

MỤC LỤC

Chương 1. TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU KHÔNG GIAN.....	7
1.1. Tổng quan về cơ sở dữ liệu không gian.....	7
1.1.1. CSDL không gian.....	7
1.1.2. Đặc trưng của CSDL không gian.....	7
1.2. Mô hình.....	8
1.2.1. POINT.....	8
1.2.2. LINE.....	8
1.2.3. POLYGON.....	9
1.3. Mối quan hệ không gian là gì?.....	9
1.3.1. Phân loại.....	9
1.3.2. Kết hợp hình học vào mô hình dữ liệu DBMS.....	10
Chương 2. POSTGRESQL VÀ POSTGIS.....	12
2.1. PostgreSQL.....	12
2.1.1. Định nghĩa.....	12
2.1.2. So sánh PostgreSQL với một số hệ cơ sở quản trị dữ liệu khác.....	12
2.1.3. Quản trị cơ sở dữ liệu qua giao diện.....	15
2.2. PostGIS.....	23
2.2.1. Giới thiệu về PostGIS.....	23
2.2.2. Công cụ shp2pgsql.....	24
2.2.3. Công cụ psql.....	25
2.2.4. Phương pháp load dữ liệu định dạng file .sql.....	25
2.2.5. Phương pháp load dữ liệu dạng shape file vào CSDL.....	26
2.2.6. OpenGIS Well-Know Text.....	27
2.2.7. Bảng siêu dữ liệu.....	28
2.2.8. Bảng không gian.....	30
2.3. Hàm trong PostGIS.....	32
2.3.1. Nhóm hàm điều khiển.....	32
2.3.2. Nhóm hàm khởi tạo hình học.....	33
2.3.3. Hàm trả về kiểu hình học ở đầu ra.....	34
2.3.4. Hàm xác định mối quan hệ không gian.....	34
2.3.5. Nhóm hàm đưa ra đối tượng hình mới.....	40
2.3.6. Nhóm hàm thay đổi hình học.....	42
2.3.7. Nhóm hàm accessor.....	44
2.4. Chỉ mục.....	45
2.4.1. Chỉ mục GiST.....	45
2.4.2. Sử dụng chỉ mục.....	45
2.5. Truy vấn trong cơ sở dữ liệu không gian.....	47
2.5.1. Mô tả về cơ sở dữ liệu không gian.....	47

2.5.2. Truy vấn	49
Chương 3. MỞ RỘNG TRUY VẤN KHÔNG GIAN POSTGRESQL	54
3.1. Các kiểu dữ liệu trong PostgreSQL	54
3.1.1. Kiểu dữ liệu cơ bản	54
3.1.2. Kiểu dữ liệu hỗn hợp	55
3.2. Mở rộng PostgreSQL với hàm tùy chọn.....	55
3.2.1. Hàm ngôn ngữ truy vấn (SQL)	55
3.2.2. Hàm sử dụng ngôn ngữ lập trình C	59
3.2.3. Kiểu dữ liệu do người dùng định nghĩa.....	67
3.2.4. Toán tử do người dùng định nghĩa.....	71
3.2.5. Hàm tập hợp cho người dùng định nghĩa.....	73
3.3. Viết hàm mở rộng cho PostgreSQL.....	74
TỔNG KẾT	82

LỜI NÓI ĐẦU

Ngày nay, cùng với sự phát triển của xã hội, ngành CNTT cũng có nhiều bước phát triển đáng kể và đã dần đi vào cuộc sống của mọi người và được sử dụng hầu hết trong tất cả các ngành nghề trong xã hội. Với số lượng tài liệu trong các cơ quan, tổ chức tăng theo cấp số nhân theo từng năm, từng thời kỳ, thì việc lưu trữ số lượng tài liệu đó trở nên vô cùng khó khăn, đặc biệt khi nó là những tài liệu quan trọng mà lại được lưu trữ trên các thiết bị cứng. Yếu tố thời gian, các tác động bên ngoài có thể làm cho những tài liệu đó bị hỏng hóc, khó bảo quản. Do đó, nhu cầu sử dụng các phần mềm hỗ trợ khả năng lưu trữ các dữ liệu đảm bảo các yếu tố an toàn và tiện lợi trong thao tác với dữ liệu đó là vô cùng cần thiết. Và nhu cầu đó sẽ trở nên dễ dàng khi có sự vào cuộc của CNTT, đặc biệt là các hệ quản trị cơ sở dữ liệu (CSDL).

Nói đến CNTT thì không thể không nói đến các hệ quản trị cơ sở dữ liệu. Đó là phần mềm hay hệ thống được thiết kế để quản trị một CSDL, nó hỗ trợ khả năng lưu trữ, sửa chữa, xóa và tìm kiếm trong tin trong một CSDL. Và có rất nhiều loại hệ quản trị CSDL khác nhau : từ phần mềm nhỏ chạy trên máy tính cá nhân cho đến những hệ quản trị phức tạp chạy trên một hoặc nhiều siêu máy tính. Chúng ta có thể kể tới các hệ quản trị CSDL như: MySQL, Oracle, SQL Server, PostgreSQL...và mỗi loại trên có những tính năng, lợi ích riêng.

Đặc biệt, hệ quản trị CSDL postgresQL có những tính năng và lợi thế hơn hẳn các hệ quản trị CSDL khác. PostgreSQL là sự lựa chọn sử dụng của nhiều người vì nó có nhiều ưu điểm nổi trội so với các hệ quản trị CSDL khác. Thứ nhất, PostgreSQL là phần mềm mã nguồn mở, miễn phí hoàn toàn trong sử dụng. Thứ hai, hiệu suất làm việc của PostgreSQL chênh lệch so với các hệ quản trị khác trong sai số +/-10%. Thứ ba, đây là hệ quản trị có độ tin cậy cao, bằng chứng là quá trình phát triển của nó. Thứ tư, PostgreSQL còn có thể chạy được trên rất nhiều hệ điều hành khác nhau như Window, Linux, Unix, MacOSX...Và cuối cùng, một tính năng nổi trội của PostgreSQL là khả năng mở rộng hàm, kiểu dữ liệu, toán tử...người sử dụng có thể tự định nghĩa hàm, kiểu dữ liệu, kiểu toán tử...và có thể thêm những kiểu dữ liệu, toán tử...vào hệ quản trị CSDL PostgreSQL.

Ngoài ra, do ngoài nhu cầu lưu trữ các kiểu dữ liệu thông thường như kiểu chuỗi, kiểu số, kiểu ngày tháng, người sử dụng còn có thêm nhu cầu lưu trữ các kiểu dữ liệu không gian để lưu trữ các đối tượng như Point, Line, Polygon. Do đó, PostgreSQL còn hỗ

trợ kiểu dữ liệu hình học (geometry) như Point, Line, Polygon... Và PostGIS chính là công cụ được bổ sung cho PostgreSQL để hỗ trợ hiện thị đối tượng địa lý. Nhờ PostGIS, khả năng không gian trong PostgreSQL được kích hoạt, nó cho phép PostgreSQL sử dụng như một CSDL không gian phục vụ cho các hệ thống thông tin địa lý.

PostGIS là một mã nguồn mở, mở rộng không gian cho PostgreSQL. CSDL không gian trong PostGIS được sử dụng cho hiệu suất sử dụng cao đa người dùng truy cập đến tập dữ liệu có tính liên mạch. Nếu bạn quản lý số lượng lớn đọc/ghi dữ liệu không gian, thì việc sử dụng CSDL không gian có thể cải thiện được tốc độ truy cập, dễ dàng quản lý và đảm bảo tính toàn vẹn dữ liệu. Được xây dựng như phần mở rộng đối tượng cho PostgreSQL, PostGIS đã được chứng nhận là “Simple Features for SQL”, tuân thủ theo Open Geospatial Consortium. PostGIS được phát hành lần đầu tiên vào năm 2001, và hiện đang được sử dụng trên khắp thế giới như một máy chủ hoạt động với hiệu suất cao cho các đối tượng không gian.

PostGIS cung cấp việc tạo và thao tác trên CSDL không gian. CSDL không gian cũng là CSDL thông thường, nhưng nó bổ sung thêm các kiểu dữ liệu không gian và các mối quan hệ giữa các kiểu dữ liệu đó. Một CSDL không gian bao gồm rất nhiều bảng dữ liệu không gian, ngoài các thuộc tính có kiểu dữ liệu thông thường thì bảng không gian còn chứa một thuộc tính có kiểu dữ liệu không gian mô tả về một đối tượng thực trong thực tế.

Truy vấn không gian là gì? Là các câu lệnh truy vấn được thực hiện trên bảng không gian trong CSDL để tìm ra mối quan hệ giữa các đối tượng trong không gian, mối quan hệ đó có thể là sự giao nhau, tính khoảng cách, tính diện tích, tính chu vi, tính chiều dài... và các câu lệnh truy vấn được viết ra dễ dàng hơn nhờ các hàm hỗ trợ sẵn của PostGIS. PostGIS cung cấp các nhóm hàm để hỗ trợ việc truy vấn như nhóm hàm xác định mối quan hệ không gian, nhóm hàm trả về đối tượng mới...nhờ đó, việc thực hiện truy vấn trong không gian sẽ trở nên dễ dàng và dễ thao tác hơn.

Với những ưu điểm nội trời đó, hệ quản trị CSDL PostgreSQL xứng đáng là lựa chọn của nhiều người sử dụng, đặc biệt với sự hỗ trợ của công cụ mở rộng PostGIS, việc lưu trữ các đối tượng không gian không còn khó khăn nữa.

Trong phạm vi nghiên cứu của đề tài, chúng em tập trung vào việc nghiên cứu các vấn đề sau :

Thứ nhất, nghiên cứu về CSDL không gian, qua đó, giúp chúng ta có cái nhìn tổng quan về CSDL không gian,

Thứ hai, tìm hiểu tổng quan về hệ quản trị PostgreSQL, qua đó, chúng ta có thể biết được ưu, nhược điểm của hệ quản trị này so với các hệ quản trị CSDL khác. Ngoài ra, phần giới thiệu về giao diện tương tác giúp ích cho việc thao tác với hệ quản trị PostgreSQL được dễ dàng

Thứ ba, PostGIS là công cụ mở rộng cho PostgreSQL, nó giúp PostgreSQL lưu trữ, thao tác được với CSDL không gian. Chúng tôi có giới thiệu về cách tạo CSDL không gian, cách load dữ liệu không gian có sẵn vào CSDL. Ngoài ra, chúng tôi còn cung cấp danh sách các nhóm hàm mà PostGIS hỗ trợ sẵn, làm công cụ cho việc thực hiện truy vấn trong không gian.

Và cuối cùng, chúng tôi có đưa ra nội dung về cách tạo mở rộng hàm, mở rộng kiểu dữ liệu, mở rộng kiểu toán tử và mở rộng hàm tập hợp cho hệ quản trị CSDL PostgreSQL. Từ đó người dùng biết cách tạo phần mở rộng cho PostgreSQL cho mục đích sử dụng của mình.

DANH SÁCH CÁC BẢNG

Bảng 2-1: So sánh về hệ điều hành hỗ trợ	14
Bảng 2-2: So sánh về các tính năng cơ bản	14
Bảng 2-3: So sánh về sự hỗ trợ bảng tạm và khung nhìn	14
Bảng 2-4: So sánh về chức năng đánh chỉ mục	15
Bảng 2-5: So sánh về các đối tượng khác	15
Bảng 2-6: Danh sách các tùy chọn của psql	16
Bảng 2-7: Nhóm lệnh chung của psql	18
Bảng 2-8: Nhóm lệnh truy vấn bộ đệm của psql	18
Bảng 2-9: Nhóm lệnh vào / ra của lệnh psql	18
Bảng 2-10: Nhóm lệnh trả về thông tin	18
Bảng 2-11: Nhóm lệnh định dạng của psql	19
Bảng 2-12: Danh sách lệnh \h	20
Bảng 2-13: Danh sách các tùy chọn của sph2pgsql	23
Bảng 2-14: Các ví dụ minh họa cho hàm ST_Buffer()	42
Bảng 3-1: Danh sách kiểu dữ liệu trong SQL và trong C	58

DANH SÁCH CÁC HÌNH

Hình 1-1: Mô hình đối tượng LINE	11
Hình 1-2: Mô hình đối tượng POLYGON	11
Hình 2-1: Minh họa hàm ST_Touches()	35
Hình 2-2: Minh họa hàm ST_Within()	36
Hình 2-3: Minh họa hàm ST_Contains()	37
Hình 2-4: Minh họa hàm ST_Difference()	39
Hình 2-5: Minh họa hàm ST_Union()	40

DANH SÁCH CÁC TỪ VIẾT TẮT

CSDL : Cơ sở dữ liệu

Chương 1. TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU KHÔNG GIAN

1.1. Tổng quan về cơ sở dữ liệu không gian

1.1.1. CSDL không gian

Là một hệ thống CSDL quan hệ :

- 1). Cung cấp các kiểu dữ liệu không gian trong mô hình dữ liệu và các ngôn ngữ truy vấn
- 2). Hỗ trợ các kiểu dữ liệu không gian trong việc thực thi chính nó, cung cấp các kiểu đánh chỉ mục để thực thi truy vấn nhanh nhất từ bảng dữ liệu lớn.

Giải thích :

- 1). Các kiểu dữ liệu không gian như Point, Line, Polygon. CSDL không gian cung cấp cung cấp mô hình trừu tượng cơ bản cho cấu trúc của thực thể hình học trong không gian cũng như mối quan hệ giữa chúng như quan hệ giao nhau (intersects(a, b)), thuộc tính như diện tích, chu vi của mô hình (area(a) hay perimeter(a)), hoặc tìm điểm giao giữa 2 mô hình (intersection(a.b)).
- 2). Việc đánh chỉ mục cho dữ liệu là vô cùng quan trọng, nó giúp ích cho việc tối ưu hóa truy vấn dữ liệu, giảm thời gian truy vấn, giảm bộ nhớ lưu trữ...

1.1.2. Đặc trưng của CSDL không gian

- Cơ sở dữ liệu không gian sử dụng đánh chỉ mục không gian để tăng tốc hoạt động của cơ sở dữ liệu
- Ngoài các truy vấn SQL điển hình như câu lệnh SELECT, CSDL không gian có thể thực thi đa dạng các thao tác không gian. Và nó được hỗ trợ bởi OGC :
 - Đo lường không gian : nó có khả năng tìm khoảng cách giữa các điểm, các vùng...
 - Hàm không gian : ví dụ như, sửa đổi các hàm hiện thời để tạo ra những hình mới : hàm tìm điểm hay vùng giao nhau...
 - Xác nhận không gian : nó cho phép thực hiện những truy vấn True/False.
 - Hàm tạo : tạo ra các hình mới, như chỉ ra các điểm nút có thể tạo nên đường, hay nếu đỉnh đầu và đỉnh cuối trùng nhau, chúng có thể tạo nên một đa giác.

- Hàm theo dõi : các câu truy vấn trả về thông tin cụ thể như : vị trí tâm của một đường tròn hay điểm đầu, điểm cuối của một đường

1.2. Mô hình

Có hai đối tượng quan trọng cần được hiển thị đó là :

1). Đối tượng trong không gian : đó là những đối tượng trong không gian, mô tả hình học của riêng chúng

2). Không gian

Đối tượng đơn: đối tượng cơ bản là Point, Line, Polygon

1.2.1. POINT

- Định nghĩa : hiển thị một đối tượng mà chỉ có vị trí của nó trong không gian.

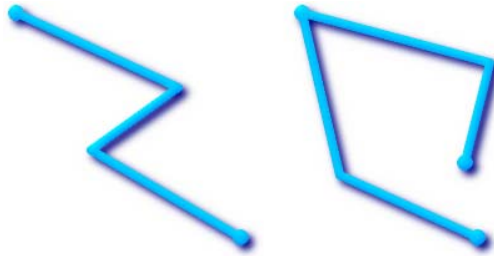
VD : một thành phố có thể được mô phỏng như 1 điểm trong mô hình mô tả 1 khu vực rộng lớn về địa lý.

- Đặc điểm :
 - + Là tọa độ đơn.
 - + Không cần thể hiện chiều dài và diện tích
 - + Điểm được sử dụng để hiển thị cho các vùng khi chúng được hiển thị ở quy mô nhỏ.
 - + Không có phép đo nào được áp dụng cho điểm.

1.2.2. LINE

- Định nghĩa : được xác định như một tập hợp dãy các điểm, mô tả đối tượng địa lý dạng tuyến tính.
- Đặc điểm :
 - + Là một dãy các cặp tọa độ.
 - + Bắt đầu và kết thúc là một điểm.
 - + Các đường nối với nhau hoặc cắt nhau tại một điểm.
 - + Hình dạng của đường được định nghĩa bởi tọa độ của điểm.

- + Cũng như tính năng của điểm, đường cũng được hiển thị ở quy mô nhỏ hiển thị là đường đơn là một đa giác.
- + Có phép đo khoảng cách đối với đường.



Hình 1-1 : Mô hình đối tượng LINE

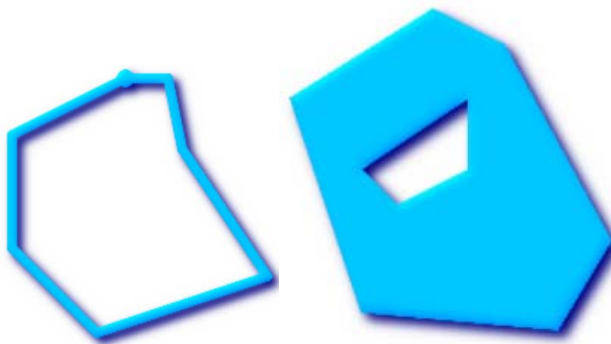
1.2.3. POLYGON

- Định nghĩa : được xác định bởi ranh giới các đường thẳng. Các đối tượng địa lý có diện tích và đóng kín bởi một đường được gọi là vùng.

VD : hồ, các toà nhà, công viên, thành phố...

- Đặc điểm :

- + Vùng được mô tả bằng tập các đường và điểm
- + Một hoặc nhiều đường là đường bao của vùng.
- + Có phép tính chu vi và diện tích cho đa giác.



Hình 1-2 : Mô hình đối tượng POLYGON

1.3. Môi quan hệ không gian là gì?

Mối quan hệ không gian chỉ ra mối quan hệ giữa các đối tượng trong không gian

1.3.1. Phân loại

Có 3 mối quan hệ không gian phổ biến nhất đó là :

- **Quan hệ topo** : như liền kề, phân chia...và các phép biến đổi topo như phép dịch chuyển, phép xoay...

- **Quan hệ định hướng**

Quan hệ định hướng có thể phân làm 2 loại : quan hệ định hướng bên ngoài và quan hệ định hướng bên trong. Quan hệ định hướng bên trong chỉ ra một đối tượng được đặt bên trong một đối tượng tham chiếu, còn quan hệ định hướng bên ngoài chỉ ra đối tượng được đặt bên ngoài một đối tượng tham chiếu.

