CREATE TABLE giangvien(

magv character(4) NOT NULL,

hoten character varying(30) NOT NULL,

diachi character varying(50) NOT NULL,

ngaysinh date NOT NULL,

CONSTRAINT khoachinhgiangvien PRIMARY KEY (magv)

);

CREATE TABLE detai(

madt character(4) NOT NULL,

tendt character varying(50) NOT NULL,

cap character varying(12) NOT NULL,

kinhphi integer,

CONSTRAINT khoachinhdetai PRIMARY KEY (madt)

);

CREATE TABLE thamgia(

magv character(4) NOT NULL,

madt character(4) NOT NULL,

sogio smallint,

CONSTRAINT khoachinhthamgia PRIMARY KEY (magv, madt),

CONSTRAINT khoangoai1 FOREIGN KEY (magv) REFERENCES giangvien (magv),

CONSTRAINT khoangoai2 FOREIGN KEY (madt) REFERENCES detai (madt)

);

INSERT INTO giangvien VALUES

('GV01','Vũ Tuyết Trinh','Hoàng Mai, Hà Nội','1975-10-10'),

('GV02','Nguyễn Nhật Quang','Hai Bà Trưng, Hà Nội','1976-11-03'),

('GV03','Trần Đức Khánh','Đống Đa, Hà Nội','1977-06-04'),

('GV04','Nguyễn Hồng Phương','Tây Hồ, Hà Nội','1983-12-10'),

('GV05','Lê Thanh Hương','Hai Bà Trưng, Hà Nội','1976-10-10');

INSERT INTO detai VALUES

('DT01','Tính toán lưới','Nhà nước',700),

('DT02','Phát hiện tri thức','Bộ',300),

('DT03','Phân loại văn bản','Bộ',270),

('DT04','Dịch tự động Anh Việt','Trường',30);

INSERT INTO thamgia VALUES

('GV01','DT01',100),

('GV01','DT02',80),

('GV01','DT03',80),

('GV02','DT01',120),

('GV02','DT03',140),

('GV03','DT03',150),

('GV04','DT04',180);

1. Giảng viên ở quận “Hai Bà Trưng”:

SELECT \* FROM giangvien

WHERE diachi LIKE '%Hai Bà Trưng%'

ORDER BY hoten DESC;

2. Đưa ra danh sách gồm họ tên, địa chỉ, ngày sinh của giảng viên có tham gia vào đề tài “Tính toán lưới”.

SELECT gv.hoten, gv.diachi, gv.ngaysinh

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

JOIN detai dt ON tg.madt = dt.madt

WHERE dt.tendt = N'Tính toán lưới';

3. Đưa ra danh sách gồm họ tên, địa chỉ, ngày sinh của giảng viên có tham gia vào đề tài “Phân loại văn bản” hoặc “Dịch tự động Anh Việt”.

SELECT gv.hoten, gv.diachi, gv.ngaysinh

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

JOIN detai dt ON tg.madt = dt.madt

WHERE dt.tendt IN (N'Phân loại văn bản', N'Dịch tự động Anh Việt');

4. Cho biết thông tin giảng viên tham gia ít nhất 2 đề tài.

SELECT gv.magv, gv.hoten

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

GROUP BY gv.magv, gv.hoten

HAVING COUNT(tg.madt) >= 2;

5. Cho biết tên giảng viên tham gia nhiều đề tài nhất.

SELECT gv.hoten

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

GROUP BY gv.hoten

ORDER BY COUNT(tg.madt) DESC

LIMIT 1;

6. Đề tài nào tốn ít kinh phí nhất?

SELECT madt, tendt

FROM detai

ORDER BY kinhphi ASC

LIMIT 1;

7. Cho biết tên và ngày sinh của giảng viên sống ở quận Tây Hồ và tên các đề tài mà giảng viên này tham gia.

SELECT gv.hoten, gv.ngaysinh, dt.tendt

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

JOIN detai dt ON tg.madt = dt.madt

WHERE gv.diachi LIKE '%Tây Hồ%';

8. Cho biết tên những giảng viên sinh trước năm 1980 và có tham gia đề tài “Phân loại văn bản”

SELECT gv.hoten

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

JOIN detai dt ON tg.madt = dt.madt

WHERE gv.ngaysinh < '1980-01-01' AND dt.tendt = N'Phân loại văn bản';

9. Đưa ra mã giảng viên, tên giảng viên và tổng số giờ tham gia nghiên cứu khoa học của từng giảng viên.

