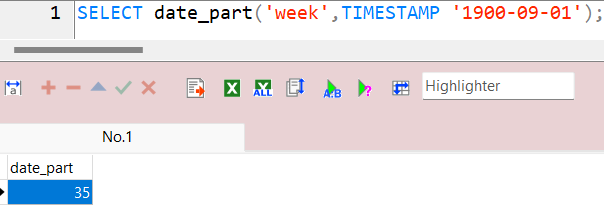
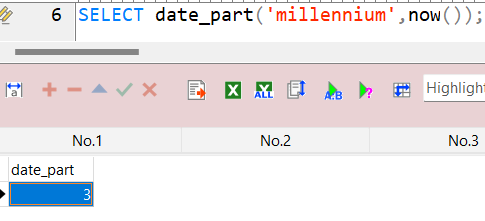
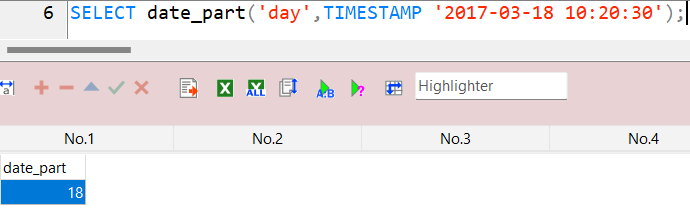
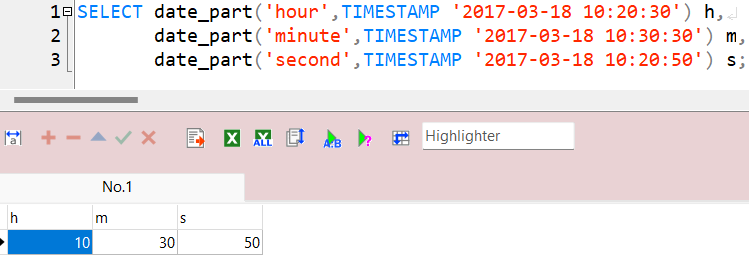
#### **LOCALTIME**

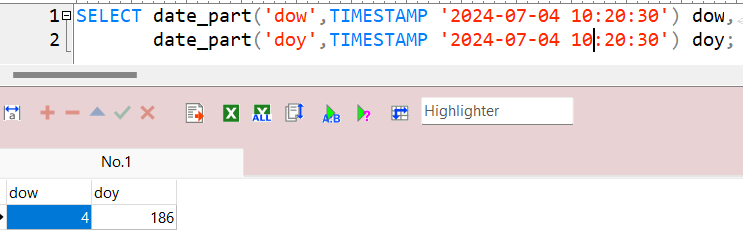
* Dùng để trả về thời gian hiện tại mà giao dịch hiện tại bắt đầu.
* Cú pháp:
* Đối số **precision** chỉ định độ chính xác đến từng giây của trường thứ hai.
* Sử dụng hàm **LOCALTIME** với độ chính xác tính bằng giây
* Lưu ý:
* Nếu bạn bỏ qua đối số **precision**, giá trị mặc định sẽ là 6.

#### **LOCALTIMESTAMP**

* Dùng để trả về ngày và giờ hiện tại mà giao dịch hiện tại bắt đầu.
* Cú pháp: 
* Đối số **precision** chỉ định độ chính xác đến từng giây của trường thứ hai.
* Sử dụng hàm **LOCALTIMESTAMP** với ví dụ về độ chính xác của giây phân số
* Chú ý:
* Hàm **LOCALTIMESTAMP** trả về giá trị **TIMESTAMP không có** **múi giờ** trong khi **CURRENT\_TIMESTAMP** hàm trả về giá trị **TIMESTAMP có** **múi giờ**.