- **Quan hệ khoảng cách**

Chỉ ra khoảng cách từ đối tượng cụ thể đến đối tượng tham chiếu.

1.3.2. Kết hợp hình học vào mô hình dữ liệu DBMS

Ý tưởng chính của việc kết hợp các mô hình hình học vào trong mô hình dữ liệu DBMS để thể hiện các “đối tượng không gian”- các đối tượng có thể là dòng sông, đất nước, thành phố...bằng các đối tượng hình học trước tiên là thuộc tính của loại dữ liệu không gian. Về cơ bản, mô hình dữ liệu DBMS luôn hỗ trợ sẵn các kiểu dữ liệu như integer, string... hoặc có thể là kiểu dữ liệu do người dùng định nghĩa. Ngoài ra, với CSDL không gian, mô hình dữ liệu DBMS còn hỗ trợ một số kiểu khác như kiểu hình học như kiểu Point, kiểu Line...

VD: Mô tả đặc điểm của sông, hay mô tả đặc điểm của thành phố ta có các bảng dữ liệu:

Rivers (rname : STRING, route : LINE)

Cities (cname : STRING, center : POINT, ext : POLYGON, cpop : INTEGER)

Nếu để ý 2 bảng dữ liệu cities và rivers, ngoài kiểu dữ liệu thông thường như STRING và INTEGER, còn có kiểu dữ liệu hình học như LINE, POINT, POLYGON. Đúng như mô tả của từng kiểu đối tượng LINE, POINT, POLYGON.

Để biểu diễn các đối tượng không gian trong mô hình 2 chiều, cách thông thường là sử dụng cách biểu diễn hệ tọa độ.

VD : biểu diễn một điểm, POINT (0,0) : điểm nằm tại tọa độ (0,0)

Biểu diễn một đường LINE (0 0, 1 1, 1 2) : đường nối 3 điểm nằm lần lượt tại các tọa độ (0,0) -> (1,1) -> (1, 2)

Chương 2. POSTGRESQL VÀ POSTGIS

2.1. PostgreSQL

2.1.1. Định nghĩa

Vào năm 1986, giáo sư Đại học California ở Berkeley và chuyên gia công nghệ về cơ sở dữ liệu Michael Stonebraker đã đưa ra vấn đề là phải xây dựng hệ thống cơ sở dữ liệu tốt hơn. Mặc dù đã có những thành công với dự án cơ sở dữ liệu trước đó, do INGRES nghiên cứu ra, Stonebraker đã quyết định phát triển lên dựa trên nền tảng đã có. Và kết quả của sự phát triển đó là Postgres.

Trong 8 năm tiếp đó, POSTGRES đã phát triển một cách phổ biến, đặc biệt là trong cộng đồng nghiên cứu. Qua một quá trình phát triển lâu dài, bản PostgreSQL 6.0 được chính thức ra đời nó dựa trên nền tảng của POSTGRES trước đó và thêm vào các thực thi SQL. Ngày nay, PostgreSQL là một trong những dự án nguồn mở phổ biến nhất trên Internet.

PostgreSQL là hệ thống quản trị cơ sở dữ liệu quan hệ đối tượng dựa trên POSTGRES bản 4.2, được phát triển tại trường đại học California tại phòng nghiên cứu máy tính Berkeley. [1]. Nó là một chương trình mã nguồn mở xây dựng trên mã nguồn ban đầu của đại học Berkeley. Nó hỗ trợ một phần rất lớn cho SQL chuẩn và cung cấp nhiều tính năng hiện đại như :

- Các truy vấn phức tạp
- Khóa ngoài
- Trigger
- Khung nhìn
- Tính toàn vẹn của các giao dịch
- Kiểm tra truy cập đồng thời đa phiên bản.

Ngoài ra, PostgreSQL có thể được mở rộng bởi nhiều người dùng bằng nhiều cách, ví dụ, người dùng có thể thêm kiểu dữ liệu, hàm, toán tử, hàm tập hợp, phương thức đánh chỉ mục và ngôn ngữ thủ tục.

2.1.2. So sánh PostgreSQL với một số hệ cơ sở quản trị dữ liệu khác

Việc so sánh hệ quản trị PostgreSQL với một số hệ quản trị cơ sở dữ liệu khác giúp chúng ta có cái nhìn tổng quan về ưu, nhược điểm của hệ quản trị PostgreSQL. Thông tin được đưa ra so sánh như : hệ điều hành hỗ trợ, các tính năng cơ bản, hỗ trợ bảng và khung nhìn, chức năng chỉ mục, và các đối tượng khác.

a. Hệ điều hành hỗ trợ

Bảng 2-1 : So sánh về hệ điều hành hỗ trợ

	Windows	Mac OS X	Linux	BSD	UNIX	z/OS ^{[[fn_11]]}
MySQL	Có	Có	Có	Có	Có	Có thể
Oracle	Có	Có	Có	Không	Có	Có
PostgreSQL	Có	Có	Có	Có	Có	Có thể

b. Các tính năng cơ bản

Bảng 2-2 : So sánh về các tính năng cơ bản.

	ACID	Referential integrity	Transactions	Unicode
MySQL	Phụ thuộc ^{[[fn_313]]}	Phụ thuộc ^{[[fn_313]]}	Phụ thuộc ^{[[fn_313]]}	Có / UTF-8 (3-byte) & UCS-2
Oracle	Có	Có	Có	Có
PostgreSQL	Có	Có	Có	Có / UTF-8 (4-byte)

c. Hỗ trợ bảng và khung nhìn

Bảng 2-3 : So sánh về sự hỗ trợ bảng tạm và khung nhìn

	Bảng tạm	Khung nhìn cụ thể
MySQL	Có	Tương tự ^{[[fn_66]]}
Oracle	Có	Có
PostgreSQL	Có	Tương tự ^{[[fn_77]]}

d. Chỉ mục

Bảng 2-4 : So sánh về chức năng chỉ mục

	Cây R-/Cây R+	Hàm băm	Biểu thức (dập trình)	Chỉ mục từng phần	Chỉ mục đảo	Bitmap	GIST
MySQL	trong SQL 5.0 MyISAM, BDB, hoặc bảng InnoDB	chỉ có bảng HEAP	Không	Không	Không	Không	Không
Oracle	chỉ có ở phiên bản EE	Bảng gộp	Có	Không	Có	Có	Không
PostgreSQL	Có	Có	Có	Có	Có [[#fn_1010]]	Có	Có

e. Các đối tượng khác

Bảng 2-5 : So sánh về các đối tượng khác

	Domain	Cursor	Trigger	Hàm [[#fn_1111]]	Thủ tục [[#fn_1111]]	External routine [[#fn_1111]]
MySQL	Không	Có	Có	Có	Có	Có
Oracle	Có	Có	Có	Có	Có	Có
PostgreSQL	Có	Có	Có	Có	Có	Có

Nếu theo dõi các bảng từ 1.1.2.a->1.1.2.e so sánh giữa 3 hệ quản trị cơ sở dữ liệu MySQL, Oracle và PostgreSQL thì thấy rằng, hệ quản trị cơ sở dữ liệu PostgreSQL rất mạnh, nó hoạt động được trên hầu hết các hệ điều hành, hỗ trợ rất nhiều tính năng cơ bản, và hỗ trợ rất nhiều kiểu đánh chỉ mục.

Ngày nay, PostgreSQL là một trong những hệ quản trị cơ sở lớn nhất hiện có. Với những tính năng được chỉ ra dưới đây, chúng ta có cái nhìn tổng quan về PostgreSQL :

- Trong quan hệ đối tượng PostgreSQL, mọi bảng định nghĩa như một lớp. PostgreSQL thực thi kế thừa giữa các bảng, hàm và toán tử là đa hình.
- Cú pháp chuẩn của PostgreSQL tuân thủ theo chuẩn của SQL92 và nhiều tính năng của SQL99.

- PostgreSQL cung cấp nhiều kiểu dữ liệu. Bên cạnh kiểu dữ liệu *numeric*, *string* thông thường, nó còn cung cấp kiểu dữ liệu *geometry*, *boolean* và kiểu dữ liệu được thiết kế đặc biệt để dùng cho các địa chỉ mạng.
- Khả năng mở rộng là một trong những tính năng của PostgreSQL đó là nó có thể được mở rộng. Nếu với những gì mà PostgreSQL cung cấp mà bạn vẫn chưa hài lòng, bạn có thể thêm vào PostgreSQL những gì của bạn. Ví dụ, bạn có thể thêm vào kiểu dữ liệu mới, hàm và toán tử mới và các thủ tục mới.

2.1.3. Quản trị cơ sở dữ liệu qua giao diện

- **psql**

Kiểu giao diện chính thao tác cơ sở dữ liệu của PostgreSQL là chương trình dòng lệnh *psql*. Nhờ chương trình dòng lệnh này, người dùng có thể thực hiện truy vấn SQL một cách trực tiếp, và thực thi chúng từ tập tin. Hơn nữa, *psql* còn cung cấp một số lượng lớn các tùy chọn lệnh, tạo điều kiện tốt để viết các câu lệnh truy vấn và tự động hóa nhiều nhiệm vụ.

Cấu trúc lệnh : *psql [option...][dbname [username]]*

Bảng 2-6 : Danh sách các tùy chọn của lệnh psql

Tùy chọn	Giải thích
-c COMMAND	Thực thi 1 dòng lệnh đơn và sau đó thoát
-d NAME	Chỉ ra CSDL. Mặc định là tài khoản hiện tại của bạn
-f NAME	Thực thi lệnh nằm trong tập tin xác định là FILENAME, và sau đó thoát
-h HOSTNAME	Chỉ ra HOSTNAME
--help	Liệt kê thực đơn trợ giúp và sau đó thoát
-l	Liệt kê tất cả CSDL đang sẵn sàng và thoát

-p PORT	Chỉ ra cổng kết nối CSDL. Mặc định là 5432
-U NAME	Chỉ ra username đang kết nối với CSDL. Mặc định là user hiện tại

a. Kết nối đến CSDL

Thao tác kết nối đến CSDL là thao tác đầu tiên cần phải làm trước khi thực hiện các thao tác khác với *psql*. Để kết nối đến CSDL, yêu cầu biết tên CSDL, địa chỉ host và cổng của máy chủ và tên người dùng mà bạn muốn kết nối. *psql* cung cấp các tham số cho việc thao tác kết nối : -d (tên CSDL), -h (địa chỉ host), -p (địa chỉ cổng), -U (tên người dùng). Nếu không thể cung cấp đầy đủ các thông tin trên, thì yêu cầu tối thiểu nhất là bạn cần phải cung cấp thông tin về CSDL và tên người dùng. Đó là yêu cầu tối thiểu để kết nối đến CSDL.

Ví dụ đơn giản, kết nối đến CSDL có tên là *testdb*, tên người dùng là *postgres*

```
%>psql -d testdb -U postgres
```

Welcome to psql 8.1.20, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms

\h for help with SQL commands

\? for help with psql commands

\g or terminate with semicolon to execute query

\q to quit

```
testdb=>
```

theo dõi các tùy chọn ở trên, \h liệt kê tất cả các câu lệnh SQL mà *psql* có hỗ trợ, tùy chọn \? Liệt kê tất cả các lệnh *psql*, \q để ngắt kết nối.

b. Lệnh trong psql

Như đã nói ở trên, để liệt kê tất cả các câu lệnh thao tác trong *psql*, chúng ta sử dụng tùy chọn “\?”. Với tùy chọn “\?”, kết quả là một danh sách của hơn 50 lệnh và được chia thành 6 nhóm. Bên dưới là danh sách các lệnh và các nhóm tương ứng :

Bảng 2-7: Nhóm lệnh chung của psql

Tên lệnh	Chức năng sử dụng
\c [DBNAME]	Kết nối đến cơ sở dữ liệu
\cd [DIR]	Thay đổi thư mục làm việc hiện tại
\q	Thoát khỏi psql
\h	Trợ giúp cú pháp lệnh SQL, chọn * nếu muốn xem tất cả

Bảng 2-8: Nhóm lệnh truy vấn bộ đệm của psql

Tên lệnh	Chức năng sử dụng
\e [FILE]	Chỉnh sửa truy vấn bộ đệm hoặc file với bộ soạn thảo
\p	Đưa ra nội dung của truy vấn bộ đệm (đã thực hiện ngay trước đó)
\g [FILE]	Gửi truy vấn bộ đệm đến máy chủ và kết quả ra file
\r	Reset lại truy vấn
\s [FILE]	Hiển thị lịch sử lệnh hoặc lưu nó lại vào một file
\w [FILE]	Viết truy vấn bộ đệm vào file câu lệnh đã thực hiện ngay trước đó.

Bảng 2-9: Nhóm lệnh vào / ra của psql

Tên lệnh	Chức năng sử dụng
\echo [STRING]	Viết ra chuỗi, kết quả ở màn hình
\i [FILE]	Thực thi lệnh từ file
\o [FILE]	Gửi tất cả các kết quả truy vấn vào file hoặc đường ống

Bảng 2-10: Nhóm lệnh thông tin của psql

Tên lệnh	Chức năng sử dụng
\d [NAME]	Đưa ra thông tin về bảng, chỉ mục hoặc khung nhìn
\d {t i s v S}	Liệt kê ra bảng/chỉ mục/khung nhìn/trình tự
\da	Liệt kê các hàm tập hợp
\db	Liệt kê tất cả các tablespace
\dc	Liệt kê tất cả các conversion (quá trình chuyển đổi)
\df	Liệt kê danh sách các hàm
\l	Liệt kê danh sách tất cả các cơ sở dữ liệu

Bảng 2-11: Nhóm lệnh định dạng của psql

Tên lệnh	Chức năng sử dụng
----------	-------------------

<code>\a</code>	Căn lên
<code>\c [STRING]</code>	đưa ra tiêu đề về chuỗi đã nhập
<code>\f [STRING]</code>	Đưa ra dấu ngăn cách về chuỗi đã nhập

Tuy nhiên, trong khuôn khổ tài liệu này, chúng tôi chỉ đưa ra các lệnh được coi là thường xuyên sử dụng trong quá trình thao tác với psql.

c. Kết nối đến CSDL mới

Trong suốt quá trình thao tác, có thể bạn cần phải làm việc với nhiều hơn một CSDL. Do vậy, để thay đổi CSDL đến CSDL mới, chúng ta có thể thao tác với tùy chọn “\connect” hoặc “\c” theo cú pháp sau : “\connect *[tên cơ sở dữ liệu mới]*”

VD : testdb=> \connect postgresdb

d. Thực thi dòng lệnh được định vị trong một file xác định

Việc thao tác với những dòng lệnh nhiều lần có thể gây nhầm chán cho người dùng, đôi khi còn gây ra lỗi không mong muốn. Có một cách rất đơn giản, chúng ta có thể lưu những dòng lệnh thường xuyên được sử dụng vào một file riêng biệt, sau đó, khi muốn thực hiện, chúng ta chỉ cần gọi file đó bằng tùy chọn “\i” theo cú pháp sau :

“\i *[tên file .sql]*”

VD : testdb=> \i audit.sql

e. Chỉnh sửa file

Các dạng file đã nói ở trên có nhiệm vụ lưu những dòng lệnh thường xuyên được sử dụng không phải lúc nào cũng chính xác với những gì bạn mong muốn. Để thực hiện sửa đổi file đó ngay tại giao diện của psql, chúng ta chỉ cần sử dụng tùy chọn “\e” để chỉnh sửa file theo cú pháp sau :

“\e *[tên file .sql]*”

VD : testdb=> \e audit.sql

f. Lưu kết quả truy vấn vào file

Nếu bạn muốn lưu kết quả sau truy vấn vào một file để thuận lợi cho mục đích sử dụng của bạn, bạn có thể sử dụng tùy chọn “\o” theo cú pháp sau :

“\o [tên file .sql]”

VD : testdb=> \e output.sql

g. Các câu lệnh SQL được psql hỗ trợ

Tùy chọn “\h” cho chúng ta một bảng danh sách các câu lệnh SQL được psql hỗ trợ.

Bảng 2-12 : Danh sách lệnh \h

ABORT	CREATE LANGUAGE	DROPPVIEW
ALTER AGGREGATE	CREATE OPERATOR CLASS	END
ALTER CONVERSION	CREATE OPERATOR	EXECUTE
ALTER DATABASE	CREATE ROLE	EXPLAIN
ALTER DOMAIN	CREATE RULE	FETCH
ALTER FUNCTION	CREATE SCHEMA	GRANT
ALTER GROUP	CREATE SEQUENCE	INSERT
ALTER LANGUAGE	CREATE TABLE	LISTEN
ALTER INDEX	CREATE TABLE AS	LOAD
ALTER OPERATOR CLASS	CREATE TABLESPACE	LOCK
ALTER OPERATOR	CREATE TRIGGER	MOVE
ALTER ROLE	CREATE TYPE	NOTIFY

ALTER SCHEMA	CREATE USER	PREPARE
ALTER TABLE	DEALLOCATE	REINDEX
ALTER TABLESPACE	DECLARE	RELEASE SAVEPOINT
ALTER TRIGGER	DELETE	RESET
ALTER TYPE	DROP AGGREGATE	REVOKE
ALTER USER	DROP CAST	ROLLBACK
ANALYZE	DROP CONVERSION	ROLLBACK PREPARED
BEGIN	DROP DATABASE	ROLLBACK TO SAVEPOINT
CHECKPOINT	DROP DOMAIN	SAVEPOINT
CLOSE	DROP FUNCTION	SELECT
CLUSE	DROP GROUP	SELECT INTO
COMMENT	DROP INDEX	SET
COMMIT	DROP LANGUAGE	SET CONSTRAINTS
COMMIT PREPARED	DROP OPERATOR CLASS	SET ROLE
COPY	DROP OPERATOR	SET SESSION AUTHORIZATION
CREAT AGGREGATE	DROP ROLE	SET TRANSACTION
CREATE CAST	DROP RULE	SHOW
CREATE CONSTRAINT	DROP SCHEMA	START TRANSACTION

TRIGGER		
CREATE CONVERSION	DROP SEQUENCE	TRUNCATE
CREATE DATABASE	DROP TABLE	UNLISTEN
CREATE DOMAIN	DROP TABLESPACE	UPDATE
CREATE FUNCTION	DROP TRIGGER	VACUUM
CREATE GROUP	DROP TYPE	
CREATE INDEX	DROP USER	

Quan sát bảng 2-12 ta thấy rằng, hệ thống hỗ trợ rất nhiều lệnh, tuy nhiên, trong khuôn khổ khóa luận này, tôi chỉ sử dụng các lệnh thường dùng như SELECT, INSERT INTO, CREATE TABLE, DROP TABLE... Để tìm hiểu kỹ hơn về một lệnh cụ thể, chúng ta thực thi lệnh theo cú pháp “\h [lệnh]”. Ví dụ, để hiểu hơn về lệnh INSERT, thực thi lệnh :

corporate=> \h INSERT

Command: INSERT

Description: create new rows in a table

Syntax:

```
INSERT INTO table [ ( column [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) | query }
```

Kết quả thu được bao gồm các thông tin : tên lệnh, giải thích, và cú pháp lệnh.