SELECT gv.magv, gv.hoten, SUM(tg.sogio) AS tonggiogiangday

FROM giangvien gv

JOIN thamgia tg ON gv.magv = tg.magv

GROUP BY gv.magv, gv.hoten;

10. Giảng viên Ngô Tuấn Phong sinh ngày 08/09/1986 địa chỉ Đống Đa, Hà Nội mới tham gia nghiên cứu đề tài khoa học. Hãy thêm thông tin giảng viên này vào bảng GiangVien.

INSERT INTO giangvien (magv, hoten, diachi, ngaysinh)

VALUES ('GVxx', 'Ngô Tuấn Phong', 'Đống Đa, Hà Nội', '1986-09-08');

11. Giảng viên Vũ Tuyết Trinh mới chuyển về sống tại quận Tây Hồ, Hà Nội. Hãy cập nhật thông tin này.

UPDATE giangvien

SET diachi = 'Tây Hồ, Hà Nội'

WHERE hoten = N'Vũ Tuyết Trinh';

12. Giảng viên có mã GV02 không tham gia bất kỳ đề tài nào nữa. Hãy xóa tất cả thông tin liên quan đến giảng viên này trong CSDL.

DELETE FROM thamgia WHERE magv = 'GV02';

DELETE FROM giangvien WHERE magv = 'GV02';

13. Đưa ra thông tin giảng viên có địa chỉ không ở quận “Hai Bà Trưng”, sắp xếp theo thứ tự giảm dần của họ tên.

SELECT \* FROM giangvien

WHERE diachi NOT LIKE N'%Hai Bà Trưng%'

ORDER BY hoten DESC;

Bài 1:

Một hệ thống cần quản lý:

- Thông tin về Giáo viên: GV# (mã giáo viên), HoTen (họ tên giáo viên, là một xâu không quá 30 ký tự), NamSinh (năm sinh, là một số nguyên), DiaChi (địa chỉ nơi ở của giáo viên, là một xâu không quá 50 ký tự).

- Thông tin về Đề tài: DT# (mã đề tài), TenDT (tên đề tài, là một xâu không quá 50 ký tự), TheLoai (thể loại của đề tài, là một xâu không quá 20 ký tự).

- Thông tin về Sinh viên: SV# (mã sinh viên), TenSV (tên sinh viên, là một xâu không quá 30 ký tự), NgaySinh (ngày tháng năm sinh của sinh viên), QueQuan (quê quán, là một xâu không quá 20 ký tự), Lop (lớp mà sinh viên học, là một xâu không quá 20 ký tự).

- Thông tin về Hướng dẫn: GV#, DT#, SV#, NamThucHien (năm thực hiện, là một số nguyên), KetQua (cho theo thang điểm 10, có lẽ đến 0.5). Trong một năm, mỗi một sinh viên chỉ thực hiện một đề tài dưới sự hướng dẫn của một hoặc nhiều giáo viên.

Yêu cầu:

1) Hãy vẽ sơ đồ thực thể liên kết với thông tin cho ở trên, sau đó chuyển sang mô hình quan hệ.

* Thực thể **Giáo viên** (GV): GV#, HoTen, NamSinh, DiaChi.
* Thực thể **Đề tài** (DT): DT#, TenDT, TheLoai.
* Thực thể **Sinh viên** (SV): SV#, TenSV, NgaySinh, QueQuan, Lop.
* Thực thể **Hướng dẫn**: GV#, DT#, SV#, NamThucHien, KetQua.

Liên kết giữa các thực thể này có thể được mô tả qua mối quan hệ giữa giáo viên và đề tài (giáo viên hướng dẫn đề tài), giữa sinh viên và đề tài (sinh viên thực hiện đề tài) và giữa giáo viên và sinh viên (giáo viên hướng dẫn sinh viên).