#### **DATE\_PART**

* Dùng để lấy các trường con như năm, tháng và tuần từ giá trị [ngày](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-date/) hoặc [giờ](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-time/) .
* Cú pháp:
* **field** là một mã định danh xác định nội dung cần trích xuất từ **source​​**.
* Giá trị của **field** có thể là:
* century
* decade
* year
* month
* day
* hour
* minute
* second
* microseconds
* milliseconds
* dow
* doy
* epoch
* isodow
* isoyear
* timezone
* timezone\_hour
* timezone\_minute
* **Source** là một biểu thức thời gian được đánh giá là **TIMESTAMP**, **TIME**, hoặc **INTERVAL**. Nếu **source** được đánh giá là **DATE**, hàm sẽ được ép kiểu thành **TIMESTAMP**.
* Sử dụng hàm **DATE\_PART**() để trích xuất thế kỷ từ dấu thời gian
* Trích xuất **năm** từ dấu thời gian
* Trích xuất **quý** từ dấu thời gian
* Trích xuất **tháng** từ dấu thời gian
* Trích xuất **một thập kỷ** từ dấu thời gian
* Trích xuất **số tuần** từ dấu thời gian
* Để lấy **thiên niên kỷ** hiện tại, bạn sử dụng hàm **DATE\_PART**() với **NOW**()
* Trích xuất **ngày** từ dấu thời gian
* Trích xuất **giờ, phút và giây** từ dấu thời gian
* Trích xuất **ngày trong tuần** hoặc **ngày trong năm** từ dấu thời gian

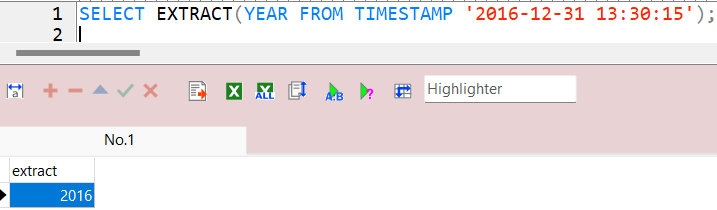


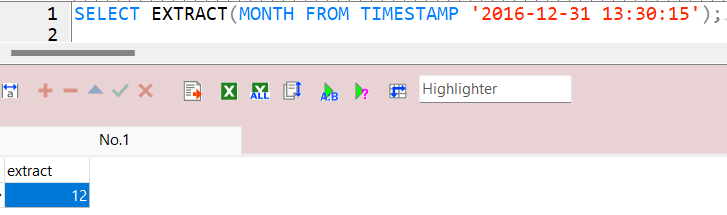
* Chú ý:
* Hàm **DATE\_PART()** trả về một giá trị có kiểu là độ chính xác kép.
* Sử dụng hàm **DATE\_PART()** để trích xuất một trường con của dấu thời gian.

#### **Extract**

* Dùng để trích xuất một trường như **năm, tháng** và **ngày** từ giá trị **ngày/giờ**.
* Cú pháp: 
* Đối số **field** chỉ định thông tin muốn trích xuất từ ​​giá trị ngày/giờ.

| **Field** | **TIMESTAMP** | **Interval** |
| --- | --- | --- |
| **CENTURY** | Thế kỷ | Số thế kỷ |
| **DAY** | Ngày trong tháng (1-31) | Số ngày |
| **DECADE** | Thập kỷ là trường năm chia cho 10 |  |
| **DOW** | Ngày trong tuần (Chủ Nhật (0), Thứ Hai (1) … Thứ Bảy (6)) | N/A |
| **DOY** | Ngày trong năm (1-365/366) | N/A |
| **EPOCH** | Số giây kể từ 1970-01-01 00:00:00 UTC | Tổng số giây trong khoảng thời gian |
| **HOUR** | Giờ (0-23) | Số giờ |
| **ISODOW** | Ngày trong tuần, từ thứ Hai (1) đến Chủ Nhật (7) | N/A |
| **ISOYEAR** | Số tuần của năm theo tiêu chuẩn ISO 8601 | N/A |
| **MICROSECONDS** | Trường thứ hai, bao gồm các phần phân số, nhân với 1000000 | Giống như TIMESTAMP |
| **MILLENNIUM** | Thiên niên kỷ | Số thiên niên kỷ |
| **MILLISECONDS** | Trường thứ hai, bao gồm các phần phân số, nhân với 1000 | Giống như TIMESTAMP |
| **MINUTE** | Phút (0-59) | Số phút |
| **MONTH** | Tháng 1-12 | Số tháng, modulo (0-11) |
| **QUARTER** | Quý của năm (1 – 4) | Số lượng quý |
| **SECOND** | Trường thứ hai, bao gồm bất kỳ giây phân số nào | Số giây |
| **TIMEZONE** | Độ lệch múi giờ so với UTC, được đo bằng giây | N/A |
| **TIMEZONE\_HOUR** | Thành phần giờ của độ lệch múi giờ | N/A |
| **TIMEZONE\_MINUTE** | Thành phần phút của độ lệch múi giờ | N/A |
| **WEEK** | Số tuần đánh số theo tiêu chuẩn ISO 8601 là tuần của năm | N/A |
| **YEAR** | Năm | Giống như TIMESTAMP |