Như đã biết, psql là công cụ quản lý và thao tác trên CSDL ở chế độ dòng lệnh, bởi vậy, tất cả các thao tác truy vấn với cơ sở dữ liệu như tạo, xóa, sửa bảng, chèn, xóa, sửa dữ liệu trong bảng dữ liệu đều được psql hỗ trợ :

Các ví dụ :

Câu lệnh SELECT : country=# SELECT * FROM t1;

Câu lệnh DELETE : DELETE FROM t1 WHERE num = 2;

Tóm lại, psql là công cụ quản lý và thao tác với cơ sở dữ liệu thông qua giao diện dòng lệnh. Nó hỗ trợ nhiều tính năng, từ việc tạo, sao lưu các câu lệnh truy vấn đến việc chỉnh sửa file dữ liệu, hỗ trợ việc thao tác với cơ sở dữ liệu bằng việc cung cấp nhiều lệnh SQL.

- **pgAdmin**

pgAdmin miễn phí và là công cụ quản lý giao diện đồ họa mã nguồn mở cho PostgreSQL, là cơ sở dữ liệu mã nguồn mở cao cấp nhất trên thế giới. Nó có thể dùng trên Linux, FreeBSD, Solaris, Mac OSX và Window.

pgAdmin được thiết kế để đáp ứng nhu cầu của tất cả người dùng, từ việc viết truy vấn đơn giản để phát triển cơ sở dữ liệu phức tạp. Giao diện đồ họa hỗ trợ tất cả các tính năng của PostgreSQL và làm cho việc quản trị dễ dàng. Ứng dụng này cũng bao gồm bộ soạn thảo cú pháp SQL, bộ soạn thảo mã server-side.

2.2. PostGIS

2.2.1. Giới thiệu về PostGIS

- **PostGIS là gì?**

PostGIS được Refraction Research Inc phát triển, như một dự án nghiên cứu công nghệ CSDL không gian. **PostGIS** hỗ trợ đối tượng địa lý cho CSDL đối tượng quan hệ PostgreSQL. PostGIS “kích hoạt khả năng không gian” cho PostgreSQL, cho phép PostgreSQL sử dụng như một CSDL không gian phụ trợ cho các hệ thống thông tin địa lý (GIS).

- **Đặc điểm của PostGIS :**

Do PostGIS được sử dụng như một CSDL không gian, nên nó bao gồm tất cả các đặc điểm của CSDL không gian được nêu ra ở mục 1.1.2. Ngoài ra, nó còn có những đặc trưng như :

- + Các kiểu dữ liệu hình học như Point, Linestring, Polygon, Multipoint, multilinestring, Multipolygons và Geometrycollection. Các kiểu dữ liệu hình học này được lưu trữ như những đối tượng hình học.

- + Các toán tử không gian cho phép xác định các phép đo không gian địa lý như tính diện tích, tính khoảng cách, tính độ dài, và tính chu vi. PostGIS hỗ trợ các hàm

nếu : ST_Area(), ST_Length(), ST_Perimeter(), ST_Distance()...các hàm này thường thực hiện chức năng kiểu phép đo.

+ Các toán tử không gian cho phép xác định không gian địa lý. Các thao tác như phép hợp, so sánh sự khác nhau giữa các đối tượng hình học. Các toán tử được PostGIS hỗ trợ để làm việc này có thể là : ST_Difference() : trả về phần khác nhau giữa 2 đối tượng hình học hay hàm ST_Buffer()...

+ PostGIS cung cấp việc đánh chỉ mục không gian tốc độ cao sử dụng GiST hoặc R-tree. Công cụ đánh chỉ mục không gian mà PostGIS hỗ trợ làm tăng tốc cho truy vấn không gian đặc biệt là trên bảng dữ liệu lớn.

+ Chỉ mục hỗ trợ chọn lọc, cung cấp việc thực hiện truy vấn bản đồ pha trộn truy vấn không gian hoặc truy vấn không có không gian

2.2.2. Công cụ shp2pgsql

shp2pgsql là công cụ dùng để chuyển định dạng file từ dạng shape file sang định dạng file .sql. Lưu ý, shape file là định dạng dữ liệu không gian địa lý vector phổ biến cho các phần mềm GIS. Shape file trong không gian mô tả các kiểu hình học chính là Line, Point và Polygon. Các kiểu Point, Line, Polygon cùng với các thuộc tính địa lý có thể tạo rất nhiều hiện thị với dữ liệu địa lý. Shape file còn dùng để lưu trữ vị trí hình học và thông tin thuộc tính liên quan.

Bảng 2-13: Các tùy chọn *shp2pgsql*

Tùy chọn	Giải thích
-D	Sử dụng để định dạng kết xuất CSDL. Mặc định, định dạng chèn được sử dụng, nó yêu cầu CSDL phân tích mỗi dòng chèn. Định dạng kết xuất tải nhanh hơn so với mặc định
-s	Sử dụng số SRID (hệ thống định danh không gian tham chiếu) khi tạo bảng và hình học. Điều này quan trọng để xác định, như biết được số SRID được yêu cầu để hỗ trợ phối hợp bên trong CSDL
-i	Sử dụng 32bit số nguyên cho tất cả các giá trị số nguyên

2.2.3. Công cụ psql

Đối với những người thích giao diện dòng lệnh thay thế giao diện đồ họa, *psql* cung cấp một cách thức mạnh mẽ để quản lý mọi mặt của máy chủ PostgreSQL. Với *psql*, bạn có thể tạo và xóa CSDL, tablespaces, bảng, thực thi transaction, thực thi các truy vấn thông thường như chọn bảng, chèn và nhiều hơn thế nữa.

Cấu trúc lệnh của *psql* : Câu lệnh thực thi *psql* thường có dạng :

Psql [option...][dbname[username]]

Yêu cầu ít nhất, bạn phải nhập tham số *dbname* và *username*. Các tùy chọn của lệnh *psql*, chúng ta có thể xem ở bảng 2-6.

Ngoài tác dụng thực thi các truy vấn, công cụ *psql* rất có hữu ích trong PostGIS, nó chính là công cụ dùng để thực thi nội dung file có định dạng là .sql sau khi nó được chuyển từ định dạng shape file.

2.2.4. Phương pháp load dữ liệu định dạng file .sql

SQL (file .sql): dữ liệu có thể được load vào PostGIS bằng cách load các tập tin lệnh SQL vào màn hình tương tác SQL. Có hai cách để load file dữ liệu định dạng file .sql. chúng ta có thể dùng pgAdmin III, đây là công cụ quản lý có giao diện nên việc thao tác trở nên dễ dàng. Ngoài ra, việc dùng dòng lệnh để load file dữ liệu định dạng file.sql cũng được dùng phổ biến trong Linux.

Đối với pgAdmin III chúng ta thực hiện các bước sau:

B1: Mở pgAdmin III

B2 : Kết nối cơ sở dữ liệu trong PostgreSQL Database Server 8.4 (localhost: 5432)

B3 : Chọn một CSDL, sau đó chọn SQL query.

B4 : Trong cửa sổ SQL Query , chọn File-> Open

B5 : Tìm đến file có định dạng “.sql”->nhấn OK để attack nội dung file.

B6 : Thực thi câu lệnh SQL bằng cách chọn Query->Execute query. Hoặc nhấn nút Execute query trên toolbar.

Cách thứ hai dùng để load file dữ liệu dạng “.sql” là dùng giao diện dòng lệnh psql. Và có lẽ, việc thao tác bằng giao diện dòng lệnh psql đơn giản và nhanh hơn đối với cách thao tác đồ họa pgAdmin III. Chúng ta chỉ cần sử dụng 1 lệnh :

```
huongnghiem@koinoi: psql -U [tên_người_dùng] -f [tên_file.sql] -d [tên_CSDL]
```

2.2.5. Phương pháp load dữ liệu dạng shape file vào CSDL

- **Shapefile là gì?**

Định dạng shapefile là định dạng dữ liệu không gian địa lý vector phổ biến cho các phần mềm GIS.

Đặc điểm của shapefile:

- + Một shapefile được tổ chức thành các tập tin riêng rẽ, tối thiểu cần có 3 tập tin với phần mở rộng là “.shp”, “.shx”, “.dbf”. và định dạng tập tin shapefile có phần mở rộng là “.shp”.
- + Tập tin có phần mở rộng dạng “.shp” chứa các thông tin về đặc điểm, hình dạng hình học của đối tượng. Tập tin có phần mở rộng dạng “.shx” chứa các thông tin về thứ tự của các đối tượng. Và tập tin có phần mở rộng dạng “.dbf” chứa các thông tin về bảng dữ liệu thuộc tính của đối tượng.
- + Shapefile là một định dạng lưu trữ vector số để lưu trữ vị trí hình học và thông tin thuộc tính liên quan.
- + Một shapefile khi hiển thị trong phần mềm GIS được gọi là lớp dữ liệu. Mỗi lớp thể hiện cho một đặc tính hình học không gian của một lớp đối tượng cần biểu diễn gồm : POINT, LINE, POLYGON và các thuộc tính liên quan đến các đối tượng đó.
- + Shapefile là dạng đơn giản vì chúng lưu trữ các kiểu dữ liệu hình học ban đầu như: LINE, POINT, POLYGON. Các kiểu hình học ban đầu này được sử dụng giới hạn mà không có bất kỳ thuộc tính để xác định những gì chúng hiển thị. Vì vậy, một bảng trong bản ghi sẽ lưu tính chất / thuộc tính của mỗi hình dạng ban đầu trong shapefile. Các hình dạng (LINE, POINT, POLYGON) cùng với các thuộc tính dữ liệu có thể tạo ra rất nhiều hiển thị với dữ liệu địa lý.

- **Các bước tiến hành**

+ Trong Window

Sử dụng công cụ *psql* và công cụ *shp2pgsql* để load file dữ liệu dạng shape (.shp).

Giả sử ta cần chuyển đổi file *bc_pubs.shp* sang file dạng *bc_pubs.sql*.

B1 : Start -> chọn Accessories -> chọn Command Prompt

B2 : Chọn đường dẫn đến thư mục chứa các file định dạng .shp

Trong ví dụ này, ta chọn đường dẫn : *C:\Program Files\PostgreSQL\8.4\bin* bằng lệnh :

```
cd \Program Files\PostgreSQL\8.4\bin
```

B3 : Sử dụng công cụ *shp2pgsql* :

```
C:\Program Files\PostgreSQL\8.4\bin\shp2pgsql  
-d -i -s 4326 bc_pubs.shp bc_pubs > bc_pubs.sql
```

B4 : Thực thi nội dung tập tin mới tạo được (*bc_pubs.sql*) bằng công cụ *psql*

```
C:\Program Files\PostgreSQL\8.2\bin\psql.exe  
-U postgres -f bc_pubs -d postgis
```

‘*-U postgres*’: tên người dùng là *postgres*

‘*-f bc_pubs*’: tên file cần thực thi là *bc_pubs.sql*

‘*-d postgis*’: tên cơ sở dữ liệu *postgis*.

B5 : Refresh pgAdmin III, trong CSDL bạn đã chọn, sẽ xuất hiện bảng *bc_pubs*.

+ Trong Linux

```
huongnghiem@koinoi:~/data$ shp2pgsql -d -i -s 4326 bc_pubs.shp bc_pubs >  
bc_pubs.sql
```

Shapefile type: Arc

Postgis type: MULTIPOLYGON[2]

```
huongnghiem@koinoi:~/data$ psql -U huongnghiem -f bc_pubs.sql -d huongnghiem
```

2.2.6. OpenGIS Well-Know Text

Đối tượng GIS hỗ trợ bởi PostGIS là một tập lớn của Simple Features được định nghĩa bởi OpenGIS Consortium (OGC). Đặc tả OpenGIS định nghĩa cách thể hiện chuẩn của đối tượng không gian đó là dạng Well-Know Text (WKT). WKT bao gồm các thông tin về kiểu của đối tượng và các tọa độ dạng đối tượng.

VD về hiển thị dạng WKT của đối tượng không gian :

POINT(0 0)

LINESTRING(0 0,1 1,1 2)

POLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))

MULTIPOINT(0 0,1 2)

MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))

MULTIPOLYGON((((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)

GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))

2.2.7. Bảng siêu dữ liệu

Với OpenGIS *Simple Features Specification for SQL* định nghĩa kiểu đối tượng GIS chuẩn, hàm được yêu cầu vận dụng chúng, và một tập các bảng siêu dữ liệu. Sau đó đảm bảo dữ liệu vẫn phù hợp, các thao tác như tạo và xóa một cột không gian được thực hiện thông qua các thủ tục đặc biệt được định nghĩa bởi OpenGIS.

Khi một cơ sở dữ liệu không gian được kích hoạt với PostGIS, có 2 bảng siêu dữ liệu được tạo ra, được chỉ định bởi OGC đó là 2 bảng siêu dữ liệu có tên SPATIAL_REF_SYS và GEOMETRY_COLUMNS.

- **Bảng GEOMETRY_COLUMNS**

Điều khiển GEOMETRY_COLUMNS như một thư mục về mô tả những gì mà một bảng đã tồn tại được kích hoạt không gian trong cơ sở dữ liệu. Nó không lưu trữ cập nhật một cách tự động, do đó, câu lệnh CREATE TABLE đơn giản bao gồm một kiểu GEOMETRY, sẽ không thêm một mục vào bảng. Để làm điều đó, hàm AddGeometryColumn() có thể dùng để thêm đồng thời một cột không gian vào bảng phi

không gian trong khi cập nhật cột GEOMETRY_COLUMNS. Hàm này sẽ được mô tả chi tiết ở phần sau.

Cấu trúc của bảng GEOMETRY_COLUMNS là :

Table "public.geometry_columns"		
Column	Type	Modifiers
f_table_catalog	character varying(256)	not null
f_table_schema	character varying(256)	not null
f_table_name	character varying(256)	not null
f_geometry_column	character varying(256)	not null
coord_dimension	integer	not null
srid	integer	not null
type	character varying(30)	not null

Mô tả về bảng GEOMETRY_COLUMNS bằng cách mô tả về mỗi cột trong bảng. Mỗi cột không gian được xác định duy nhất bởi sự kết hợp của shema/table/column. Cột COORD_DIMENSION xác định chiều không gian (2, 3 hoặc 4 chiều) của cột. Cột SRID mô tả hệ thống tham chiếu không gian, nó là khóa ngoài tham chiếu đến bảng SPATIAL_REF_SYS. Cuối cùng là cột cột TYPE mô tả kiểu hình học được mô tả trong bảng, sử dụng một trong các kiểu sau : POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION...

• Bảng SPATIAL_REF_SYS

Điều khiển SPATIAL_REF_SYS như một thư mục của hệ thống tham chiếu không gian. Mỗi kiểu hình học trong cơ sở dữ liệu không gian có liên quan đến số SRID hay còn gọi là tham số nhận diện tham chiếu không gian, và nó là một số nguyên. Muốn biết được số SRID trong một bảng dữ liệu cụ thể, ta dùng hàm ST_SRID() (để biết thêm chi tiết về hàm này, xem thêm cuốn PostGIS Manual).

Cấu trúc của bảng SPATIAL_REF_SYS :

Table "public.spatial_ref_sys"		
Column	Type	Modifiers

srid	integer	not null
auth_name	character varying(256)	
auth_srid	integer	
srtext	character varying(2048)	
proj4text	character varying(2048)	

Cột SRID là định danh duy nhất (có thể hiểu như khóa chính của bảng dữ liệu). Cột AUTH_NAME mô tả cơ quan hoặc tổ chức định nghĩa và sử dụng hệ thống tham chiếu. Cột AUTH_SRID là số nguyên được gán bởi cơ quan hoặc tổ chức, còn cột SRTEXT hiển thị WKT của hệ thống tham chiếu không gian.

* **LƯU Ý :** Mặc định, bảng SPATIAL_REF_SYS đã luân chuyển với PostGIS được lấy từ hệ thống cơ sở dữ liệu tham chiếu không gian EPSG. Ví dụ như số SRID trong bảng SPATIAL_REF_TABLE và định danh EPSG luôn luôn giống nhau.

2.2.8. Bảng không gian

• Bảng không gian là gì?

Bảng không gian là một bảng bao gồm một hoặc nhiều cột không gian. Việc tạo ra một bảng không gian, ngoài những cột có kiểu dữ liệu thông thường, còn chỉ ra cột nào là cột không gian trong bảng đó. Cột không gian chỉ có thể chấp nhận kiểu dữ liệu được yêu cầu bởi cột không gian. Kiểu hình học được dùng trong cột không gian của bảng không gian là Point, Multipoint, Linestring, MultiLinestring, Polygon, Multipolygon.

Giá trị tham chiếu không gian, viết tắt là SRID, là giá trị rất quan trọng, nó xác định tính duy nhất của hệ thống không gian trong phạm vi CSDL. Nó được yêu cầu chỉ ra khi tạo đối tượng không gian cho việc chèn vào CSDL. Thông tin của giá trị SRID được lưu trữ trong bảng *SPATIAL_REF_SYS* được tạo mặc định khi cài đặt PostGIS. SRID của hệ thống tham chiếu không gian của hình học được lưu trữ với kiểu hình học của chính nó, vì thế, điều quan trọng là bạn phải chọn SRID một cách cẩn thận.

• Tạo bảng không gian trong PostGIS

a. Cách thông thường

Tạo bảng với câu lệnh CREATE TABLE, và một thuộc tính của bảng sẽ có kiểu dữ liệu dạng “geometry”. Ví dụ, tạo bảng *points* (*name varchar, point geometry*);

Chú ý : khi chèn dữ liệu vào bảng không gian cần chú ý đến trường có kiểu dữ liệu dạng “geometry”, dữ liệu sẽ gồm các đối tượng không gian như POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON và định dạng dữ liệu nhập phải đúng như định dạng WKT;

Ví dụ, để nhập dữ liệu cho đối tượng POINT có tọa độ (0,0), chúng ta dùng dạng POINT(0 0). Còn để nhập dữ liệu cho đối tượng LINESTRING nối tọa độ (0,0) và (3, 4), chúng ta dùng dạng LINESTRING (0 0,3 4);

```
INSERT INTO points VALUES ('a', 'POINT(0 0)');
```

b. Dùng hàm AddGeometryColumn()

Để tạo một bảng dữ liệu không gian việc đầu tiên là tạo ra bảng dữ liệu, tuy nhiên, trong bảng dữ liệu này sẽ không chứa cột dữ liệu không gian. Sau đó, để có được bảng dữ liệu không gian, chúng ta cần thêm cột dữ liệu không gian bằng cách sử dụng hàm AddGeometryColumn(). Bảng dữ liệu không gian đã được tạo ra, công việc chèn dữ liệu vào bảng cũng tương tự như cách thông thường.