2) Hãy dùng ngôn ngữ định nghĩa dữ liệu SQL để tạo các bảng có được ở bước 1. Chú ý: định nghĩa đầy đủ các ràng buộc khóa chính, khóa ngoài.

CREATE TABLE GiaoVien (

GV# CHAR(5) PRIMARY KEY,

HoTen VARCHAR(30),

NamSinh INT,

DiaChi VARCHAR(50)

);

CREATE TABLE DeTai (

DT# CHAR(5) PRIMARY KEY,

TenDT VARCHAR(50),

TheLoai VARCHAR(20)

);

CREATE TABLE SinhVien (

SV# CHAR(5) PRIMARY KEY,

TenSV VARCHAR(30),

NgaySinh DATE,

QueQuan VARCHAR(20),

Lop VARCHAR(20)

);

CREATE TABLE HuongDan (

GV# CHAR(5),

DT# CHAR(5),

SV# CHAR(5),

NamThucHien INT,

KetQua FLOAT,

PRIMARY KEY (GV#, DT#, SV#),

FOREIGN KEY (GV#) REFERENCES GiaoVien(GV#),

FOREIGN KEY (DT#) REFERENCES DeTai(DT#),

FOREIGN KEY (SV#) REFERENCES SinhVien(SV#)

);

3) Dùng ngôn ngữ thao tác dữ liệu SQL để thể hiện các yêu cầu truy vấn sau:

a. Đưa ra thông tin về giáo viên có mã là "GV001".

SELECT \* FROM GiaoVien WHERE GV# = 'GV001';

b. Cho biết có bao nhiêu đề tài thuộc thể loại "Ứng dụng".

SELECT COUNT(\*) FROM DeTai WHERE TheLoai = 'Ứng dụng';

c. Cho biết giáo viên có mã "GV012" đã hướng dẫn bao nhiêu sinh viên có quê quán ở "Hải Phòng".

SELECT COUNT(\*) FROM HuongDan hd JOIN SinhVien sv ON hd.SV# = sv.SV#

WHERE hd.GV# = 'GV012' AND sv.QueQuan = 'Hải Phòng';

d. Cho biết tên của đề tài chưa có sinh viên nào thực hiện.

SELECT TenDT FROM DeTai

WHERE DT# NOT IN (SELECT DT# FROM HuongDan);

e. Do sơ xuất, thông tin về ngày sinh của sinh viên tên là "Nguyễn Xuân Dũng", quê quán "Hà Nam" đã bị nhập chưa chính xác. Ngày sinh chính xác là "12/11/1991". Hãy cập nhật thông tin này.

UPDATE SinhVien

SET NgaySinh = '1991-11-12'

WHERE TenSV = 'Nguyễn Xuân Dũng' AND QueQuan = 'Hà Nam';

f. Vì lý do khách quan, sinh viên "Lê Văn Luyện", quê quán "Bắc Giang" đã xin thôi học. Hãy xóa toàn bộ thông tin liên quan đến sinh viên này.

DELETE FROM HuongDan WHERE SV# IN (SELECT SV# FROM SinhVien WHERE TenSV = 'Lê Văn Luyện' AND QueQuan = 'Bắc Giang');

DELETE FROM SinhVien WHERE TenSV = 'Lê Văn Luyện' AND QueQuan = 'Bắc Giang';

CREATE TABLE Company (

CompanyID SERIAL PRIMARY KEY,

Name VARCHAR(40),

NumberofEmployee INT,

Address VARCHAR(50),

Telephone CHAR(15),

EstablishmentDay DATE

);

CREATE TABLE Product (

ProductID SERIAL PRIMARY KEY,

Name VARCHAR(40),

Color CHAR(14),

Price DECIMAL(10,2)

);

CREATE TABLE Supply (

CompanyID INT,

ProductID INT,

Quantity INT,

PRIMARY KEY(CompanyID, ProductID),

FOREIGN KEY (CompanyID) REFERENCES Company(CompanyID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID)

);

1. Find the information about the companies which located in London

SELECT \*

FROM Company

WHERE Address = 'London, UK';

2. Print the name, colour and price of the product that its color is 'black' and its price is above 5000

SELECT Name, Color, Price

FROM Product

WHERE Color = 'black' AND Price > 5000;