* **Source** là giá trị có kiểu **TIMESTAMP** hoặc **INTERVAL**. Nếu bạn truyền một giá trị **DATE**, hàm sẽ ép kiểu đó thành một giá trị **TIMESTAMP**.
* Hàm **EXTRACT**() trả về giá trị có độ chính xác kép.
* Sử dụng hàm **EXTRACT**() để trích xuất năm từ dấu thời gian
* Sử dụng hàm **EXTRACT()** để trích xuất tháng từ dấu thời gian

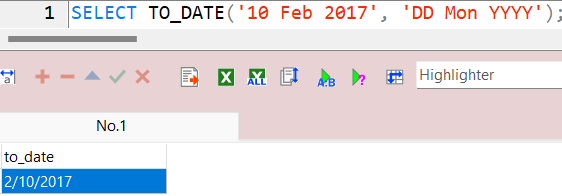


* Sử dụng hàm **EXTRACT()** để trích xuất ngày từ dấu thời gian

#### **TO\_DATE**

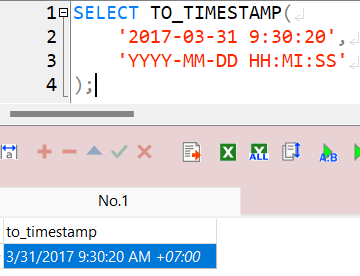
* Dùng để chuyển đổi chuỗi thành ngày.
* Cú pháp: 
* **text**: là chuỗi đầu vào mà muốn chuyển đổi thành ngày.
* **format**: chỉ định định dạng của chuỗi đầu vào.

| **Pattern** | **Description** |
| --- | --- |
| **Y,YYY** | năm gồm 4 chữ số có dấu phẩy |
| **YYYY** | năm gồm 4 chữ số |
| **YYY** | 2 chữ số cuối của năm |
| **YY** | Chữ số cuối cùng của năm |
| **Y** | 3 chữ số cuối của năm đánh số tuần theo ISO 8601 |
| **IYYY** | Năm đánh số tuần theo ISO 8601 (4 chữ số trở lên) |
| **IYY** | 2 chữ số cuối của năm đánh số tuần theo ISO 8601 |
| **IY** | Chữ số cuối cùng của năm đánh số tuần theo ISO 8601 |
| **I** | Tên tháng viết hoa viết tắt, ví dụ: Jan, Feb, v.v. |
| **BC, bc, AD or ad** | Chỉ số thời đại không có dấu chấm |
| **B.C., b.c., A.D. ora.d.** | Chỉ báo thời đại với các giai đoạn |
| **MONTH** | Tên tháng tiếng Anh viết hoa |
| **Month** | Tên tháng tiếng Anh viết hoa đầy đủ |
| **month** | Tên tháng tiếng Anh viết thường đầy đủ |
| **MON** | Tên tháng viết hoa viết tắt, ví dụ: JAN, FEB, v.v. |
| **Mon** | Số tuần trong năm (1-53) (tuần đầu tiên bắt đầu vào ngày đầu tiên của năm) |
| **mon** | Tên tháng viết thường viết tắt, ví dụ: Jan, Feb, v.v. |
| **MM** | số tháng từ 01 đến 12 |
| **DAY** | Tên ngày viết hoa đầy đủ |
| **Day** | Tên ngày viết hoa đầy đủ |
| **day** | Tên ngày viết thường đầy đủ |
| **DY** | Tên ngày viết hoa viết tắt |
| **Dy** | Tên ngày viết tắt viết hoa |
| **dy** | Tên ngày viết thường viết tắt |
| **DDD** | Ngày trong năm (001-366) |
| **IDDD** | Ngày của năm đánh số tuần theo ISO 8601 (001-371; ngày 1 của năm là thứ Hai của tuần ISO đầu tiên) |
| **DD** | Ngày trong tháng (01-31) |
| **D** | Ngày trong tuần, Chủ Nhật (1) đến Thứ Bảy (7) |
| **ID** | Ngày trong tuần theo tiêu chuẩn ISO 8601, Thứ Hai (1) đến Chủ Nhật (7) |
| **W** | Ngày trong tuần theo tiêu chuẩn ISO 8601, Thứ Hai (1) đến Chủ Nhật (7) |
| **WW** | Ví dụ thế kỷ 21, 22, v.v. |
| **IW** | Số tuần của năm đánh số tuần theo ISO 8601 (01-53; thứ Năm đầu tiên của năm là tuần 1) |
| **CC** | Ví dụ thế kỷ 21, 22, v.v. |
| **J** | Ngày Julian (ngày nguyên kể từ ngày 24 tháng 11 năm 4714 trước Công nguyên vào lúc nửa đêm UTC) |
| **RM** | Tháng bằng số La Mã viết hoa (I-XII; I=tháng 1) |
| **rm** | Tháng bằng số La Mã viết thường (i-xii; i=tháng 1) |