B1 : Tạo bảng thông thường (không phải bảng dữ liệu không gian)

```
CREATE TABLE ROADS_GEOM (ID int4, NAME varchar(25))
```

B2 : Thêm cột không gian vào bảng sử dụng hàm “AddGeometryColumn”

AddGeometryColumn

(<schema_name>,<table_name>,<column_name>,<srid>,<type>,<dimension>)

Giải thích các tham số của hàm :

Hàm AddGeometryColumn : thêm cột hình học vào bảng đã tồn tại.

+ <schema_name> : tên của bảng sơ đồ bảng.

+ <table_name> : tên của bảng cần thêm cột không gian

+ <column_name> : tên cột cần thêm theo kiểu không gian

+ <srid_name> : srid là một giá trị nguyên, xác định tính duy nhất của hệ thống tham chiếu không gian trong phạm vi của CSDL. Nghĩa là SRID của các bảng khác nhau trong CSDL không gian phải hoàn toàn khác nhau.

+ <type>: xác định kiểu hình học cho cột.

+ <dimension> : thuộc chiều nào (0, 1, 2 hoặc 3)

Nếu đang ở sơ đồ hiện tại thì bỏ qua thông số <schema_name>:

AddGeometryColumn (<table_name>,<column_name>,<srid>,<type>,<dimension>)

VD : yêu cầu tạo bảng tên *points* (*name varchar(20)*) là một bảng không gian, sau đó thêm trường *the_geom* bằng hàm AddGeometryColumn();

```
CREATE TABLE points(name varchar(20));
```

```
SELECT AddGeometryColumn('public','points','the_geom',-1,'POINT',2);
```

```
INSERT INTO points(name, the_geom) values('A','POINT(1 0)');
```

```
INSERT INTO points(name, the_geom) values('B','POINT(0 1)');
```

2.3. Hàm trong PostGIS

2.3.1. Nhóm hàm điều khiển

- AddGeometryColumn()

- Chức năng của hàm AddGeometryColumn là thêm một cột hình học vào bảng đã tồn tại. Hàm này rất quan trọng trong việc tạo bảng trong CSDL không gian.

- Cú pháp :

```
text AddGeometryColumn(varchar table_name, varchar column_name, integer  
srid, varchar type, integer dimension);
```

```
text AddGeometryColumn(varchar schema_name, varchar table_name, varchar  
column_name, integer srid, varchar type, integer dimension);
```

* chú ý : *schema_name* : là tên của bảng sơ đồ, thường mặc định khi cài đặt PostgreSQL

Srid : phải có giá trị là một số nguyên, tham chiếu đến bảng SPATIAL_REF_SYS.

Type : xác định kiểu hình học cho cột cần thêm, ví dụ : POLYGON, MULTILINESTRING, POINT, MULTIPOINT...

Dimention : xác định chiều hình học, chiều tương ứng với kiểu hình học.

- Ví dụ : Tạo bảng hình học có tên là my_spatial_table (id serial);

Thêm cột hình học (the_geom) có kiểu POINT, trong không gian 2D :

```
SELECT AddGeomtryColumn('my_schema', 'my_spatial_table', 'the_geom',  
4326, 'POINT', 2);
```

- **DropGeometryColumn()**

- Chức năng của hàm DropGeometryColumn là loại bỏ một cột hình học từ bảng không gian.

- Cú pháp :

```
text DropGeometryColumn(varchar table_name, varchar column_name);
```

```
text DropGeometryColumn(varchar schema_name, varchar table_name, varchar  
column_name);
```

- Ví dụ :

```
SELECT DropGeomtryColumn('my_schema','my_spatial_table','the_geom');
```

- **DropGeometryTable**

- Chức năng của hàm DropGeometryTable() là loại bỏ bảng và tất cả những gì tham chiếu trong cột hình học.

- Cú pháp :

```
boolean DropGeometryTable(varchar table_name);
```

```
boolean DropGeometryTable(varchar schema_name, varchar table_name);
```

- Ví dụ : SELECT DropGeometryTable('my_schema', 'my_spatial_table');

2.3.2. Nhóm hàm khởi tạo hình học

- **ST_GeometryFromText()**

- Chức năng của hàm ST_GeometryFromText là trả về giá trị được chỉ định ST_Geometry từ hiển thị WKT.

- Cú pháp : geometry ST_GeometryFromText(text WKT);

- Ví dụ : `SELECT ST_GeometryFromText('POINT(1 0)');`
`st_geomfromtext`

 0101000000000000000000F03F0000000000000000

* Chú ý : Hàm `ST_GeometryFromText()` cũng có thể được viết là `ST_GeomFromText()`

2.3.3. Hàm trả về kiểu hình học ở đầu ra.

- **ST_AsText()**

- Chức năng của hàm `ST_AsText` là trả về hiển thị dạng WKT của hình.
- Cú pháp : `text ST_AsText(geometry g);`
- Ví dụ : `SELECT ST_AsText(ST_Union(ST_GeomFromText('POINT(1 2)'), ST_GeomFromText('POINT(1 2)')));`

Giá trị trả về : `POINT (1 2);`

`SELECT ST_AsText('0101000000000000000000F03F0000000000000000 ');`

Giá trị trả về : `POINT(1 0);`

2.3.4. Hàm xác định mối quan hệ không gian

- **ST_Equals()**

- Chức năng của hàm `ST_Equals` là trả về `True` nếu đưa ra những hình coi là “bằng nhau trong không gian”. Lưu ý, “bằng nhau trong không gian ” nghĩa là `ST_Within(A, B)=True` và `ST_Within(B,A)=True` và cũng có nghĩa là sắp xếp của các điểm có thể khác nhau nhưng cấu trúc hiển thị hình học lại giống nhau.

- Cú pháp : `boolean ST_Equals(geometry A, geometry B);`
- Ví dụ : `SELECT ST_Equals(ST_GeoFromText('LINESTRING (0 0, 10 10)'), ST_GeoFromText('LINESTRING (0 0, 5 5, 10 10)');`

Giá trị trả về là `True` vì : `LINESTRING(0 0, 10 10)` và `LINESTRING (0 0, 5 5, 10 10)` đều trả về đoạn thẳng từ điểm (0, 0)-> điểm (10, 10)

- **ST_Disjoint()**

- Chức năng của hàm ST_Disjoint là trả về True nếu các hình “không giao nhau trong không gian” nếu chúng không chia sẻ bất cứ khoảng không gian nào cho nhau, hay là tách biệt hẳn với nhau. Nếu bất kỳ các hàm ST_Overlaps(), ST_Touches(), ST_Within() trả về True thì các hình đó không phải có không gian phân chia. Lưu ý, hàm ST_Disjoint() không sử dụng cơ chế đánh chỉ mục.

- Cú pháp : boolean ST_Disjoint (geometry A, geometry B);

- Ví dụ : SELECT ST_Disjoint('POINT (0 0) '::geometry, 'LINESTRING (2 0, 0 2) '::geometry);

Giá trị trả về là True vì : điểm (0, 0) và đoạn thẳng nối 2 điểm (2, 0) và điểm (0, 2) không có bất kỳ điểm nào chung.

• ST_Intersects()

- Chức năng của hàm ST_Intersects là trả về True nếu các hình gọi là “giao nhau trong không gian” và trả về False nếu chúng không có bất cứ điểm nào giao nhau. Nếu các hàm ST_Overlaps(), ST_Touches(), ST_Within() trả về true, thì những hình đó được coi là giao nhau.

- Cú pháp : boolean ST_Intersects(geometry A, geometry B);

- Ví dụ : SELECT ST_Intersects('POINT(0 0) '::geometry, 'LINESTRING (2 0, 0 2) '::geometry);

Giá trị trả về là False vì : ST_Disjoint('POINT(0 0) '::geometry, 'LINESTRING (2 0, 0 2) '::geometry); trả về giá trị True, hay nói cách khác là điểm (0, 0) và đoạn thẳng (2,0)

-> (0,2) không có bất kỳ điểm giao nhau nào.

• ST_Touches()

- Chức năng của hàm ST_Touches là trả về True nếu các hình có ít nhất 1 điểm chung, nhưng bên trong của chúng lại không giao nhau. Quan hệ ST_Touches() áp dụng cho Vùng/Vùng, Đường/Đường, Đường/Vùng, Điểm/Vùng, Điểm/Đường nhưng không áp dụng cho cặp Điểm/Điểm.

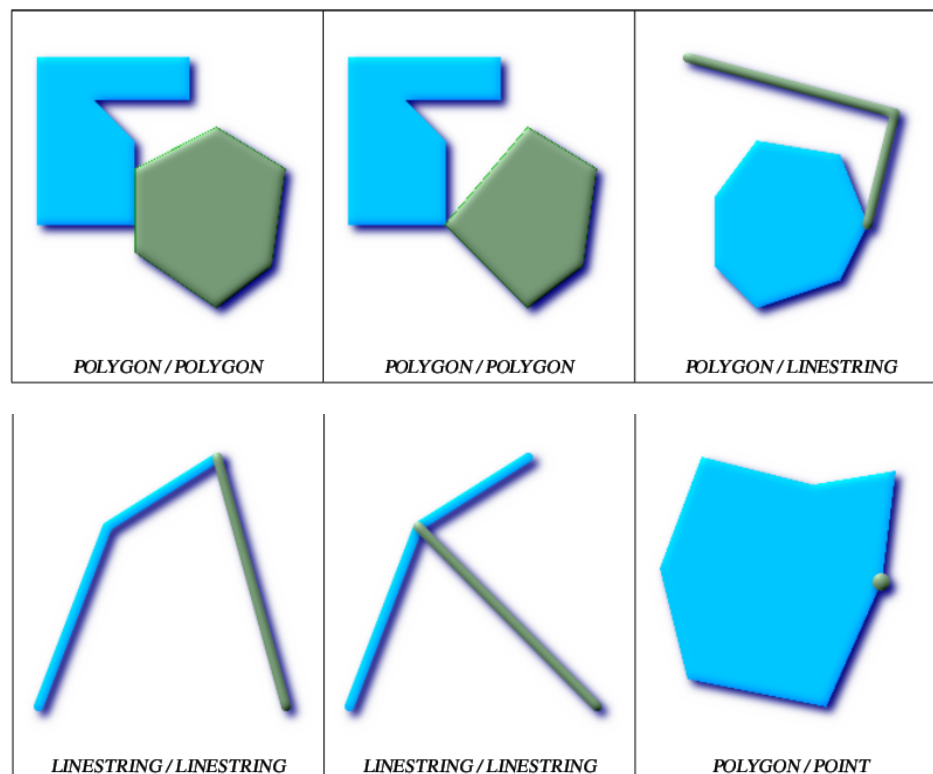
- Cú pháp : boolean ST_Touches(geometry g1, geometry g2);

- Ví dụ : `SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry);`

Trả về giá trị True vì đoạn thẳng từ điểm (0,0)->(1,1)->(0,2) tiếp xúc với điểm (0,2) tại đầu đoạn thẳng chứ không phải điểm giữa của đoạn thẳng. Nếu xét đoạn thẳng trên với điểm

(1, 1) thì giá trị trả về là False vì chúng tiếp xúc nhau với điểm giữa của đoạn thẳng.

Các minh họa về quan hệ ST_Touches() trả về giá trị True.



Hình 2-1 : Minh họa hàm ST_Touches().

• ST_Overlaps()

- Chức năng của hàm ST_Overlaps là trả về True nếu các hình có khoảng không gian chia sẻ, có cùng chiều, nhưng chúng không hoàn toàn bị chứa bởi hình khác.

- Cú pháp : `boolean ST_Overlaps(geometry A, geometry B);`

- Ví dụ :

• ST_Crosses()

- Chức năng của hàm ST_Crosses là trả về True nếu đối tượng hình học thu được có chiều nhỏ hơn chiều lớn nhất của 2 đối tượng hình học ban đầu. Đối tượng thu được phải chứa các điểm bên trong của 2 đối tượng hình học ban đầu và đối tượng thu được phải không bằng một trong 2 đối tượng đầu vào. Trường hợp còn lại, trả về False.

- Cú pháp : boolean ST_Crosses(geometry g1, geometry g2);

- Ví dụ : có 2 bảng roads (id , the_geom) và highways(id, the_geom)

Xác định danh sách road giao với highway :

```
SELECT roads.id FROM roads, highways WHERE ST_Crosses(roads.the_geom, highways.the_geom);
```

• ST_Within()

- Chức năng của hàm ST_Within là trả về True nếu hình A nằm hoàn toàn bên trong hình B

* Lưu ý : ST_Within(A, B)=ST_Contains(B, A)

- Cú pháp : boolean ST_Within(geometry A, geometry B)

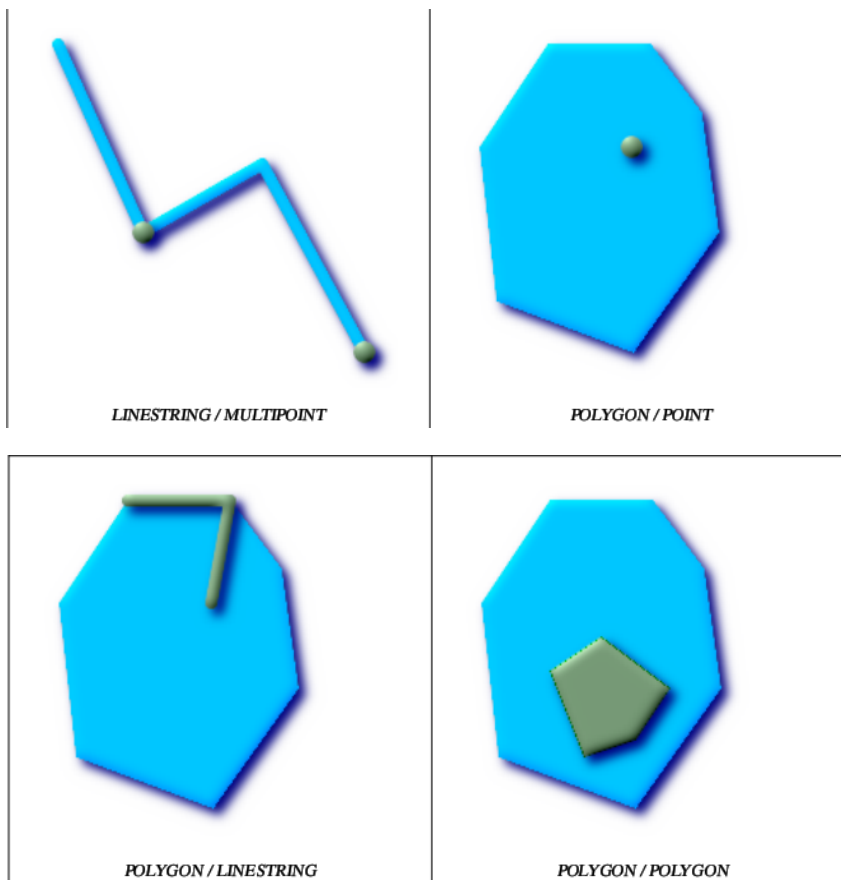
- Ví dụ : đường tròn nhỏ nằm hoàn toàn bên trong đường tròn to



Hình 2-2 : Minh họa hàm ST_Within()

• ST_Contains()

- Chức năng của hàm ST_Contains là trả về True khi và chỉ khi không có điểm nào của B nằm bên ngoài A, và ít nhất 1 điểm bên trong B nằm bên trong A.



Hình 2-3 : Minh họa hàm ST_Contains()

- Cú pháp : boolean ST_Contains(geometry B, geometry A);

• **ST_Distance()**

- Chức năng : hàm ST_Distance trả về khoảng cách giữa 2 điểm, giữa điểm và đường trong không gian 2D. Đơn vị mặc định là “meter”.

- Cú pháp : float ST_Distance (geometry g1, geometry g2);

- Ví dụ : Khoảng cách của 2 điểm POINT (0 0) và POINT (3 4);

```
SELECT ST_Distance ('POINT(0 0)', 'POINT(3 4)');
```

```
st_distance= 5;
```

Khoảng cách từ điểm POINT(0 0) đến đường LINESTRING(0 3, 3 4);

```
st_distance = 3;
```

- **ST_Length()**

- Chức năng : hàm ST_Area trả về diện tích của hình nếu nó là POLYGON hoặc MULTIPOLYGON. Đơn vị mặc định là m².

- Cú pháp : float ST_Area(geometry g1);

- **Ví dụ :** Bảng dữ liệu bc_voting_area lưu trữ thông tin của các vùng tham gia bầu cử. Yêu cầu tính tổng diện tích của tất cả các vùng có tham gia bầu cử có số người tham gia bầu cử >100?

```
SELECT Sum(ST_Area(the_geom))/10000 AS hectares
FROM bc_voting_areas
WHERE vttotal > 100;
```

```
hectares
-----
36609425.2114911
(1 row)
```

- **ST_Area()**

- Chức năng : hàm ST_Length() trả về độ dài 2d của hình nếu chúng là LINESTRING hoặc MULTILINESTRING. Đơn vị mặc định của độ dài là “meter”

- Cú pháp : float ST_Length(geometry Linestring);

- Ví dụ : Tính độ dài của Linestring sau :