3. Find the name and telephone number of the companies that provided products whose color is 'red'

SELECT DISTINCT C.Name, C.Telephone

FROM Company C

JOIN Supply S ON C.CompanyID = S.CompanyID

JOIN Product P ON S.ProductID = P.ProductID

WHERE P.Color = 'red';

4. Find the company which provided both products of 'blue' and 'black' color.

SELECT C.CompanyID

FROM Company C

JOIN Supply S ON C.CompanyID = S.CompanyID

JOIN Product P ON S.ProductID = P.ProductID

WHERE P.Color IN ('blue', 'black')

GROUP BY C.CompanyID

HAVING COUNT(DISTINCT P.Color) = 2;

5. Print the name of the item which is the most expensive.

SELECT Name

FROM Product

ORDER BY Price DESC

LIMIT 1;

6. Print the company identity which provided at least two products

SELECT CompanyID

FROM Supply

GROUP BY CompanyID

HAVING COUNT(DISTINCT ProductID) >= 2;

7. Print the name of the company that provided all 'blue' products

SELECT C.Name

FROM Company C

WHERE NOT EXISTS (

SELECT 1

FROM Product P

WHERE P.Color = 'blue' AND NOT EXISTS (

SELECT 1

FROM Supply S

WHERE S.ProductID = P.ProductID AND S.CompanyID = C.CompanyID

)

);

8. Find the name of the company which deliverred the most products

SELECT CompanyID

FROM Supply

GROUP BY CompanyID

ORDER BY COUNT(ProductID) DESC

LIMIT 1;

9. Give information about the identity, the name of the company and the average number of items that the company provided, order by the average number descending.

SELECT C.CompanyID, C.Name, AVG(S.Quantity) AS AverageQuantity

FROM Company C

JOIN Supply S ON C.CompanyID = S.CompanyID

GROUP BY C.CompanyID

ORDER BY AVG(S.Quantity) DESC;

10. Create a new table has the same content with Product table using "select into" statement.

SELECT \*

INTO NewProductTable

FROM Product;

11. The company whose identity is '1' moved its headquarter to a new place 'Hanoi, Vietnam'. Please update this information.

UPDATE Company

SET Address = 'Hanoi, Vietnam'

WHERE CompanyID = 1;

12. The company whose identity is '14' went into bankruptcy. Remove all information about this company and its distribution from the database.

DELETE FROM Supply

WHERE CompanyID = 14;

DELETE FROM Company

WHERE CompanyID = 14;

13. Write the statement to back-up this database.

pg\_dump dbname > outfile

Index:

Trong PostgreSQL, một index (chỉ mục) là một cấu trúc dữ liệu giúp cải thiện tốc độ truy vấn trên bảng dữ liệu. Index hoạt động bằng cách lưu trữ một phần của dữ liệu bảng theo một cách mà nó có thể truy xuất nhanh chóng. Các loại index phổ biến trong PostgreSQL bao gồm B-tree, Hash, GiST, SP-GiST và GIN.

**Ví dụ Minh Họa:**

Giả sử chúng ta có bảng **students** với cấu trúc như sau:

* **id** (khóa chính)
* **name**
* **age**
* **grade**

1. Tạo Index B-tree:

Đây là loại index thông dụng nhất, thích hợp cho các truy vấn so sánh bằng (=), lớn hơn (>) và nhỏ hơn (<).

CREATE INDEX idx\_name ON students (name);

Trong ví dụ này, chúng ta tạo một index kiểu B-tree cho cột **name** trong bảng **students**. Điều này giúp tăng tốc độ truy vấn dữ liệu khi tìm kiếm theo tên học sinh.

2. Sử Dụng Index trong Truy Vấn:

Sau khi tạo index, nếu thực hiện một truy vấn tìm kiếm theo **name**, PostgreSQL sẽ tự động sử dụng index để tối ưu hóa truy vấn.

SELECT \* FROM students WHERE name = 'John Doe';

Trong truy vấn này, PostgreSQL sẽ sử dụng **idx\_name** để nhanh chóng tìm kiếm thông tin của học sinh có tên là 'John Doe'.