* Chuyển đổi chuỗi 10 Feb 2017 thành giá trị ngày

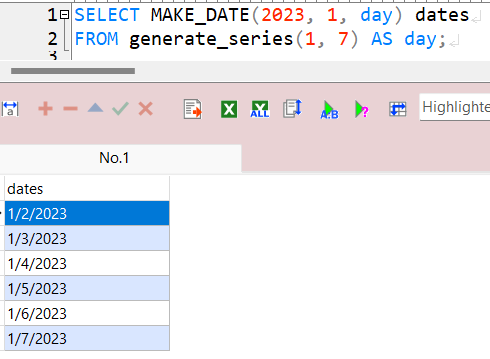
#### **TO\_TIMESTAMP**

* Dùng để chuyển đổi chuỗi thành dấu thời gian dựa trên định dạng đã chỉ định.
* Cú pháp:



* Chú ý:
* Chỉ bỏ qua các khoảng trắng và trả về giá trị dấu thời gian chính xác.
* Hàm này coi mili giây hoặc micro giây là giây sau dấu thập phân.

#### **MAKE\_DATE**

* Dùng để tạo giá trị ngày tháng từ năm, tháng và ngày.
* Cú pháp:
* Sử dụng hàm **MAKE\_DATE()** để tạo các ngày tuần tự

#### **MAKE\_TIME**

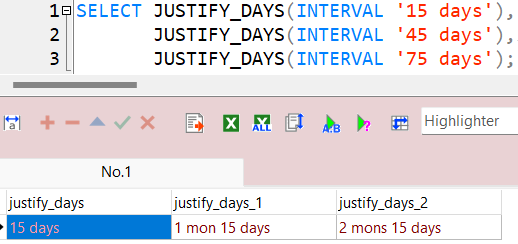
* Dùng để tạo giá trị thời gian từ các giá trị giờ, phút và giây.
* Cú pháp:
* **hour**: Phần giờ của thời gian. Phạm vi hợp lệ cho giờ là từ 0 đến 12 (AM biểu thị sáng và PM biểu thị trưa).
* **min**: Phần phút của thời gian. Phạm vi hợp lệ cho giây là từ 0 đến 59.
* **sec**: Giây trong một phút. Phạm vi hợp lệ của nó là từ 0 đến 59.999999
* Sử dụng hàm **MAKE\_TIME**() với đối số chuỗi

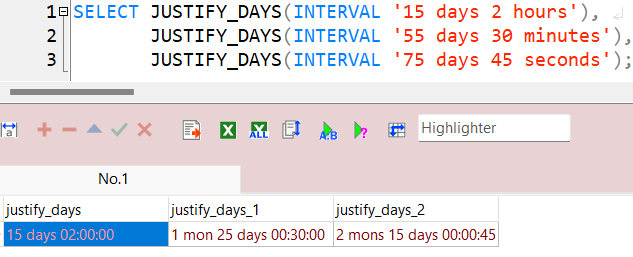
#### **AGE**

* Dùng để tính tuổi.
* Cú pháp:

#### **JUSTIFY\_DAYS**

* Dùng để điều chỉnh khoảng thời gian 30 ngày thành tháng.
* Cú pháp:
* **value** là giá trị khoảng mà muốn giải thích.
* Sử dụng hàm **JUSTIFY\_DAYS**() với các khoảng thời gian