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238
2967450,743265 2967450,743265.625 2967416,743238 2967416)',2249));

st_length
-----
122.630744000095
```

- **ST_Perimeter()**

- Chức năng : hàm ST_Perimeter trả về chu vi của hình nếu nó có dạng Polygon hoặc Multipolygon. Đơn vị mặc định là meter.

- Cú pháp : float ST_Perimeter(geometry g1);

- Ví dụ : SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450, 743265.625 2967416,743238 2967416))', 2249));

Giá trị trả về : st_perimeter : 122.630744000095

2.3.5. Nhóm hàm đưa ra đối tượng hình mới.

• ST_Intersection()

- Chức năng của hàm ST_Intersection là trả về một hình, hiển thị phần chung giữa hình A và hình B. Nếu hình A và hình B không có bất kỳ điểm chung thì trả về đối tượng hình rỗng.

- Cú pháp : geometry ST_Intersection(geometry A, geometry B);

- Ví dụ : SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry', 'LINESTRING(2 0, 0 2)::geometry));

Giá trị trả về EMTRY

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry', 'LINESTRING (0 0,0 2)::geometry));

Giá trị trả về : POINT(0 0).

• ST_Difference()

- Chức năng của hàm ST_Difference là trả về một hình hiển thị phần của hình A mà không giao với hình B. Chúng ta có thể hiểu theo công thức sau :

$ST_Difference() = GeometryA - ST_Intersection(A, B).$

Nếu A hoàn toàn nằm trong B, thì A và B không có điểm khác biệt, nghĩa là, hàm ST_Difference() trả về giá trị rỗng.

- Chức năng của hàm ST_SymDifference trả về một hình hiển thị phần của hình A và hình B không giao nhau. Nó được gọi là sự khác nhau đối xứng lý do : $ST_SymDifference(A, B) = ST_SymDifference(B, A)$. Chúng ta có thể hiểu theo công thức sau :

$$ST_SymDifference(A, B) = ST_Union(A, B) - ST_Intersection(A, B).$$



Hình 2-5 : Minh họa hàm ST_SymDifference().

Hình 2-7 biểu diễn kết quả của hàm ST_SymDifference(), nó trả về 1 phần của đường A và 1 phần của đường B mà không giao nhau.

- Cú pháp : `geometry ST_SymDifference (geometry geomA, geometry geomB);`

- Ví dụ : `SELECT ST_AsText(ST_SymDifference (`

`ST_GeomFromText('LINESTRING (50 100, 50 200)'),`

`ST_GeomFromText('LINESTRING (50 50, 50 150)')));`

Giá trị trả về : `MULTILINESTRING ((50 150, 50 200),(50 50, 50 100));`

2.3.6. Nhóm hàm thay đổi hình học

- **ST_Buffer() :**

- Chức năng của hàm ST_Buffer trả về một hình hiển thị cho tất cả các điểm mà khoảng cách của chúng từ hình \leq khoảng cách.

- Tùy chọn `buffer_style` :

+ `quad_segs`: số đoạn được sử dụng để xấp xỉ $\frac{1}{4}$ vòng tròn (mặc định là 8).

+ `endcap`= *round*|*flat*|*square*: kiểu kết thúc, mặc định là *round*.

+ `joint`=*round*|*mitre*|*bevel*: kiểu nối, mặc định là *round*

+ *mitre_limit* : tỉ lệ giới hạn mép

Đơn vị của bán kính được đo bằng đơn vị của hệ thống tham chiếu không gian. Đầu ra của hàm có thể là POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON và GEOMETRYCOLLECTION.

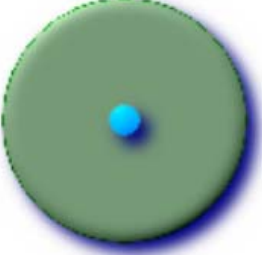
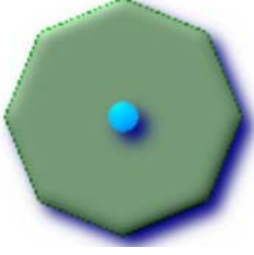


- Cú pháp : geometry ST_Buffer (geometry g1, float R);

Geometry ST_Buffer(geometry g1, float R, integer num_seg_quater_circle);

Geometry ST_Buffer(geometry g1, float R, text buffer_style_parameters);

- Ví dụ :

Bảng 2-14: Các ví dụ minh họa cho hàm ST_Buffer()

 Quad_segs=8 SELECT ST_Buffer(ST_GeomFromText('POINT(100, 90)'), 50, 'quad_segs=8');	 Quad_segs=2 SELECT ST_Buffer (ST_GeomFromText('POINT(100,90)'), 50, 'quad_seds=2');
	

Endcap=round và join=round SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50, 150 150, 150 50)'), 10, 'endcap=round join=round');	Endcap=bevel và join=round SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50, 150 150, 150 50)'), 10, 'endcap=square join=round');
--	---

2.3.7. Nhóm hàm accessor

• ST_GeometryType()

- Chức năng của hàm ST_GeometryType là trả về kiểu hình học dưới dạng chuỗi. Ví dụ : ST_Linestring, ST_Polygon, ST_MultiPolygon...
- Cú pháp : text ST_GeometryType(geometry g1);
- Ví dụ : SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(10 10, 20 20)'));

Giá trị trả về là ST_Linestring.

• GeometryType()

- Chức năng của hàm GeometryType là trả về kiểu của hình dưới dạng chuỗi như : LINESTRING, POLYGON, MULTIPOINT...
- Cú pháp : text GeometryType(geometry geomA);
- Ví dụ : SELECT GeometryType(ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2)'));

Giá trị trả về : LINESTRING.

• ST_IsValid()

- Chức năng của hàm ST_IsValid là trả về True nếu hình đó là hợp lệ. Khái niệm hợp lệ trong trường hợp này, nghĩa là, các kiểu hình học có dạng POINT, POLYGON...và được biểu diễn hợp lý như POINT(0 0), POLYGON(0 0, 1 1, 1 2, 0 0). Trong trường hợp kiểu hình học là không hợp lệ, thì PostgreSQL sẽ đưa ra thông báo chi tiết tại sao kiểu hình học đó lại không hợp lệ.
- Cú pháp : boolean ST_IsValid(geometry g)

- Ví dụ : `SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)'));`

Giá trị trả về là True.

2.4. Chỉ mục

Chỉ mục giúp việc sử dụng một cơ sở dữ liệu không gian với dữ liệu lớn nhất có thể được trở lên dễ dàng. Nếu không có việc đánh chỉ mục, thì bất kỳ việc tìm kiếm nào cũng phải yêu cầu việc “quét tuần tự” tất cả các bản ghi có trong CSDL. Như vậy, gây cản trở cho việc tìm kiếm như yêu cầu thời gian quét quá lớn, bộ nhớ máy tính cần phải lớn. Vì thế, việc đánh chỉ mục có vai trò rất quan trọng trong việc truy vấn một CSDL, đặc biệt là CSDL loại lớn.

Việc đánh chỉ mục nhằm tăng tốc độ tìm kiếm bằng cách tổ chức dữ liệu thành cây tìm kiếm, nó có duyệt một cách nhanh chóng để tìm một bản ghi cụ thể. PostgreSQL hỗ trợ 3 cách đánh chỉ mục đó là : B-Tree, R-Tree, và chỉ mục GiST.

2.4.1. Chỉ mục GiST

Chỉ mục GiST là kiểu chỉ mục mà PostGIS dùng. Cơ chế đánh chỉ mục GiST được áp dụng cho cột dữ liệu kiểu không gian trong CSDL không gian. Nó cũng có tác dụng làm tăng tốc độ tìm kiếm trên tất cả các kiểu dữ liệu.

Cú pháp để xây dựng chỉ mục GiST trên một cột không gian :

```
CREATE INDEX [index_name] ON [table_name] USING GIST  
([geometry_field]);
```

Chỉ mục GiST có 2 ưu điểm hơn chỉ mục R-Tree trong PostgreSQL ở chỗ. Thứ nhất, chỉ mục GiST là “Null safe”, nghĩa là, chúng có thể đánh chỉ mục cho tất cả các cột, bao gồm cả những cột có chứa giá trị Null. Thứ hai, chỉ mục GiST hỗ trợ các khái niệm “lossiness”, nó quan trọng khi phân chia với đối tượng GIS lớn hơn kích thước của trang (8K). “lossiness” cho phép PostgreSQL chỉ lưu trữ những phần quan trọng của một đối tượng trong một chỉ mục (áp dụng cho đối tượng GIS). Các đối tượng GIS lớn hơn 8K sẽ gây ra thất bại trong quá trình xử lý đối với kiểu chỉ mục R-Tree.

2.4.2. Sử dụng chỉ mục

Như đã biết, tác dụng của việc đánh chỉ mục là để tăng tốc độ tìm kiếm dữ liệu, và nó đặc biệt có tác dụng đối với lượng dữ liệu lớn.

Các ví dụ sau giúp ta theo dõi được hiệu quả tìm kiếm trước và sau khi đánh chỉ mục cho dữ liệu.

Đối với bảng dữ liệu đơn giản, lượng dữ liệu nhỏ bảng *points(name, the_geom)*.
Với câu lệnh `SELECT name, ST_AsText(the_geom) from points;`

Câu lệnh đánh chỉ mục cho cột *the_geom* :

```
CREATE INDEX the_geom_gist ON points USING GIST (the_geom);
```

Thời gian truy vấn trước đánh chỉ mục:

```
Total query runtime: 31 ms.
```

```
2 rows retrieved.
```

Thời gian truy vấn sau khi đánh chỉ mục cho cột *the_geom*

```
Total query runtime: 0 ms.
```

```
2 rows retrieved.
```

Có thể thấy, tác dụng của đánh chỉ mục, thời gian truy vấn giảm gấp nhiều lần.

Đối với bảng dữ liệu lớn bảng *bc_border(gid, border_id, the_geom)*, dữ liệu của bảng rất lớn, khoảng hơn 5000 hàng. Nếu không có cơ chế đánh chỉ mục, để tìm kiếm dữ liệu trong bảng, hệ thống phải “quét tuần tự” từ đầu cho đến khi nào tìm thấy dữ liệu theo yêu cầu. Cho nên, thời gian dành cho truy vấn sẽ rất lớn.

Nếu thực hiện câu lệnh truy vấn : `SELECT gid, border_id, ST_AsText(the_geom) FROM bc_border;`

Đánh chỉ mục cho cột *the_geom* : `CREATE INDEX border_the_geom_gist ON bc_border USING GIST (the_geom);`

Thời gian truy vấn trước khi đánh chỉ mục :

```
Total query runtime: 2125 ms.
```

```
5199 rows retrieved.
```

Thời gian truy vấn sau khi đánh chỉ mục :

```
Total query runtime: 1890 ms.
```

```
5199 rows retrieved.
```

So sánh tổng thời gian truy vấn trước và sau khi đánh chỉ mục, thấy rằng, việc đánh chỉ mục đã tiết kiệm được rất nhiều thời gian truy vấn. Tóm lại, công cụ đánh chỉ mục đã giúp ích rất nhiều trong quá trình thực hiện truy vấn của hệ thống. Thời gian mà hệ thống phải dành ra để thực hiện truy vấn giảm đi được một lượng đáng kể. Tuy nhiên, chúng ta chỉ nên đánh chỉ mục đối với những bảng, những cột thường xuyên được sử dụng cho mục đích tìm kiếm dữ liệu.

2.5. Truy vấn trong cơ sở dữ liệu không gian

2.5.1. Mô tả về cơ sở dữ liệu không gian

• Bảng bc_pubs

Column	Type	Description
gid	integer	Unique ID
id	integer	Unique ID
name	character varying	Tên Pub
address	character varying	Địa chỉ phố
city	character varying	Tên thành phố
province	character varying	Quận / huyện
postal	character varying	Mã bưu điện
the_geom	geometry	mô tả hình học (Point)

• Bảng bc_voting_areas

Column	Type	Description
gid	integer	Unique ID
mã	character varying	mã bầu cử
id	character varying	Area ID
riding	character varying	tên khu vực bầu cử
region	character varying	tên vùng
number	character varying	số vùng tham gia bầu cử
ndp	integer	# of NDP Votes
liberal	integer	# of Liberal Votes

green	integer	# of Green Votes
unity	integer	# of Unity Votes
vtotal	integer	tổng phiếu
vreject	integer	# of Spoiled Ballots
vregist	integer	# of Registered Voters
the_geom	geometry	mô tả hình học(Polygon)

• **Bảng bc_roads**

Column	Type	Description
gid	integer	Unique ID
name	character varying	Road Name
the_geom	geometry	mô tả hình học (Linestring)

• **Bảng bc_border**

Column	Type	Description
gid	integer	Unique ID
border_id	integer	border Name
the_geom	geometry	mô tả hình học (Linestring)

• **Bảng bc_hospitals**

Column	Type	Description
gid	integer	Unique ID
id	integer	Unique ID
authority	character varying	người đứng đầu
name	character varying	Hospital Name
the_geom	geometry	mô tả hình học (Point)

- **Bảng bc_municipality**

Column	Type	Description
gid	integer	Unique ID
mã	integer	Unique ID
name	character varying	City / Town Name
the_geom	geometry	Location Geometry (Polygon)

2.5.2. Truy vấn

- **Sử dụng nhóm hàm đo lường**

Sử dụng hàm ST_Area() để tính diện tích của các thành phố có trong bảng bc_municipality. Giá trị trả về của hàm ST_Area() là kiểu Numeric.

huongnghiem=>SELECT ST_Area(the_geom) FROM bc_municipality;

VD1 : Tính diện tích của thành phố PRINCE GEORGE, tính theo đơn vị hectar?

```
SELECT ST_Area(the_geom)/10000 AS hectares
FROM bc_municipality
WHERE name = 'PRINCE GEORGE';
      hectares
-----
32657.9103824927
(1 row)
```

VD2: Tìm ra đô thị có diện tích lớn nhất trong tỉnh?

```
SELECT name, ST_Area(the_geom)/10000 AS hectares
FROM bc_municipality
ORDER BY hectares DESC
LIMIT 1;
      name      | hectares
-----+-----
```

```
TUMBLER RIDGE | 155020.02556131
(1 row)
```

VD3 : Tính tổng diện tích tất cả các vùng có tham gia bầu cử, tính theo đơn vị hectar?

```
SELECT Sum(ST_Area(the_geom))/10000 AS hectares
FROM bc_voting_areas;
      hectares
-----
94759319.6833071
(1 row)
```

VD4 : Tổng diện tích của các vùng có tham gia bầu cử, có số người tham gia bầu cử >100?

```
SELECT Sum(ST_Area(the_geom))/10000 AS hectares
FROM bc_voting_areas
WHERE vtotal > 100;
      hectares
-----
36609425.2114911
(1 row)
```

Sử dụng hàm ST_Perimeter() để tính chu vi của đối tượng có kiểu POLYGON hoặc MULTIPOLYGON. Áp dụng, tính chu vi của các thành phố liệt kê trong bảng bc_municipality. Giá trị trả về của hàm ST_Perimeter() là kiểu Numeric.

huongnghiem=>SELECT ST_Perimeter(the_geom) FROM bc_municipality;

VD1 : Tính chu vi của đô thị thuộc VANCOUVER

```
SELECT ST_Perimeter(the_geom)
FROM bc_municipality
WHERE name = 'VANCOUVER';
      st_perimeter
-----
```

```
57321.7782018048
(1 row))
```

Sử dụng hàm ST_Length() để tính chiều dài của các đối tượng có kiểu LINESTRING, MULTILINESTRING. Áp dụng, tính chiều dài các con đường được liệt kê trong bảng bc_roads. Giá trị trả về của hàm ST_Length() là kiểu Numeric.

huongnghiem=>SELECT ST_Length(the_geom) FROM bc_roads;

VD1 : Tính tổng độ dài tất cả các con đường có trong tỉnh?

```
SELECT Sum(ST_Length(the_geom))/1000 AS km_roads
FROM bc_roads;
      km_roads
-----
70842.1243039643
(1 row)
```

VD2 : Tính độ dài của đường có tên là Douglas St?

```
SELECT Sum(ST_Length(the_geom))/1000 AS kilometers
FROM bc_roads
WHERE name = 'Douglas St';
      kilometers
-----
19.8560819878386
(1 row)
```

Sử dụng hàm ST_Distance() để tính khoảng cách giữa các đối tượng POINT/POINT, POINT/LINESTRING, LINESTRING/LINESTRING. Áp dụng để tính khoảng cách giữa hospitals và pubs có gid bằng nhau:

*Huongnghiem=>SELECT ST_Distance(hos.the_geom,
pub.the_geom) FROM bc_hospitals as hos, bc_pubs as pub
WHERE hos.gid = pub.gid;*

5.2.2. Nhóm hàm so sánh

ST_Intersects(geometryA, geometryB) trả về kiểu True nếu các hình giao nhau

```
Huongnghiem=> SELECT ST_Intersects(voting.the_geom, mun.the_geom) FROM  
bc_voting_areas AS voting, bc_municipality AS mun WHERE  
mun.name='TUMBLER RIDGE';
```

ST_Contains(geometryA, geometryB) trả về kiểu True nếu hình A chứa hình B

```
Huongnghiem=> SELECT ST_Contains(voting.the_geom, mun.the_geom) FROM  
bc_voting_areas AS voting, bc_municipality AS mun WHERE  
mun.name='TUMBLER RIDGE';
```

ST_Within(geometryA, geometryB) trả về True nếu hình A ở bên trong hình B với một khoảng cách xác định :

VD : Tìm tất cả các vị trí của pub trong vòng 250m so với hospital

```
Huongnghiem=> SELECT h.name, p.name FROM bc_hospitals AS h, bc_pubs AS  
p WHERE ST_Dwithin(h.the_geom, p.the_geom, 250);
```

ST_IsValid(geometry) trả về True nếu hình hợp lệ

```
Huongnghiem=> SELECT gid FROM bc_voting_areas WHERE NOT  
ST_IsValid(the_geom);  
Giá trị trả về của gid : 4897
```

ST_Relate(geometryA, geometryB) trả về kiểu Strings – tên mỗi quan hệ giữa hai hình.

• Sử dụng nhóm hàm trả về đối tượng

Đây là nhóm hàm có chức năng tìm ra mối quan hệ giữa 2 đối tượng. Quan hệ đó là quan hệ giao nhau, sự khác nhau giữa 2 đối tượng, phép hợp 2 đối tượng...

VD : Tìm ra giao giữa các vùng có tham gia bầu cử (thuộc bảng bc_voting_areas) và các đô thị (thuộc bảng bc_municipality) thuộc thành phố PRINCE GEORGE.