**Lưu Ý:**

* Việc sử dụng index có thể làm tăng tốc độ truy vấn đáng kể, nhất là đối với bảng dữ liệu lớn.
* Tuy nhiên, mỗi index cũng chiếm không gian lưu trữ và có thể làm chậm quá trình cập nhật, thêm mới hoặc xóa bỏ dữ liệu trên bảng.
* Việc lựa chọn và quản lý index cần phải cân nhắc cả lợi ích và chi phí đi kèm.

View:

Trong PostgreSQL, một **view** là một truy vấn được lưu trữ trong cơ sở dữ liệu dưới dạng một bảng ảo. View không lưu trữ dữ liệu thực tế; nó chỉ chứa một truy vấn SQL mà bạn có thể thực hiện trên một hoặc nhiều bảng. Views thường được sử dụng để:

1. Đơn giản hóa truy vấn phức tạp.
2. Cung cấp một giao diện đơn giản hóa cho người dùng hoặc ứng dụng.
3. Bảo vệ dữ liệu bằng cách chỉ hiển thị thông tin cần thiết và ẩn đi thông tin nhạy cảm.

**Ví dụ:**

Giả sử bạn có hai bảng: **students** (chứa thông tin của học sinh) và **grades** (chứa điểm số của học sinh). Cấu trúc của hai bảng này có thể như sau:

* **students**
  + id (int)
  + name (varchar)
  + class (varchar)
* **grades**
  + id (int)
  + student\_id (int, khóa ngoại tham chiếu đến students.id)
  + subject (varchar)
  + grade (int)

Tạo View:

Bạn muốn tạo một view để hiển thị tên học sinh cùng với điểm số môn học của họ mà không cần truy vấn trực tiếp hai bảng.

CREATE VIEW student\_grades AS SELECT students.name, grades.subject, grades.grade FROM students JOIN grades ON students.id = grades.student\_id;

Sử Dụng View:

Sau khi tạo view, bạn có thể truy vấn nó giống như một bảng thông thường:

SELECT \* FROM student\_grades;

Truy vấn này sẽ trả về danh sách tên học sinh, môn học và điểm số của họ.

**Lợi Ích:**

* **An Toàn Dữ Liệu**: Bạn có thể giới hạn thông tin hiển thị trong view, giúp bảo vệ dữ liệu nhạy cảm.
* **Dễ Quản Lý**: Thay đổi truy vấn trong view không ảnh hưởng đến cấu trúc bảng gốc.
* **Tối Ưu Hóa Truy Vấn**: View cho phép bạn tối ưu hóa truy vấn phức tạp và tái sử dụng chúng mà không cần viết lại truy vấn.

**Hạn Chế:**

* **Không Lưu Trữ Dữ Liệu**: View không lưu trữ dữ liệu thực tế, mà chỉ là kết quả của truy vấn.
* **Hiệu Suất**: Truy vấn trên view có thể chậm hơn so với truy vấn trực tiếp trên bảng, đặc biệt với view phức tạp trên bảng dữ liệu lớn.

Views trong PostgreSQL cung cấp một cách linh hoạt và mạnh mẽ để quản lý truy vấn và trình bày dữ liệu.

Stored Procedure và Trigger

Trong PostgreSQL, "Stored Procedure" và "Trigger" là hai khái niệm quan trọng giúp tối ưu hóa và tự động hóa các tác vụ liên quan đến cơ sở dữ liệu.

1. Stored Procedure (Thủ tục lưu trữ):

Stored Procedure là một tập hợp các câu lệnh SQL được lưu trữ trong cơ sở dữ liệu. Nó cho phép bạn thực hiện các tác vụ phức tạp, tái sử dụng mã và tăng tính bảo mật.

Ví dụ:

Giả sử bạn muốn tạo một stored procedure để cập nhật điểm của học sinh:

CREATE OR REPLACE PROCEDURE update\_grade(student\_id int, new\_grade int)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE grades

SET grade = new\_grade

WHERE student\_id = student\_id;

END;

$$;

Để gọi thủ tục này:

CALL update\_grade(1, 90);

Điều này sẽ cập nhật điểm của học sinh có student\_id = 1 thành 90.