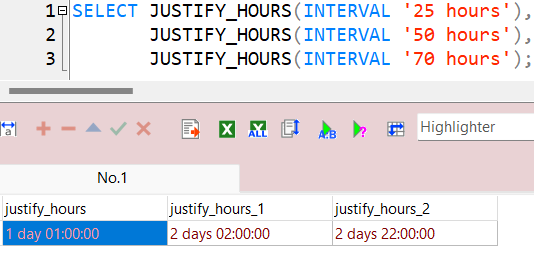


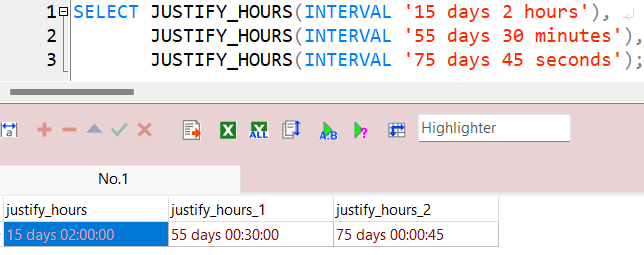
* Sử dụng hàm **JUSTIFY\_DAYS()** với các khoảng thời gian bao gồm giờ
* Chú ý:
* Những ngày vượt quá 30 sẽ được chuyển đổi thành tháng.
* Những ngày còn lại được tính theo ngày.
* Giờ, phút và giây vẫn còn nguyên vẹn.

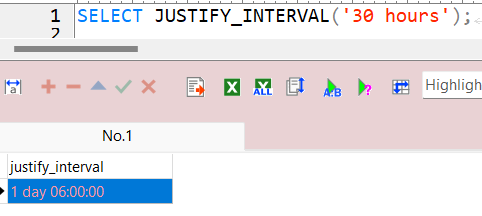
#### **JUSTIFY\_HOURS**

* Dùng để điều chỉnh khoảng thời gian 24 giờ thành ngày.
* Cú pháp:



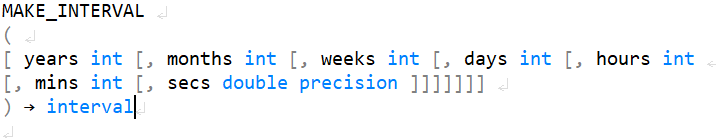
* Chú ý:
* Giờ vượt quá 24 giờ sẽ được chuyển đổi thành ngày.
* Các giờ còn lại được giữ nguyên.
* Phút, giây và các đơn vị khác không thay đổi.
* Sử dụng hàm **JUSTIFY\_HOURS**() với các khoảng thời gian
* Sử dụng hàm **JUSTIFY\_HOURS**() với các khoảng thời gian bao gồm giờ

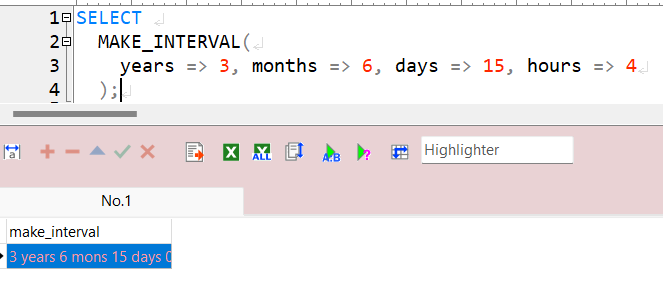
1. **JUSTIFY\_INTERVAL**

* Dùng để điều chỉnh khoảng thời gian.
* Cú pháp:
* Chú ý:
* Chuyển đổi những ngày vượt quá 30 ngày thành tháng và những ngày còn lại.
* Chuyển đổi số giờ vượt quá 24 giờ thành số ngày và số giờ còn lại.
* Đánh dấu đúng (dương hoặc âm) trong toàn bộ thời gian.
* Căn chỉnh khoảng thời gian tính theo ngày thành tháng

1. **MAKE\_INTERVAL**

* Dùng để tạo một khoảng thời gian từ các thành phần của khoảng thời gian.
* Cú pháp:



* **years** là số nguyên biểu thị số năm.
* **months** là số nguyên biểu thị số tháng.
* **weeks** là số nguyên biểu thị số tuần.
* **days** là số nguyên biểu thị số ngày.
* **hours** là số nguyên biểu thị số giờ.
* **mins** là số nguyên biểu thị số phút.
* **secs** là số có độ chính xác kép biểu diễn số giây.
* Tạo khoảng thời gian biểu thị 1 năm, 2 tháng, 3 ngày và 4 giờ