```
SELECT ST_AsText(ST_Intersection(v.the_geom, m.the_geom)) AS
FROM bc_voting_areas AS v, bc_municipality AS m

WHERE ST_Intersects(v.the_geom,m.the_geom) AND m.name='PRINCE
GEORGE';
```

VD : tạo bảng chứa thông tin của tất cả các vùng có tham gia bầu cử và tên của thành phố thỏa mãn yêu cầu là vùng tham gia bầu cử giao với thành phố có tên là PRINCE GEORGE, và 2 vùng này phải giao nhau

```
CREATE TABLE pg_voting_areas AS
SELECT
  ST_Intersection(v.the_geom, m.the_geom) AS intersection_geom,
  ST_Area(v.the_geom) AS va_area,
  v.*,
  m.name
FROM
  bc_voting_areas v,
  bc_municipality m
WHERE
  ST_Intersects(v.the_geom, m.the_geom) AND
  m.name = 'PRINCE GEORGE';
```

Tính diện tích của vùng giao nhau tìm được trong bảng vừa tạo

```
SELECT Sum(ST_Area(intersection_geom))
FROM pg_voting_areas;

      sum
-----
326579103.824927
(1 row)
```

Chương 3. MỞ RỘNG TRUY VẤN KHÔNG GIAN POSTGRESQL

Đối với một hệ thống quan hệ chuẩn, chúng có thể lưu trữ thông tin về CSDL, bảng, cột... và thường được gọi là hệ thống catalog. Một điểm khác biệt giữa Postgres và hệ thống quan hệ chuẩn là Postgres có thể lưu trữ được rất nhiều thông tin bên trong catalog, không chỉ thông tin về bảng, cột mà còn thông tin về kiểu dữ liệu, hàm, phương thức truy cập. Người dùng có thể sửa đổi được bảng dữ liệu, và cơ sở hoạt động trên các bảng biểu, Postgres được coi là có thể mở rộng bởi người dùng.

Postgres không giống như bộ quản lý dữ liệu khác. Server của Postgres có thể kết hợp mã do người dùng viết vào bên trong hệ thống thông qua bộ nạp động. Điều đó có nghĩa là, người dùng có thể chỉ định đối tượng mã tập tin, thực thi kiểu dữ liệu hoặc hàm mới, và Postgres sẽ tải nó vào hệ thống theo yêu cầu.

Có thể nói, PostgreSQL là một CSDL linh hoạt và có tính mở rộng. Một mặt, PostgreSQL có thể sử dụng cho nhiều mục đích. Mặt khác, PostgreSQL có thể dễ dàng được mở rộng và cung cấp nhiều giao diện lập trình được thiết kế để mở rộng các tính năng cốt lõi của PostgreSQL. Chúng ta có thể thêm nhiều hàm mới, nhiều toán tử mới và tùy chọn kiểu dữ liệu cho PostgreSQL, và những khả năng trên là hoàn toàn dễ dàng làm được.

Như chúng ta đã biết, PostGIS là một modul được kết hợp trong PostgreSQL cho phép người dùng lưu trữ các lớp dữ liệu không gian. Không những thế, nó còn cho phép người dùng truy vấn, xử lý dữ liệu không gian. Tuy nhiên, hầu hết các hỗ trợ hàm thao tác, và các phép truy vấn trong PostGIS được thực hiện trong hệ không gian 2 chiều. Do đó, điều mong muốn là chúng ta có thể viết ra các kiểu hiển thị đối tượng trong không gian 3 chiều, viết các hàm mở rộng và toán tử mở rộng để có thể thực hiện tính toán các phép toán trong không gian 3 chiều. Vì vậy, việc nghiên cứu tìm hiểu cách viết các mở rộng về kiểu, hàm và toán tử trong PostgreSQL sẽ giúp chúng ta tạo ra các kiểu dữ liệu, kiểu hàm mới phục vụ cho việc thao tác CSDL trong không gian 3 chiều.

3.1. Các kiểu dữ liệu trong PostgreSQL

Kiểu dữ liệu của PostgreSQL được chia ra thành các kiểu : kiểu dữ liệu cơ bản, kiểu dữ liệu hỗn hợp.

3.1.1. Kiểu dữ liệu cơ bản

Kiểu dữ liệu cơ bản như *int4*, là những kiểu dữ liệu cơ bản trong PostgreSQL. Nhìn chung, chúng tương ứng với những gì thường được biết đến như là kiểu dữ liệu trừu tượng. PostgreSQL chỉ có thể hoạt động trên các kiểu các hàm được người dùng cung cấp và chỉ hiểu cách vận hành của các kiểu đó đến mức mà người dùng mô tả chúng.

3.1.2. Kiểu dữ liệu hỗn hợp

Kiểu dữ liệu hỗn hợp là kiểu dữ liệu được xây dựng dựa trên các kiểu dữ liệu cơ bản khác, và do đó, các hàm bổ sung luôn sẵn sàng để báo cho CSDL biết kiểu dữ liệu được sử dụng như thế nào?

Ví dụ, xây dựng một kiểu dữ liệu hỗn hợp có tên là *employee* gồm các thuộc tính : *name, salary, age, room*. Kiểu dữ liệu được biểu diễn như sau :

```
CREATE TABLE employee ( Name text, Salary numeric, Age integer, Room integer );
```

3.2. Mở rộng PostgreSQL với hàm tùy chọn

Một hệ thống CSDL phức tạp, quan trọng hơn là sự tồn tại của các quy tắc và các quy ước làm cho mã rõ ràng và dễ hiểu hơn nhiều. Đối với PostgreSQL, một số quy tắc cơ bản đã được đưa ra việc cân nhắc khi thực thi hàm được thêm. Đây là yếu tố quan trọng làm cho hệ thống dễ dàng hiểu hơn. Trước khi chúng ta gọi các quy ước, hãy xem các hàm sẽ được thực thi như thế nào?

PostgreSQL cung cấp hai kiểu hàm : *hàm ngôn ngữ truy vấn* (hàm viết bởi SQL) và *hàm ngôn ngữ lập trình* (hàm viết bởi ngôn ngữ lập trình được biên dịch như ngôn ngữ lập trình C). Mọi loại hàm có thể dùng kiểu dữ liệu cơ bản, kiểu dữ liệu hỗn hợp hoặc kết hợp chúng. Thêm nữa, mọi loại hàm có thể có thể trả về một kiểu dữ liệu cơ bản hoặc một kiểu dữ liệu hỗn hợp. Các hàm có thể được định nghĩa để trả về một tập giá trị cơ bản hoặc giá trị hỗn hợp.

3.2.1. Hàm ngôn ngữ truy vấn (SQL)

Vì PostgreSQL là phần mềm rất linh hoạt, và ta có thể dễ dàng thêm hàm vào CSDL. Dùng mã SQL để viết hàm bổ sung cho PostgreSQL, và chúng ta sẽ thấy khả năng thực thi các hàm đơn giản bằng cách sử dụng mã SQL thông thường. Và, sử dụng SQL để viết hàm bổ sung là việc dễ dàng như sử dụng bất kỳ ngôn ngữ lập trình nào khác.

Hàm SQL thực thi một danh sách các câu lệnh SQL tùy ý, trả về kết quả truy vấn cuối cùng trong danh sách. Trong trường hợp đơn giản, hàng đầu tiên của kết quả truy vấn cuối cùng sẽ được trả về (lưu ý rằng, hàng đầu tiên của nhiều hàng kết quả là không định nghĩa rõ, trừ khi bạn sử dụng mệnh đề ORDER BY). Nếu truy vấn cuối cùng xảy ra, không trả lại hàng nào, thì giá trị null sẽ được trả về.

Ngoài ra, hàm SQL có thể được khai báo để trả về một tập, bằng cách xác định kiểu trả về của hàm bằng SETOF, hoặc bằng cách RETURN TABLE (cột). Trong trường hợp này, tất cả các hàng của kết quả truy vấn cuối cùng được trả về.

Thân của một hàm SQL phải là một danh sách các câu lệnh SQL được cách nhau bằng dấu chấm phẩy. Dấu chấm phẩy sau câu lệnh cuối cùng, trừ khi, hàm được khai báo trả về *void*, câu lệnh cuối cùng phải là SELECT, INSERT, UPDATE, DELETE.

• **Cú pháp :**

```
CREATE FUNCTION name ( [ argumenttype [, ...] ] )  
  
RETURNS returntype  
  
AS 'definition'  
  
LANGUAGE 'language_name'
```

Giải thích :

- CREATE FUNCTION *name* ([*argumenttype* [, ...]]) : *name* là tên của hàm mới sẽ được tạo ra. Bên trong hàm là các kiểu dữ liệu của đối số sẽ được truyền vào, cách nhau bởi dấu phẩy. Đối số trong hàm SQL được tham chiếu trong thân hàm SQL sử dụng cú pháp : *\$n*. *\$1* nghĩa là đối số thứ nhất, *\$2* nghĩa là đối số thứ hai...*\$n* là đối số thứ n. Nếu đối số thuộc kiểu dữ liệu hỗn hợp thì chúng ta phải khai báo theo cú pháp: *\$1.name*, khi đó, có thể truy cập thuộc tính của đối số. Chúng ta có thể để trống bên trong hàm nếu hàm đó không yêu cầu nhập đối số.
- RETURNS *returntype* : kiểu dữ liệu trả về là kiểu dữ liệu duy nhất của giá trị mà nó được trả về bởi hàm.
- AS '*definition*' : nội dung của hàm.
- LANGUAGE '*language_name*' : tên ngôn ngữ dùng để viết hàm.

- Ví dụ

Hàm SQL với kiểu dữ liệu cơ bản : Các kiểu dữ liệu cơ bản thường dùng như *integer, float, text...*, chúng thường được dùng để khai báo kiểu cho đối số đối với những hàm có đối số truyền vào, hoặc dùng để khai báo kiểu cho giá trị trả về của hàm có hoặc không có đối số.

Viết hàm tính tổng của 2 số nguyên, hàm có tên là *add_em(integer, integer)*, có 2 đối số kiểu *integer* được truyền vào, giá trị trả về của hàm là kiểu *integer*.

```
CREATE FUNCTION add_em(integer, integer) RETURNS integer  
AS 'SELECT $1 + $2' LANGUAGE 'sql';
```

Truy vấn : `SELECT add_em(1,2) AS answer;`

Kết quả truy vấn : `answer=3;`

Hàm SQL sử dụng kiểu dữ liệu hỗn hợp : là kiểu dữ liệu do người dùng định nghĩa, nó cũng bao hàm việc sử dụng các kiểu dữ liệu cơ bản cho việc khai báo các thuộc tính của kiểu dữ liệu hỗn hợp. Nghĩa là, không chỉ định ra đối số mà còn phải chỉ ra thuộc tính của đối số đó.

Viết hàm tính lương gấp đôi của nhân viên. Hàm *double_salary(employee)* : đối số truyền vào có kiểu dữ liệu hỗn hợp *employee(name, salary, age, room)* (được định nghĩa ở phần 1.2), giá trị trả về có kiểu *numeric*.

```
CREATE FUNCTION double_salary(employee) RETURNS numeric  
AS 'SELECT $1.salary * 2 AS salary;' LANGUAGE 'sql';
```

Truy vấn : `SELECT name, double_salary(employee.*) FROM employee`

```
WHERE employee.room = point '(2,1)';
```

Chú ý : cú pháp `$1.salary` . `$1` là đối số đầu tiên cũng là duy nhất của hàm *double_salary(employee)* và `$1.salary` là lấy thuộc tính *salary* của đối số truyền vào.

Hàm SQL sử dụng tham số đầu ra: nếu theo cú pháp tạo hàm bằng câu lệnh SQL có chứa từ khóa `RETURNS` dùng để trả về kiểu dữ liệu của hàm. Nhưng thay vào sử dụng từ khóa `RETURNS` chúng ta sử dụng tham số đầu ra. Theo dõi ví dụ :

```
CREATE FUNCTION add_em(IN x int, INT y int , OUT sum
int) AS 'SELECT $1+$2' LANGUAGE SQL;
```

```
CREATE FUNCTION add_em(int x, int y) RETURNS int AS `
SELECT $1+$2' LANGUAGE SQL;
```

2 hàm này có chức năng tương tự nhau là tính tổng 2 số nguyên, tuy nhiên cách tạo hàm thứ nhất sử dụng từ khóa RETURNS để trả lại kết quả của hàm, còn cách tạo thứ hai thì lại dùng tham số đầu ra. Khi sử dụng hàm cũng giống nhau, tức là cùng có 2 tham số kiểu int được truyền vào. Tuy nhiên, khi chúng ta thực hiện xóa hàm đó thì cần phải phân biệt:

```
DROP FUNCTION add_em(x int, y int, OUT sum int);
```

```
DROP FUNCTION add_em(int, int);
```

Hàm SQL sử dụng giá trị mặc định cho đối số: hàm có thể được định nghĩa với giá trị mặc định cho một số hoặc tất cả các đối số truyền vào. Giá trị mặc định được chèn vào khi hàm được gọi truyền thiếu tham số, tất nhiên, chỉ có thể thiếu tham số đã gán giá trị mặc định, nếu thiếu các tham số khác thì hệ thống sẽ báo lỗi do bạn truyền không đủ tham số. Nếu hàm được truyền đủ tham số thì giá trị mặc định sẽ không được sử dụng. Ví dụ:

```
CREATE FUNCTION foo(a int, a int DEFAULT 2, c int
DEFAULT 3) RETURN int LANGUAGE SQL AS 'SELECT $1+$2+$3';
```

```
SELECT foo(10,20,30); → Kết quả trả về là 60
```

```
SELECT foo(10,20); → Kết quả trả về là 33
```

```
SELECT foo(10); → Kết quả trả về là 15
```

Hàm SQL truy vấn từ bảng : tất cả các hàm SQL có thể được sử dụng trong mệnh đề FROM của một truy vấn, nhưng nó đặc biệt có tác dụng cho hàm trả về kiểu dữ liệu hỗn hợp. Nếu hàm được định nghĩa trả về kiểu dữ liệu cơ bản thì bảng trả về sẽ là một cột, còn nếu hàm được định nghĩa trả về kiểu dữ liệu hỗn hợp thì bảng trả về sẽ bao gồm nhiều cột, mỗi cột là một thuộc tính của kiểu hỗn hợp. Ví dụ : CREATE TABLE foo (fooid int, foosubid int, fooname text);

```
INSERT INTO foo VALUES (1, 1, 'Joe');
```

```
INSERT INTO foo VALUES (1, 2, 'Ed');
```

```

INSERT INTO foo VALUES (2, 1, 'Mary');

CREATE FUNCTION getfoo(int) RETURNS foo AS $$
    SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;

SELECT *, upper(foiname) FROM getfoo(1) AS t1;

```

```

fooid | foosubid | foiname | upper
-----+-----+-----+-----
      1 |          1 | Joe      | JOE

```

Hàm SQL trả về một tập: khi một hàm SQL được đưa ra kiểu trả về SETOF, kết quả truy vấn cuối cùng của hàm được hoàn thành thì mỗi hàng ở đầu ra của nó sẽ được trả lại như yếu tố của tập kết quả.

```

CREATE FUNCTION getfoo(int) RETURNS SETOF foo AS $$
    SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;

SELECT * FROM getfoo(1) AS t1;

fooid | foosubid | foiname
-----+-----+-----
      1 |          1 | Joe
      1 |          2 | Ed
(2 rows)

```

3.2.2. Hàm sử dụng ngôn ngữ lập trình C

Việc thực thi phần mở rộng PostgreSQL thực sự nhanh hơn khó có thể đạt được bằng cách sử dụng bất cứ điều gì khác hơn C. Vì PostgreSQL được viết hoàn toàn bằng ngôn

ngữ C, điều này có vẻ hợp lý. Viết mã bằng ngôn ngữ C có thể không phải là cách nhanh nhất của việc thực thi các tính năng, nhưng việc thực hiện các hàm sẽ không bị ảnh hưởng nhiều mà các ngôn ngữ lập trình khác gây ra.

Hàm do người dùng định nghĩa có thể viết bằng ngôn ngữ lập trình C hoặc ngôn ngữ lập trình cao cấp hơn C như C++. Thông thường, hàm do người dùng định nghĩa được thêm vào PostgreSQL bằng cách sử dụng bộ nạp đối tượng (thư viện chia sẻ). Các thư viện chia sẻ được nạp tại thời điểm chạy (khi hàm được gọi lần đầu tiên) và ở lại trong bộ nhớ cho phần còn lại của phiên làm việc. Những hiểu biết này thực sự quan trọng cho mục đích gỡ lỗi. Nếu bạn muốn kiểm tra mở rộng của bạn với sự giúp đỡ của *psql*, cần kết nối lại với CSDL trước khi biên dịch lại và bổ sung module vào CSDL của bạn. Nếu không, các đối tượng cũ vẫn còn trong bộ nhớ.

Bảng dưới chỉ ra kiểu trong C tương ứng với kiểu trong SQL khi viết hàm bằng ngôn ngữ C sử dụng để tích hợp kiểu của PostgreSQL.

Bảng 3-1 : Danh sách kiểu dữ liệu trong SQL và trong C

Kiểu SQL	Kiểu C	Thư viện khai báo trong PostgreSQL
Abstime	AbsoluteTime	Utils/nabstime.h
Boolean	Bool	Postgres.h
Box	BOX*	Utils/geo_decls.h
Bytea	Bytea*	Postgres.h
“char”	Char	
Character	BpChar*	Postgres.h
cid	CommandId	Postgres.h
Date	DateADT	Utils/date.h
Smallint(int2)	Int2 or int16	Postgres.h

Int2vector	Int2vector*	Postgres.h
Integer(int4)	Int4 or int32	Postgres.h
Real(float4)	Float4*	Postgres.h
Doubl precision (float8)	Float8*	Postgres.h
Interval	Interval*	Utils/timestamp.h
Lseg	LSEG*	Utils/geo_decls.h
Name	Name	Postgres.h
Oid	Oid	Postgres.h
Oidvector	Oidvector*	Postgres.h
Path	PATH*	Utils/geo_decls.h
Point	POINT*	Utils/geo_decls.h
Regproc	Regproc	Postgres.h
Reltime	relaticeTiem	Utils/nabstime.h
Text	Text*	Postgres.h
Tid	ItemPointer	Storage/itemptr.h
Time	timeADT	Utils/date.h
Time with time zone	TimeTzADT	Utils/date.h
Timestamp	Timestamps*	Utils/timestamp.h
Tinterval	timeInterval	Utils/nabstime.h

Varchar	varChar*	Postgres.h
Xid	TransactionId	Postgres.h

Sẽ có hai giai đoạn phải làm để thêm hàm mở rộng vào PostgreSQL. Đầu tiên, chúng ta viết hàm mở rộng bằng ngôn ngữ mình chọn, chúng ta chọn ngôn ngữ C, sau đó biên dịch chúng vào bộ nạp đối tượng (file .dll nếu ở trong Window và file .so nếu ở trong Linux/Unix). Tiếp theo, hãy cho postgresQL biết về hàm mở rộng, sử dụng lệnh CREATE FUNCTION để thêm hàm vào trong cơ sở dữ liệu. Nó là hai giai đoạn chính để chúng ta có thể thêm hàm mở rộng vào PostgreSQL, chi tiết các giai đoạn sẽ được tìm hiểu kỹ hơn ở các phần tiếp theo.

a.Cách viết hàm bằng ngôn ngữ C

Có một số bước cần thiết để viết một hàm mở rộng cho PostgreSQL bằng ngôn ngữ C. Khi chúng ta gọi một hàm trong chương trình C cơ bản, chúng ta cần phải biết tại thời điểm nào mã chương trình sẽ làm thế nào để gọi hàm đó. Chúng ta cần biết bao nhiêu đối số được yêu cầu và biết được kiểu dữ liệu của mỗi đối số đó. Nếu chúng ta cung cấp số lượng đối số không chính xác hoặc không đúng kiểu dữ liệu, khi đó chương trình của chúng ta sẽ không thể thực thi.

Ngoài ra, phiên bản của PostgreSQL cũng ảnh hưởng rất lớn đến quá trình viết mã chương trình. PostgreSQL tồn tại 2 phiên bản đó là Version 0 và Version 1. Trong phạm vi của khóa luận, tôi chỉ đưa ra bàn bạc với Version 1.

Để viết và thực thi một hàm mở rộng cho PostgreSQL được viết bằng ngôn ngữ C hoạt động tốt thì chúng ta cần chú ý một số quy tắc viết. Có thể nói, đây là những quy tắc cơ bản nhất mà người lập trình viết hàm mở rộng cần biết. Nó sẽ giúp ích rất lớn cho việc viết và xây dựng hàm C. Những quy tắc cơ bản đó là :

- Sử dụng *pg_config --includedir-server* để tìm kiếm các tập tin tiêu đề của PostgreSQL được cài đặt trên hệ thống.
- Khi phân bổ bộ nhớ, sử dụng các hàm trong PostgreSQL là *palloc* và *pfree* thay vì thư viện hàm trong C tương ứng là *malloc* và *free*. Bộ nhớ được phân bổ bởi

hàm *palloc* sẽ được trả lại một cách tự động tại cuối mỗi giao tác, ngăn chặn rò rỉ bộ nhớ.

- Hầu hết các kiểu bên trong PostgreSQL đều được khai báo trong tập tin *postgres.h*, còn các hàm quản lý giao diện được khai báo trong tập tin *fmgr.h*, vì vậy, chúng ta cần phải dùng đến ít nhất hai tập tin đó.
- Tên biến, các ký hiệu bên trong tập tin không được trùng nhau hoặc không được trùng với các ký hiệu, tên được định nghĩa trong PostgreSQL. Khi đó, bạn cần phải đổi tên hàm hoặc biến nếu hệ thống đưa ra lỗi trong trường hợp này.

Version 1

Trong cả 2 phiên bản của PostgreSQL thì, phần khai báo các thư viện hàm là việc không thể thiếu trong khi viết mã.