2. Trigger (Kích hoạt):

Trigger trong PostgreSQL là một quy tắc được định nghĩa để tự động thực hiện hoặc gọi một stored procedure khi có các thay đổi nhất định (như INSERT, UPDATE, DELETE) xảy ra trên bảng.

Ví dụ:

Giả sử bạn muốn ghi lại mọi thay đổi điểm số của học sinh vào một bảng grade\_changes.

Đầu tiên, tạo bảng grade\_changes:

CREATE TABLE grade\_changes (

change\_id serial PRIMARY KEY,

student\_id int NOT NULL,

old\_grade int,

new\_grade int,

change\_time timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

Tiếp theo, tạo trigger:

CREATE OR REPLACE FUNCTION record\_grade\_change()

RETURNS trigger AS $$

BEGIN

IF NEW.grade != OLD.grade THEN

INSERT INTO grade\_changes(student\_id, old\_grade, new\_grade)

VALUES (NEW.student\_id, OLD.grade, NEW.grade);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER grade\_change\_trigger

AFTER UPDATE ON grades

FOR EACH ROW

EXECUTE FUNCTION record\_grade\_change();

Trigger này sẽ được kích hoạt sau mỗi lần cập nhật điểm trong bảng grades, và nếu có thay đổi về điểm số, nó sẽ ghi nhận thay đổi đó vào bảng grade\_changes.

Kết Luận:

Stored Procedure: Hữu ích cho việc đóng gói logic phức tạp, tái sử dụng mã và tăng tính bảo mật.

Trigger: Tự động hóa các tác vụ dựa trên sự kiện cụ thể trong cơ sở dữ liệu, giúp đồng bộ hóa dữ liệu hoặc ghi nhận các hoạt động quan trọng.

Function:

Trong PostgreSQL, một Function (hàm) là một tập hợp các câu lệnh SQL và các lệnh lập trình khác, được lưu trữ trong cơ sở dữ liệu. Functions có thể trả về một giá trị, và chúng thường được sử dụng để thực hiện các tác vụ tính toán, xử lý dữ liệu, và các logic phức tạp trong cơ sở dữ liệu.

Ví dụ Minh Họa:

Giả sử bạn muốn tạo một function để tính tuổi trung bình của các học sinh trong một lớp.

1. Tạo Bảng Dữ Liệu:

Đầu tiên, tạo một bảng students để lưu trữ thông tin của học sinh:

CREATE TABLE students (

id serial PRIMARY KEY,

name varchar(100),

birth\_date date,

class varchar(50)

);

2. Thêm Dữ Liệu Mẫu:

Thêm một số dữ liệu mẫu vào bảng students:

INSERT INTO students (name, birth\_date, class) VALUES ('Alice', '2005-04-12', '10A');

INSERT INTO students (name, birth\_date, class) VALUES ('Bob', '2006-05-15', '10A');

-- Thêm thêm một số học sinh khác

3. Tạo Function Tính Tuổi Trung Bình:

Tạo một function để tính tuổi trung bình của học sinh trong một lớp cụ thể:

CREATE OR REPLACE FUNCTION average\_age(class varchar)

RETURNS float AS $$

DECLARE

total\_age int;

count\_students int;

BEGIN

SELECT SUM(EXTRACT(YEAR FROM age(birth\_date))) INTO total\_age FROM students WHERE class = $1;

SELECT COUNT(\*) INTO count\_students FROM students WHERE class = $1;

IF count\_students = 0 THEN

RETURN 0;

ELSE

RETURN total\_age / count\_students;

END IF;

END;

$$ LANGUAGE plpgsql;

4. Sử Dụng Function:

Gọi function để tính tuổi trung bình cho lớp '10A':

SELECT average\_age('10A');

Kết Luận:

Function trong PostgreSQL cho phép bạn đóng gói các tác vụ phức tạp thành một đơn vị logic đơn giản, có thể tái sử dụng. Điều này giúp việc quản lý và xử lý dữ liệu trở nên dễ dàng và hiệu quả hơn. Functions có thể được viết bằng nhiều ngôn ngữ khác nhau, bao gồm PL/pgSQL (một ngôn ngữ lập trình mở rộng của SQL được sử dụng trong PostgreSQL), C, Python, và nhiều ngôn ngữ khác.