1. **AT TIME ZONE**

* Dùng để chuyển đổi dấu thời gian hoặc dấu thời gian có múi giờ sang múi giờ khác.
* Cú pháp:

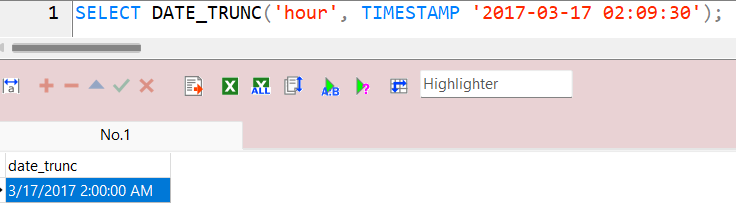


* **timestamp\_expression** là dấu thời gian hoặc dấu thời gian có giá trị múi giờ mà bạn muốn chuyển đổi.
* **target\_timezone** là múi giờ mục tiêu mà bạn muốn chuyển đổi. Đây có thể là tên múi giờ hoặc biểu thức đánh giá thành tên múi giờ.
* Chuyển đổi dấu thời gian bằng cách sử dụng múi giờ được đặt tên

#### **DATE\_TRUNC**

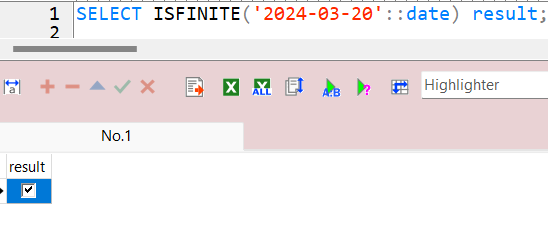
* Dùng để cắt bớt dấu thời gian hoặc khoảng thời gian theo độ chính xác được chỉ định.
* Cú pháp:

****

* **source** là giá trị hoặc biểu thức có kiểu **timestamp**, **timestamp with time zone** hoặc **interval**.
* **field** chỉ định độ chính xác nào cần cắt bớt **source.**
* Cắt bớt **TIMESTAMP** giá trị thành **hour** một phần.

#### **ISFINITE**

* Dùng để xác định xem ngày, dấu thời gian hoặc khoảng thời gian có hữu hạn hay không.
* Cú pháp:
* Kiểm tra xem một ngày có hữu hạn hay không

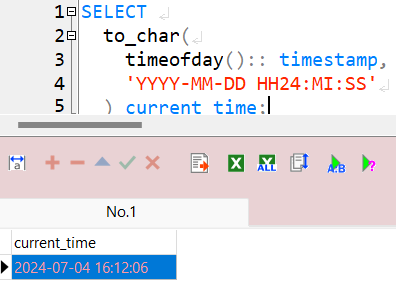


* Chú ý:
* Chấp nhận giá trị có kiểu là ngày, dấu thời gian hoặc khoảng thời gian.
* Trả về **true** nếu giá trị là hữu hạn hoặc **false** nếu không. Nó trả về **NULL** nếu giá trị là **NULL**.

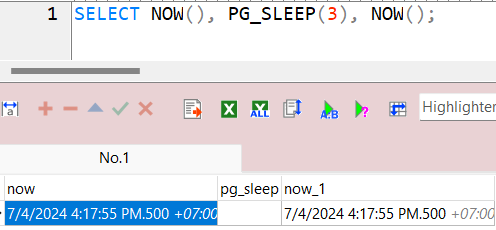
#### **TIMEOFDAY**

* Dùng để lấy ngày và giờ hiện tại dưới dạng chuỗi được định dạng.
* Cú pháp:



* Có thể định dạng đầu ra
* Chú ý:
* Trả về cùng một kết quả như hàm **CLOCK\_TIMESTAMP**() trên nhưng ở dạng chuỗi văn bản.