```
#include <postgres.h>

#include <fmgr.h>

#include <string.h> ...
```

Việc khai báo các thư viện hàm đảm bảo là khi biên dịch tập tin trong đó có sử dụng các kiểu, các hàm hỗ trợ của PostgreSQL thì hệ thống có thể dễ dàng xác định được các kiểu, các hàm hỗ trợ đó ở đâu mà có. Ví dụ, để sử dụng hàm `PG_FUNCTION_ARGS` thì chúng ta cần phải khai báo thư viện hàm có tên *fmgr.h*. Về cơ bản, khi viết hàm mở rộng cho PostgreSQL bằng ngôn ngữ C, thì trong hàm đó luôn luôn cần khai báo hai thư viện hàm *postgres.h* và *fmgr.h*. Tất nhiên, PostgreSQL còn có nhiều thư viện hàm khác, nhưng bạn chỉ nên khai báo khi bạn thực sự cần đến nó, chẳng hạn là phát triển mã mã phức tạp hơn.

Version 1 hỗ trợ cả việc gọi các quy ước, và Version 1 hỗ trợ nhiều tính năng và hoạt động linh hoạt hơn. Có hai hệ quả quan trọng đối với Version 1. Đầu tiên, tất cả các hàm trong Version 1 đều trả về kiểu dữ liệu giống nhau là : *Datum*. Một *Datum* là kiểu dữ liệu phổ biến. Bất kỳ kiểu dữ liệu trong PostgreSQL đều có thể được truy cập thông qua *Datum*. PostgreSQL cung cấp một tập lớn các macro để nó có thể dễ dàng làm việc với *Datum*. Thứ hai, hàm trong Version 1 sử dụng tập các macro để truy cập các đối số của hàm. Mọi hàm trong Version 1 được khai báo theo cách sau :

```
Datum function-name (PG_FUNCTION_ARGS);
```

Một khối :

```
PG_FUNCTION_INFO_V1(function_name);
```

PG_FUNCTION_INFO_V1 là hàm quản lý những thay đổi gần đây được xây dựng với trình biên dịch ngôn ngữ đa dạng. Nó cần được sử dụng trong tất cả các tập tin. Tuy nó không cần thiết cho các hàm bên trong postgresSQL nhưng nó cần được khai báo để sử dụng cho các hàm “bộ nạp động”.

Mỗi đối số cụ thể được lấy ra bằng cách sử dụng hàm *PG_GETARG_xxx()* tương ứng với kiểu dữ liệu của mỗi đối số đó. Với xxx là kiểu dữ liệu cụ thể được liệt kê trong bảng...các tham số đầu vào của hàm PG_GETARG_xxx() được tính bắt đầu từ số 0. Lưu ý rằng, tham số truyền vào phải đảm bảo không phải là giá trị NULL.

VD: `int32 arg = PG_GETARG_INT32(0);`

```
text *t = PG_GETARG_TEXT_P(0);
```

Hậu tố `_P` nghĩa là ‘một con trỏ’.

Khai báo kiểu int23 cho nhiều biến :

```
int32 arg1 = PG_GETARG_INT32(0);
```

```
int32 arg2 = PG_GETARG_INT32(1);...
```

Ngoài ra, hàm *PG_GETARG_xxx_COPY()* đảm bảo việc sao chép đối số được chỉ định được an toàn trong việc ghi chép vào đó.

Để tính giá trị trả về của các đối số chúng ta sử dụng hàm *PG_RETURN_xxx()*. Tương tự với hàm *PG_GETARG_xxx()* thì xxx cũng là kiểu dữ liệu được liệt kê trong bảng. Trong trường hợp, hàm trả về giá trị NULL, chúng ta sử dụng hàm *PG_RETURN_NULL()*. Chú ý, khi chúng ta sử dụng đồng thời hai hàm *PG_GETARG_xxx()* và *PG_RETURN_xxx()* thì kiểu xxx bắt buộc phải trùng nhau. Giả sử chúng ta có hàm *add_one()* với mục đích là cộng 1 đơn vị cho đối số truyền vào, đối số này có kiểu dữ liệu là *int32*. Vì thế, hàm *PG_GETARG_INT32()* và hàm *PG_RETURN_INT32()* có cùng kiểu dữ liệu là *int32*:

```
Datum add_one (PG_FUNCTION_ARGS)
```



```

{
    int32 arg = PG_GETARG_INT32(0);
    PG_RETURN_INT32(arg + 1);
}

```

Để kiểm tra đối số truyền vào có phải là giá trị NULL hay không, chúng ta sử dụng hàm `PG_ARGISNULL(n)`. Nếu `n=0` nghĩa là `PG_ARGISNULL(0)` trả về đúng.

Các ký hiệu `VARDATA`, `VARSIZE`, `VARHDRSZ`, `VARATT_xxx` được định nghĩa trong file “`postgres.h`” được sử dụng để truy cập các yếu tố của các kiểu biến dữ liệu cấu trúc ví dụ như kiểu `TEXT`. Ký hiệu `VARHDRSZ` là hằng số chứa kích thước một phần cố định của kiểu dữ liệu cấu trúc. `VARSIZE()` trả về kích thước của toàn bộ kiểu dữ liệu cấu trúc. `VARDATA()` thì trả về một con trỏ đến byte đầu tiên của giá trị `TEXT`. Chiều dài của giá trị `TEXT` sẽ là `VARSIZE() – VARHDRSZ`.

Để cấp phát bộ nhớ hoặc trả bộ nhớ biến ta sử dụng hàm `palloc()` và `pfree()`. `Palloc()` có chức năng tương tự như `malloc()` : nó dùng để cấp phát số lượng byte được yêu cầu và trả về con trỏ đến khoảng không mới. Hàm `palloc()` được dùng nhiều do nó ngăn chặn được sự rò rỉ của bộ nhớ, tức là sau khi cấp phát bộ nhớ cho biến, biến đã sử dụng xong bộ nhớ đó PostgreSQL có thể tự động thu hồi lại bộ nhớ, đảm bảo bộ nhớ không bị lãng phí. Do vậy, chúng ta nên sử dụng hàm `palloc()` và `pfree()` khi viết hàm mở rộng hơn là sử dụng hàm `malloc()` và hàm `free()`. không

b. Thao tác biên dịch file

Công việc vô cùng cần thiết theo sau việc viết mã C là công việc biên dịch file. Như đã biết, sử dụng được file viết bằng ngôn ngữ C thì chúng ta cần biên dịch chúng thành file có mở rộng là “`.dll`” nếu thao tác trong môi trường Window còn file có mở rộng là “`.so`” nếu thao tác file trong môi trường Linux/Unix. PostgreSQL cung cấp việc xây dựng cơ sở cho phần mở rộng, được gọi là PGXS, vì thế, modul mở rộng đơn giản có thể xây dựng lại một cách đơn giản việc cài đặt server . Lưu ý rằng, cơ sở này không có dụng ý cho việc xây dựng hệ thống framework mà có thể sử dụng để xây dựng tất cả các phần mềm giao diện cho PostgreSQL;

Để làm biên dịch từ file “`.c`” sang file “`.so`” trong môi trường Linux/Unix chúng ta cần thực hiện các bước sau :

B1 : tạo file có tên là Makefile với nội dung sau :

```
MODULES = file_name  
PGXS := $(shell pg_config --pgxs)  
Include $(PGXS)
```

Giải thích :

MODULES : liệt kê các đối tượng chia sẻ được xây dựng từ tập tin nguồn tương tự.

PG_CONFIG : đường dẫn đến chương trình pg_config cho việc cài đặt postgresQL để xây dựng lại.

File_name : là tên file cần biên dịch (không bao gồm phần mở rộng)

B2 : chạy lệnh “make”, sau đó chạy lệnh “make install”

Sau khi thực hiện xong hai bước trên, file “.c” sẽ được biên dịch sang dạng file “.so”.

Mặc định, hàm mở rộng được biên dịch và cài đặt cho postgresQL tương ứng với chương trình pg_config đầu tiên được tìm thấy trong đường dẫn của bạn.

c. Thao tác tạo và sử dụng hàm.

Sau khi đã biên dịch file .c thành file .so, đến đây tạo và sử dụng hàm mở rộng không còn là việc khó khăn. Việc tạo và sử dụng hàm mở rộng được thao tác bằng ngôn ngữ SQL. Như đã giới thiệu phần trên, để tạo hàm mở rộng chúng ta dùng lệnh CREATE FUNCTION

```
VD: CREATE FUNCTION add_one(integer) RETURNS integer  
AS 'funcs.so', 'add_one' LANGUAGE 'C' ;
```

Cú pháp lệnh :

```
CREATE FUNCTION [tên hàm] RETURNS [kiểu trả về] AS  
[file .so], [tên hàm sử dụng] LANGUAGE [ngôn ngữ]
```

Hàm mở rộng đã được tạo sau câu lệnh trên, cuối cùng là việc sử dụng hàm. Việc xây dựng hàm mở rộng trong postgresQL được thực hiện một lần, nhưng sẽ được sử dụng trong bất cứ lần khác bởi vì, sau khi biên dịch dạng file .so đã luôn tồn tại trong thư mục */usr/lib/pgsql/*

3.2.3. Kiểu dữ liệu do người dùng định nghĩa

Kiểu dữ liệu do người dùng định nghĩa luôn luôn xuất hiện ở đầu vào và đầu ra của hàm. Những hàm đó xác định kiểu dữ liệu sẽ xuất hiện như thế nào trong chuỗi (đầu vào hoặc đầu ra) và kiểu dữ liệu được tổ chức như thế nào trong bộ nhớ.

Để biểu diễn kiểu dữ liệu, chúng ta sử dụng cấu trúc trong ngôn ngữ C :

```
VD: typedef struct point3d {  
    float4 x;  
    float4 y;  
    float4 z;  
}point3d;
```

a. Định nghĩa hàm input và output trong C

- Định dạng bên trong, định dạng bên ngoài là gì?

Trước khi tìm hiểu về cách định nghĩa hàm nhập và xuất trong C, chúng ta cần có những hiểu biết về định dạng bên trong và định dạng bên ngoài của giá trị cần nhập.

Định dạng bên ngoài của một kiểu dữ liệu định nghĩa xem người dùng sẽ nhập dữ liệu như thế nào? Và giá trị được hiển thị cho người dùng biết là gì? Và có thể nói định dạng này được sử dụng để tương tác với người dùng.

Định dạng bên trong của kiểu dữ liệu định nghĩa xem giá trị sẽ được hiển thị bên trong CSDL như thế nào? Ví dụ, khi bạn nhập vào một giá trị số : 7218942 thì định dạng này sẽ được chuyển đổi từ dạng chuỗi sang dạng 4Byte có giá trị 00 6E 26 FE, và trong CSDL, giá trị số đó sẽ được lưu ở dạng 4byte đó. Vì vậy, định dạng bên trong của kiểu dữ liệu được định dạng bên trong CSDL.

- Nhưng tại sao lại có hai định dạng này?

Như đã biết, mỗi ngôn ngữ lập trình sẽ cung cấp những kiểu dữ liệu riêng, và người lập trình cũng chỉ hiểu và sử dụng những kiểu dữ liệu đó thông qua dạng hiển thị bên ngoài (có thể hiểu là định dạng bên ngoài). Ví dụ, trong ngôn ngữ lập trình C có định nghĩa kiểu dữ liệu *Int*, nó có thể lưu trữ số nguyên có miền xác định trong bộ biên dịch của ngôn ngữ này. Khi sử dụng kiểu dữ liệu *int*, người lập trình chỉ có cái nhìn về kiểu dữ

liệu này như : phạm vi biểu diễn là $[-32768, 32767]$ và kích thước là 2byte. Và khi nhập dữ liệu kiểu *int* cho các thao tác như cộng, trừ, nhân, chia, người lập trình chỉ cần nhập dạng số như “2+3” mà không cần quan tâm bộ biên dịch C sẽ xử lý thế nào đối với những số đã nhập vào. Mặt khác bộ biên dịch C cũng có thể hiểu được các phép thao tác như cộng, trừ, nhân, chia các giá trị số nguyên. Tuy nhiên, công việc của bộ biên dịch C là cần đưa ra các mã cần thiết để thực hiện các phép tính số học đó, và các mã đó có thể coi là định dạng bên trong của giá trị.

Sau khi định nghĩa kiểu dữ liệu, việc cần làm đó là định nghĩa hàm đầu vào và hàm đầu ra trong C. Mục đích của công việc này chính là xác định định dạng bên ngoài của kiểu dữ liệu sẽ như thế nào?

Ví dụ như, định nghĩa kiểu dữ liệu Complex theo cú pháp :

```
Typedef struct Complex {  
    Double x;  
    Double y;  
} Complex;
```

Ta có, chuỗi hiển thị bên ngoài của kiểu dữ liệu dạng (x,y).

Để có được dạng hiển thị bên ngoài như trên, chúng ta cần tạo ra hàm đầu vào và đầu ra cho kiểu Complex. Các hàm đầu vào và đầu ra thông thường không khó để viết, đặc biệt là hàm đầu ra. Nhưng, khi định nghĩa chuỗi hiển thị bên ngoài của kiểu dữ liệu, bạn cần phải xác định và phân tích cú pháp được đưa ra trong hàm đầu vào.

Theo dõi hàm đầu vào :

```
PG_FUNCTION_INFO_V1(complex_in);  
Datum complex_in(PG_FUNCTION_ARGS)  
{  
    char *str = PG_GETARG_CSTRING(0);  
    double x,  
    y;  
    Complex *result;
```

```

        if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2)
            ereport(ERROR,
                (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
                errmsg("invalid input syntax for complex: \"%s\"", str)));
        result = (Complex *) palloc(sizeof(Complex));
        result->x = x;
        result->y = y;
        PG_RETURN_POINTER(result);
    }

```

Đối với hàm đầu vào, chúng ta cần quan tâm đến số thuộc tính của kiểu dữ liệu đã định nghĩa. Ví dụ, đối với kiểu Complex đã định nghĩa ở trên, chúng có 2 thuộc tính x và y. Vì vậy, số thuộc tính được nhập trong chuỗi bắt buộc phải là 2 mới hợp lệ. Do đó, có lệnh để kiểm tra số thuộc tính đã được nhập trong chuỗi có bằng 2 hay không?

```

if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2) //quét chuỗi,
//đưa ra số thuộc tính nhập và kiểm tra số thuộc tính đó
ereport(ERROR, (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
errmsg("invalid input syntax for complex: \"%s\"", str)));

```

Và kiểu trả về của hàm đầu vào là kiểu Complex.

Theo dõi hàm đầu ra :

```

PG_FUNCTION_INFO_V1(complex_out);
Datum complex_out(PG_FUNCTION_ARGS)
{
    Complex *complex = (Complex *) PG_GETARG_POINTER(0);
    char *result;
    result = (char *) palloc(100);
    snprintf(result, 100, "(%g,%g)", complex->x, complex->y);
}

```

```
PG_RETURN_CSTRING(result);
}
```

Đối với hàm đầu ra, kiểu trả về của hàm là kiểu chuỗi, có nhiệm vụ chỉ ra cách hiển thị của kiểu dữ liệu đối với người dùng. Ví dụ, hiển thị của kiểu Complex với người dùng sẽ có dạng (a, b).

Tóm lại, việc tạo ra hàm đầu vào và đầu ra cho kiểu dữ liệu rất quan trọng trong việc hiển thị định dạng của chúng ra bên ngoài với người dùng. Nhờ vào chúng mà người dùng có thể dễ dàng hiểu được kiểu dữ liệu đó cần phải có các yếu tố nào và được hiển thị như thế nào...?

b. Sử dụng câu lệnh SQL để tạo hàm

Nếu như ở phần trước chúng ta có đề cập đến cách mở rộng hàm sử dụng ngôn ngữ truy vấn SQL và cú pháp lệnh được dùng là CREATE FUNCTION. Thì trong phần này, cú pháp lệnh đó cũng sẽ được dùng để tạo hàm. Sau khi biên dịch file .c thành dạng file .so, để hệ thống có thể nhận biết và nạp hàm thì cần thiết có bước tạo hàm theo cú pháp :

```
CREATE FUNCTION [tên_hàm] RETURNS [kiểu_dữ_liệu]
AS '[tên_file](thông thường sẽ là file có định dạng ".so")',
[tên_hàm](thường là trùng với tên hàm khai báo ở trước)
LANGUAGE C IMMUTABLE STRICT;
```

VD : tạo ra 2 hàm *complex_in* và *complex_out*

```
CREATE FUNCTION complex_in(cstring) RETURNS complex
AS 'complex.so', //tên file đã được biên dịch từ file
//complex.c
'complex_in' //tên hàm
LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION complex_out(complex) RETURNS cstring
AS 'complex.so', 'complex_out'
LANGUAGE C IMMUTABLE STRICT;
```

c. Sử dụng câu lệnh SQL tạo kiểu dữ liệu

Kiểu dữ liệu chỉ hoàn toàn được chấp nhận sau khi lệnh “*CREATE TYPE complex;*” hoàn toàn được hệ thống chấp nhận, có nghĩa là lệnh này thực thi trả về kết quả đúng.

```
Cấu trúc lệnh CREATE TYPE name (  
    INPUT= input_function,  
    OUTPUT=output_function,  
    INTERNALLENGTH= internallength);
```

Tất nhiên, tùy chọn lệnh CREATE TYPE còn rất nhiều như RECEIVE, SEND, ANALYZE... tuy nhiên, tùy chọn cần thiết cho cấu trúc lệnh là *input_function*, *output_function* và *internallength*. Các tùy chọn này cung cấp các thông tin cần thiết và tối thiểu về hàm đầu vào, hàm đầu ra và độ dài kiểu dữ liệu. Do đó, để tạo một kiểu dữ liệu thì không thể bỏ qua 3 tùy chọn cơ bản này.

Trong trường hợp tạo ra kiểu dữ liệu Complex, các tùy chọn được cung cấp đó là, hàm đầu vào : *complex_in*, hàm đầu ra: *complex_out*, độ dài kiểu dữ liệu : *16byte*, ta thực thi lệnh sau

```
CREATE TYPE complex (  
    internallength = 16,  
    input = complex_in,  
    output = complex_out  
);
```

Câu lệnh thực thi thành công thì kiểu dữ liệu có tên Complex sẽ được thêm vào hệ thống PostgreSQL với độ dài 16byte, kiểu dữ liệu đầu vào dạng số thực, và kiểu dữ liệu đầu ra là dạng chuỗi mô tả (a, b).

3.2.4. Toán tử do người dùng định nghĩa.

PostgreSQL cho phép người dùng tạo ra các toán tử tùy chọn được thêm vào các hàm tùy chọn. Đôi khi, toán tử được gọi là “*syntactic sugar*” cho hàm. Tại sao lại gọi như vậy? Về mặt kỹ thuật, toán tử chỉ là một cú pháp để thay thế cho một hàm hiện có. Do đó, trước khi tạo ra toán tử mới chúng ta cần tạo ra hàm trước.

Toán tử PostgreSQL được định nghĩa có thể bao gồm một vài lệnh tùy chọn, nó thông báo cho hệ thống biết các tùy chọn, hệ thống sẽ xử lý ứng với mỗi tùy chọn nhất định. Với mỗi tùy chọn sẽ có nhiệm vụ riêng trong cú pháp tạo toán tử mới. Sử dụng lệnh dưới đây để tạo ra một toán tử mới.

```
CREATE OPERATOR name (  
    PROCEDURE = functionname  
    [, LEFTARG = type1 ]  
    [, RIGHTARG = type2 ]  
    [, COMMUTATOR = commutatorop ] );
```

Để hiểu rõ hơn về tác dụng của các tùy chọn trong lệnh CREATE OPERATOR, chúng ta sẽ đi sâu vào tìm hiểu mục đích, tác dụng và giải thích các tùy chọn.

+ *Name* : là tên của toán tử mới sẽ được tạo ra. Toán tử có tên *Name* chỉ có thể bao gồm các ký tự được chấp nhận dưới đây :

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

+ *PROCEDURE=functionname* : khai báo tên của hàm sẽ được gọi trong toán tử mới. hàm này luôn luôn được tạo ra trước khi tạo toán tử mới. Có thể coi, toán tử mới chỉ là cú pháp để thay thế cho hàm này. Ví dụ, chúng ta tạo ra hàm *number_add()* với mục đích là cộng hai số, tất nhiên, bên trong hàm sẽ có mã mã nhằm thực hiện chức năng đó. Và khi muốn sử dụng hàm, chúng ta có thể gọi trực tiếp hàm, việc này có thể gây ra nhiều lỗi như nhập không đúng tham số truyền vào cả về số lượng và kiểu dữ liệu. Do đó, việc tạo toán tử + thay thế cho hàm *number_add()* dựa trên hàm *number_add()* sẽ giúp tránh được những sai sót khi sử dụng.

+ LEFTARG = *type*: chỉ ra kiểu dữ liệu của biến bên trái

+ RIGHTARG=type : chỉ ra kiểu dữ liệu của biến bên phải.

+ COMMUTATOR = ký tự của toán tử

VD: tạo ra toán tử “+” hai số thực, thay thế cho hàm *complex_add(complex, complex)* có chức năng cộng hai số thực.

```
PG_FUNCTION_INFO_V1 (complex_add) ;
```



```

Datum complex_add(PG_FUNCTION_ARGS)
{
    Complex *complex1=(Complex *)PG_GETARG_P(0);
    Complex *complex2=(Complex *)PG_GETARG_P(1);
    Complex*complex3=(Complex *)palloc (sizeof(Complex)) ;
        complex3->x=complex1->x + complex2->x ;
        complex3->y=complex1->y + complex2->y ;
        PG_RETURN_POINTER(complex3);
}

CREATE OPERATOR +(
    LEFTARG=Complex,
    RIGHTARG=Complex,
    PROCEDURE=complex_add,
    COMMUTATOR=+);

```

3.2.5. Hàm tập hợp cho người dùng định nghĩa

Như đã biết, hàm tập hợp trong PostgreSQL theo chuẩn SQL nghĩa là, nó có tác dụng xử lý tất cả các hàng có trong một cột dữ liệu, từ hàng đầu tiên đến hàng cuối cùng của cột đó. Ví dụ, với hàm tính tổng theo chuẩn SQL có tên là sum(), khi người dùng gọi hàm này, tương đương với việc người dùng muốn tính tổng các giá trị tất cả các hàng trong một cột cụ thể nào đó : SELECT sum (price) FROM Price;

Tương tự với các hàm tập hợp theo chuẩn SQL, hàm tập hợp do người định nghĩa mở rộng cho PostgreSQL cũng có nhiệm vụ tương tự là xử lý tất cả các hàng có trong một cột dữ liệu, từ hàng đầu tiên đến hàng cuối cùng. Để tạo hàm tập hợp mới, sử dụng cú pháp sau :

```

CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type

```

```
[ , INITCOND = initial_condition ]
)
```

+ *name* : tên của hàm tập hợp tạo ra.

+ *input_data_type* : kiểu dữ liệu của tham số khi nhập vào hàm.

+ *sfunc* : tên hàm được sử dụng để thực hiện mục đích của hàm tập hợp. Ví dụ, để tạo hàm tập hợp thực hiện chức năng tính tổng các dữ liệu kiểu point3d, thì cần có một hàm thực hiện việc tính tổng trên kiểu dữ liệu point3d.

+ *initial_condition* : thiết lập ban đầu cho giá trị trạng thái. Nghĩa là điều kiện ban đầu cho kiểu dữ liệu được nhập vào hàm.

VD : tạo hàm tập hợp để tính tổng các số phức

```
CREATE AGGREGATE sum_complex(Complex)
(
    sfunc = complex_add,
    Stype = Complex,
    Initcond = '(0,0)'
);
```

Sau khi hàm tập hợp được tạo, nó sẽ được sử dụng như những hàm tập hợp chuẩn. Tuy nhiên, với những hàm tập hợp chuẩn, nó có thể thực thi với mọi kiểu dữ liệu chuẩn, còn hàm tập hợp do người dùng định nghĩa chỉ thực thi khi người dùng truyền đúng kiểu dữ liệu cho biến khi sử dụng hàm.

3.3. Viết hàm mở rộng cho PostgreSQL

Bài toán 1 : Tạo kiểu dữ liệu điểm trong không gian 3 chiều có tên là point3d gồm các thuộc tính hoành độ (x), tung độ (y), cao độ (z).

Xây dựng toán tử, cộng điểm, trừ điểm. Xây dựng hàm tính khoảng cách giữa 2 điểm, so sánh giữa 2 điểm.

Xây dựng hàm tập hợp, tính tổng các điểm có trong bảng dữ liệu. Tính trọng tâm của các điểm có trong bảng dữ liệu.

Thực hiện :

Mục đích của bài toán là xây dựng kiểu dữ liệu điểm trong không gian 3 chiều, trên kiểu dữ liệu này, người dùng có thể thực hiện thao tác cộng, trừ, nhân, chia 2 điểm. Hơn nữa, người dùng có thể tìm được trọng tâm của các điểm trong không gian 3 chiều.

- Tạo kiểu dữ liệu

```
typedef struct point3d {  
    float4      x;  
    float4      y;  
    float4      z;  
} point3d;
```

- Tạo hàm nhập và hàm xuất

```
PG_FUNCTION_INFO_V1(point3d_in_test); //hàm nhập  
Datum point3d_in_test(PG_FUNCTION_ARGS)  
{  
    char *str = PG_GETARG_CSTRING(0);  
    float4 x,y,z;  
    point3d *result;  
    if (sscanf(str, " ( %f , %f, %f )", &x, &y,&z) != 3)  
        ereport(ERROR,  
                (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),  
                 errmsg("invalid input syntax for complex:  
\"%s\"",str)));\br/>    result = (point3d *) palloc(sizeof(point3d));  
    result->x = x;  
    result->y = y;  
    result->z = z;
```

```

    PG_RETURN_POINTER(result);
}

PG_FUNCTION_INFO_V1(point3d_out_test); //hàm xuất
Datum point3d_out_test(PG_FUNCTION_ARGS)
{
    point3d *point3D = (point3d *)
    PG_GETARG_POINTER(0);
    Char *result;
    result = (char *) palloc(100);
    snprintf(result, 100, "(%g %g %g)", point3D->x,
point3D->y, point3D->z);
    PG_RETURN_CSTRING(result);
}

```

- Tạo hàm bằng truy vấn SQL

```

CREATE FUNCTION point3d_out_test(point3d) RETURNScstring
AS 'point3d_test.so','point3d_out_test' LANGUAGE C STRICT
IMMUTABLE;

```

```

CREATE FUNCTION point3d_in_test(cstring) RETURNS point3d
AS 'point3d_test.so','point3d_in_test' LANGUAGE C STRICT
IMMUTABLE;

```

➔ kiểu dữ liệu point3d :

```

CREATE TYPE
point3d(internallength=16,input=point3d_in_test,output=poi
nt3d_out_test);

```

Như vậy, kiểu dữ liệu điểm trong không gian gọi là point3d có định dạng đầu vào thông qua hàm *point3d_in_test*, định dạng đầu ra thông qua hàm *point3d_out_test*; Sau khi đã có kiểu dữ liệu point3d, thực hiện công, trừ, tính khoảng cách giữa 2 điểm...

- Cộng 2 điểm

```
PG_FUNCTION_INFO_V1(point3d_add);
Datum point3d_add(PG_FUNCTION_ARGS)
{
    point3d *point3d1=(point3d *)PG_GETARG_POINTER(0);
    point3d *point3d2=(point3d *)PG_GETARG_POINTER(1);
    point3d *point3d3=(point3d *)palloc(sizeof(point3d));
    point3d3->x=point3d1->x+point3d2->x;
    point3d3->y=point3d1->y+point3d2->y;
    point3d3->z=point3d1->z+point3d2->z;
    PG_RETURN_POINTER(point3d3);
}

CREATE FUNCTION point3d_add(point3d, point3d) RETURNS
point3d AS 'point3d_test.so', 'point3d_add' LANGUAGE C
STRICT IMMUTABLE;

CREATE OPERATOR + (leftarg=point3d, rightharg=point3d,
procedure=point3d_add, commutator=+);
```

- Trừ 2 điểm

```
PG_FUNCTION_INFO_V1(point3d_minus);
Datum point3d_minus(PG_FUNCTION_ARGS)
{
    point3d *point3d1=(point3d *)PG_GETARG_POINTER(0);
    point3d *point3d2=(point3d *)PG_GETARG_POINTER(1);
    point3d *point3d3=(point3d *)palloc(sizeof(point3d));
    point3d3->x=point3d1->x-point3d2->x;
    point3d3->y=point3d1->y-point3d2->y;
```

```

    point3d3->z=point3d1->z-point3d2->z;

    PG_RETURN_POINTER(point3d3);
}

CREATE FUNCTION point3d_minus(point3d,point3d) RETURNS
point3d AS 'point3d_test.so', 'point3d_minus' LANGUAGE C
STRICT IMMUTABLE;

CREATE OPERATOR - (leftarg=point3d, rightarg=point3d,
procedure=point3d_minus, commutator=-);

- Tính khoảng cách giữa 2 điểm

PG_FUNCTION_INFO_V1(distance_point3d);
Datum distance_point3d(PG_FUNCTION_ARGS)
{
    point3d *point3d1=(point3d *)PG_GETARG_POINTER(0);
    point3d *point3d2=(point3d *)PG_GETARG_POINTER(1);
    float4 result;

    float4 tmp1=(point3d1->x - point3d2->x)*(point3d1->x -
point3d2->x);

    float4 tmp2=(point3d1->y - point3d2->y)*(point3d1->y -
point3d2->y);

    float4 tmp3=(point3d1->z - point3d2->z)*(point3d1->z -
point3d2->z);

    result=sqrt(tmp1+tmp2+tmp3);

    PG_RETURN_FLOAT4(result);
}

CREATE FUNCTION distance_point3d(point3d,point3d) RETURNS
float AS 'point3d_test.so', 'point3d_add' LANGUAGE C
STRICT IMMUTABLE;

```

- Hàm tính Point3d * float

```
PG_FUNCTION_INFO_V1(point3d_multi_float);
Datum point3d_multi_float(PG_FUNCTION_ARGS)
{
    point3d *point3d1=(point3d *)PG_GETARG_POINTER(0);
    float8 m=PG_GETARG_FLOAT8(1);
    point3d *point3d2=(point3d *)palloc(sizeof(point3d));
    point3d2->x=m*point3d1->x;
    point3d2->y=m*point3d1->y;
    point3d2->z=m*point3d1->z;
    PG_RETURN_POINTER(point3d2);
}

CREATE FUNCTION point3d_multi_float(point3d,float) RETURN
point3d AS 'point3d_test.so', 'point3d_multi_float'
LANGUAGE C STRICT IMMUTABLE;

CREATE OPERATOR * (leftarg = point3d, rightarg=float,
procedure = point3d_multi_float, commutator = * );
```

- Hàm tính Point3d / float

```
PG_FUNCTION_INFO_V1(point3d_multi_float);
Datum point3d_multi_float(PG_FUNCTION_ARGS)
{
    point3d *point3d1=(point3d *)PG_GETARG_POINTER(0);
    float8 m=PG_GETARG_FLOAT8(1);
    point3d *point3d2=(point3d *)palloc(sizeof(point3d));
    point3d2->x=point3d1->x/m;
    point3d2->y=point3d1->y/m;
```

```

point3d2->z=point3d1->z/m;

PG_RETURN_POINTER(point3d2);
}

CREATE FUNCTION point3d_div_float(point3d,float) RETURN
point3d AS 'point3d_test.so', 'point3d_div_float'
LANGUAGE C STRICT IMMUTABLE;

CREATE OPERATOR / (leftarg =point3d, rightarg = float,
procedure=point3d_div_float, commutator = / );

```

- Tạo hàm tập hợp tính tổng các điểm

```

CREATE AGGREGATE sum_point3d(point3d)(sfunc=point3d_add, stype=point3d,
initcond = '(0,0,0)');

```

- Trọng tâm của các điểm được tính theo công thức

$$G = \sum (p_i * m_i) / \sum m_i$$

Với bảng dữ liệu mypoint3d (id int, a point3d, m float); thì tính trọng tâm của các điểm có trong cột a theo công thức :

```

SELECT sum_point3d(a * m) / sum (m) as trong_tam FROM
mypoint3d;

```

Bài toán 2 : Tạo kiểu dữ liệu mô tả hình cầu trong không gian (*sphere*) có các thuộc tính tâm I (hoành độ x, tung độ y, cao độ z) và bán kính r. Xây dựng hàm tính thể tích hình cầu.

Thực hiện : PostGIS chỉ hỗ trợ kiểu dữ liệu hình học bao gồm POINT, LINESTRING, POLYGON, nhưng chưa thấy xuất hiện kiểu hình cầu. Do vậy mục đích của bài toán là tạo ra kiểu dữ liệu hình cầu và xây dựng hàm tính thể tích hình cầu.

- Tạo kiểu dữ liệu sphere bao gồm toạ độ của tâm hình cầu I(x,y,z) và bán kính hình cầu r được biểu diễn :

```

Typedef struct sphere {
    float4 x,
    float4 y,

```



```

float4 z,
float4 r

};

```

- Tương tự, tạo hàm nhập và hàm xuất dữ liệu có tên *sphere_in()* và *sphere_out()*
- Kiểu dữ liệu hình cầu

```

CREATE TYPE sphere (internallength = 16, input =
sphere_in, output = sphere_out);

```

- Tạo hàm tính thể tích hình cầu :

```

PG_FUNCTION_INFO_V1(sphere_area);
Datum sphere_area(PG_FUNCTION_ARGS)
{
    float8 tmp,result;
    sphere *mysphere=(sphere *)PG_GETARG_POINTER(0);
    tmp=(mysphere->r) * (mysphere->r) * (mysphere->r);
    result=4.0/3.0 * M_PI * tmp;
    PG_RETURN_FLOAT8(result);
}

CREATE FUNCTION sphere_area(sphere) returns float as
'sphere.so', 'sphere_area' LANGUAGE C STRICT IMMUTABLE;

```

Như vậy, kiểu dữ liệu hình cầu trong không gian gọi là sphere đã được tạo, có định dạng đầu vào thông qua hàm *sphere_in*, định dạng đầu ra thông qua hàm *sphere_out*; từ đó, để tính thể tích hình cầu rất đơn giản, tính theo công thức :

$\text{Thể tích} = \frac{4}{3} * M_PI * r^3$;

TỔNG KẾT

Sau một thời gian nghiên cứu và tìm hiểu, khóa luận đã thu được các kết quả như sau:

- Cài đặt, thao tác thành thạo với hệ quản trị CSDL PostgreSQL thông qua các kiểu giao tương tác. Ngoài ra, nắm rõ được lịch sử phát triển và những ưu điểm của nó so với các hệ quản trị khác.
- Đối với PostGIS – mô đun mở rộng cho PostgreSQL. Tôi đã trình bày những kiến thức như cách tạo CSDL không gian, cách xử lý với dữ liệu không gian...đặc biệt là nắm được tác dụng và cách sử dụng các hàm hỗ trợ của PostGIS và đã sử dụng thử nghiệm số lượng lớn các hàm đó. Từ đó, áp dụng các hàm đó vào truy vấn không gian trong bảng không gian.
- Với phần mở rộng trong PostgreSQL, tôi đã trình bày khá chi tiết về cách mở rộng trong PostgreSQL. Từ đó nắm rõ được cách viết mở rộng, sử dụng các mở rộng và áp dụng nó vào một số bài toán cụ thể.
- Phần thực nghiệm của KLTN đã định nghĩa một số kiểu dữ liệu không gian mở rộng và viết các hàm truy vấn.....

Trong khóa luận này, tôi hy vọng đã đưa ra những kiến thức cần thiết nhất về hệ quản trị CSDL PostgreSQL và PostGIS – mô đun mở rộng của PostgreSQL. Với kiến thức về PostGIS hỗ trợ truy vấn trong CSDL PostgreSQL, tôi mong rằng chúng sẽ được áp dụng một cách thiết thực vào đời sống thực tế.

TÀI LIỆU THAM KHẢO

Sách:

- [1] Ewald Geschwinde and Hans-Juergen Schoening, PHP and PostgreSQL Advanced Web Programming, 2002, Sams Publishing.
- [2] Korry Dougla and Susan Douglas, The comprehensive guide to building programming and administering PostgreSQL database, 2nd, 2005, Sams Publishing.
- [3] Paul Ramsey, PostGIS Workshop, Refrations Research, Suite 300 – 1207 Douglas Street, Victoria – British Columbia, CANADA – V8W 2E7
- [4] PostGIS 1.5.0 Manual
- [5] Ralf Hartmut Gueting, An introduction to databases system, Praktische Informatik IV, FenUniversity Hagen, Germany
- [6] The PostgreSQL Global Development Group, PostgreSQL 8.4 Documentation, 1996-2009.
- [7] W.Jason Gilmore and Robert H.Treat, Beginning PHP and PostgreSQL 8: From Novice to Professional, Feb 2006, Kinetic Publishing Service.

Web

- [1] www.postgis.refrations.net
- [2] www.postgresql.org
- [3] www.vi.wikipedia.org/wiki/So_sánh_các_hệ_quản_trị_cơ_sở_dữ_liệu_quan_hệ
- [4] www.vi.wikipedia.org/wiki/Shapefile