#### **PG\_SLEEP**

* Dùng để tạm dừng thực thi truy vấn.
* Cú pháp:
* Sử dụng hàm PG\_SLEEP() với hàm NOW()
* Sử dụng giây phân số



### **Function String**

| **Function** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| **ASCII** | Trả về giá trị mã ASCII của một ký tự hoặc điểm mã Unicode của một ký tự UTF8 | ASCII(‘A’) | 65 |
| **CHR** | Chuyển đổi mã ASCII thành ký tự hoặc mã Unicode thành ký tự UTF8 | CHR(65) | ‘A’ |
| **CONCAT** | Nối hai hoặc nhiều chuỗi thành một | CONCAT(‘A’,’B’,’C’) | ‘ABC’ |
| **CONCAT\_WS** | Nối các chuỗi bằng một dấu phân cách được chỉ định. | CONCAT\_WS(‘,’,’A’,’B’,’C’) | ‘A,B,C’ |
| **FORMAT** | Định dạng chuỗi dựa trên mẫu | FORMAT(‘Hello %s’,’PostgreSQL’) | ‘Hello PostgreSQL’ |
| **INITCAP** | Chuyển đổi các từ trong chuỗi thành chữ hoa đầu dòng | INITCAP(‘hI tHERE’) | Hi There |
| **LEFT** | Trả về ký tự n đầu tiên trong một chuỗi | LEFT(‘ABC’,1) | ‘A’ |
| **LENGTH** | Trả về số ký tự trong một chuỗi | LENGTH(‘ABC’) | 3 |
| **LOWER** | Chuyển đổi một chuỗi thành chữ thường | LOWER(‘hI tHERE’) | ‘hi there’ |
| **LPAD** | Mở rộng chuỗi theo độ dài bằng cách thêm các ký tự được chỉ định vào bên trái | LPAD(‘123′, 5, ’00’) | ‘00123’ |
| **LTRIM** | Xóa chuỗi dài nhất chứa các ký tự được chỉ định ở bên trái của chuỗi đầu vào string | LTRIM(‘00123’) | ‘123’ |
| **MD5** | Trả về hàm băm MD5 của một chuỗi ở dạng thập lục phân | MD5(‘ABC’) |  |
| **POSITION** | Trả về vị trí của một chuỗi con trong một chuỗi | POSITION(‘B’ in ‘A B C’) | 3 |
| **REGEXP\_MATCHES** | Thay thế các chuỗi con khớp với biểu thức chính quy POSIX bằng một chuỗi con mới | SELECT REGEXP\_MATCHES(‘ABC’, ‘^(A)(..)$’, ‘g’); | {A,BC} |
| **REGEXP\_REPLACE** | Thay thế chuỗi con bằng biểu thức chính quy. | REGEXP\_REPLACE(‘John Doe’,'(.\*) (.\*)’,’\2, \1′); | ‘Doe, John’ |
| **REPEAT** | Lặp lại một chuỗi theo số lần đã chỉ định. | REPEAT(‘\*’, 5) | ‘\*\*\*\*\*’ |
| **REPLACE** | Thay thế một chuỗi con trong một chuỗi bằng một chuỗi mới. | REPLACE(‘ABC’,’B’,’A’) | ‘AAC’ |
| **REVERSE** | thế một chuỗi con trong một chuỗi bằng một chuỗi mới | REVERSE(‘ABC’) | ‘CBA’ |
| **RIGHT** | thế một chuỗi con trong một chuỗi bằng một chuỗi mới | RIGHT(‘ABC’, 2) | ‘BC’ |
| **RPAD** | Kéo dài chuỗi theo độ dài nhất định bằng cách thêm các ký tự được chỉ định. | RPAD(‘ABC’, 6, ‘xo’) | ‘ABCxox’ |
| **RTRIM** | Xóa chuỗi dài nhất chứa các ký tự được chỉ định từ bên phải chuỗi đầu vào | RTRIM(‘abcxxzx’, ‘xyz’) | ‘abc’ |
| **SPLIT\_PART** | Chia một chuỗi theo một dấu phân cách được chỉ định và trả về chuỗi con thứ n | SPLIT\_PART(‘2017-12-31′,’-‘,2) | ’12’ |
| **SUBSTRING** | Trích xuất một chuỗi con từ một chuỗi | SUBSTRING(‘ABC’,1,1) | A’ |
| [**TRIM**](https://www.postgresqltutorial.com/postgresql-string-functions/postgresql-trim-function/) | Xóa các ký tự đầu và cuối khỏi một chuỗi | TRIM(‘ ABC ‘) | ‘ABC’ |
| **UPPER** | đổi một chuỗi thành chữ hoe | UPPER(‘hI tHERE’) | ‘HI THERE’ |