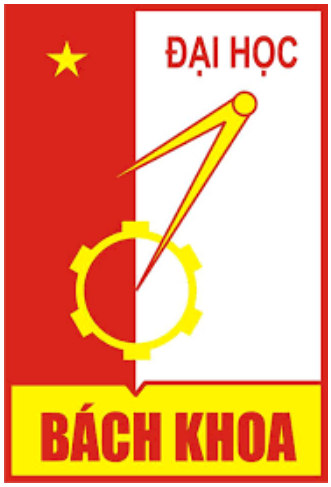
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CNTT & TT



BÁO CÁO BÀI TẬP LỚN MÔN
HỆ THỐNG THÔNG TIN ĐỊA LÝ

Đề tài
BẢN ĐỒ HỆ THỐNG BUS HÀ NỘI

Giảng viên hướng dẫn: TS. Vũ Tuyết Trinh

Bùi Minh Hiếu	MSSV: 20173114
Nguyễn Quốc Chiến	MSSV: 20172975

Hà Nội, 09/2021

Mục lục

1. Chương I: Giới thiệu đề tài

Hệ thống giao thông công cộng nói chung và hệ thống xe bus nói riêng có vai trò quan trọng trong sự phát triển văn minh đô thị. Việc xây dựng cơ sở dữ liệu thống nhất quản lý các hệ thống trên rất cần thiết trong việc phổ biến ứng dụng của các hệ thống đó với mọi người, đồng thời hỗ trợ việc nâng cấp các hệ thống này trong tương lai.

Trong phạm vi môn học Hệ thống thông tin địa lý, chúng em xây dựng một cơ sở dữ liệu thống nhất phục vụ cho việc quản lý hệ thống bus của thành phố Hà Nội. Từ đó, làm nền tảng cho việc xây dựng các ứng dụng hỗ trợ người dùng sử dụng các hệ thống này trong tương lai.

2. Chương II: Cơ sở lý thuyết

2.1. Hệ cơ sở dữ liệu địa lý

2.1.1. Giới thiệu

Dữ liệu địa lý nhiều hơn so với hình ảnh điện tử. Dữ liệu địa lý không chỉ mô tả các đối tượng thật và các quan hệ trong không gian mà còn thể hiện tham chiếu không gian , hình học và các thông tin chuyên đề. Tuy nhiên để khai thác một cách đầy đủ các đặc trưng đó, cần có một công cụ bổ sung. Một hệ thống thông tin địa lý là một hệ thống thông tin được thiết kế để tương thích, lưu trữ, chỉnh sửa, phân tích, chia sẻ và hiển thị thông tin địa lý cho các nhà xây dựng, thiết kế. Ứng dụng GIS là những công cụ cho phép người sử dụng tạo ra các truy vấn tương tác, phân tích thông tin không gian, chỉnh sửa dữ liệu trong bản đồ và hiển thị kết quả của tất cả các hoạt động đó.

Một hệ thống thông tin địa lý (HTTTĐL) là sự kết hợp của bản đồ, phân tích, thống kê và công nghệ cơ sở dữ liệu. Đó là thiết kế có thể tùy chỉnh cho một tổ chức. Một HTTTĐL được phát triển với mục đích thẩm quyền hay các doanh nghiệp có thể không nhất thiết phải tương thích hoặc tương thích với hệ thống GIS đã được xây dựng cho các ứng dụng khác.

2.1.2. Các thành phần của GIS

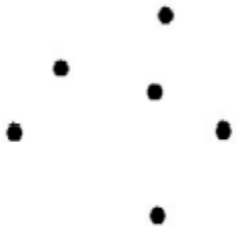
Tất cả GIS cung cấp các cách xử lý cho việc hiển thị và quản lý thông tin địa lý dựa trên bốn yếu tố cơ bản:

- Đối tượng
- Thuộc tính

2.1.3. Đối tượng

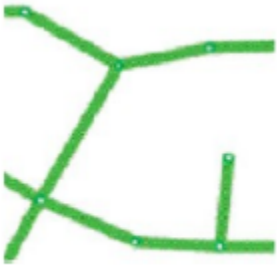
Một đối tượng được sử dụng để mô tả một thực thể trong không gian – thời gian. Bên cạnh một số đối tượng được thêm vào, các đối tượng địa lý cơ bản là tập hợp các điểm, đường và vùng. Chúng thể hiện những điều xảy ra tự nhiên, đối tượng rời rạc (như sông, thảm thực vật) đối với các công trình xây dựng (như đường sá, đường ống, giếng, các tòa nhà…) hay các khu đất (quận, khu vực chính trị và các mảnh đất…)

Điểm: xác định những vật rời rạc mà chúng quá nhỏ để mô tả theo kiểu đường hay vùng như địa điểm, bốt điện thoại, và các điểm máy đo dòng chảy… Điểm cũng có thể đại diện cho vị trí địa điểm nào đó, tọa độ GPS hay đỉnh núi.



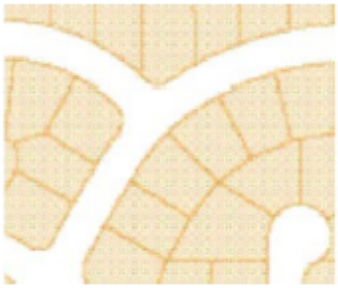
Hình 1.1: Đối tượng điểm

Đường: đường thể hiện hình dạng của đối tượng địa lý quá hẹp để mô tả theo kiểu một vùng (như đường phố, dòng chảy). Đường cũng được sử dụng thể hiện các đối tượng có chiều dài nhưng không phải vùng như đường đồng mức và địa giới hành chính.



Hình 1.2: đối tượng đường

Vùng: thể hiện hình dạng của các đối tượng lớn như các tiểu bang, quận hạt, loại đất, hiện trạng sử dụng đất…



Hình 1.3: Đối tượng vùng thể hiện các thửa

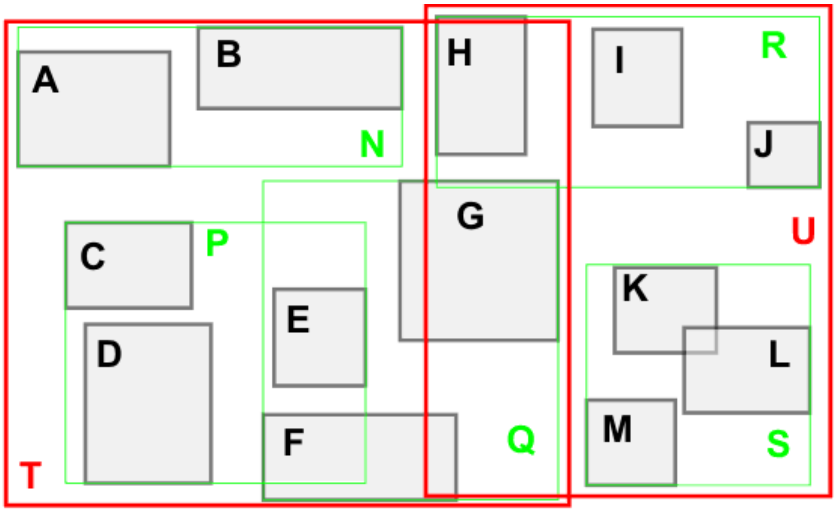
2.1.4. Thuộc tính

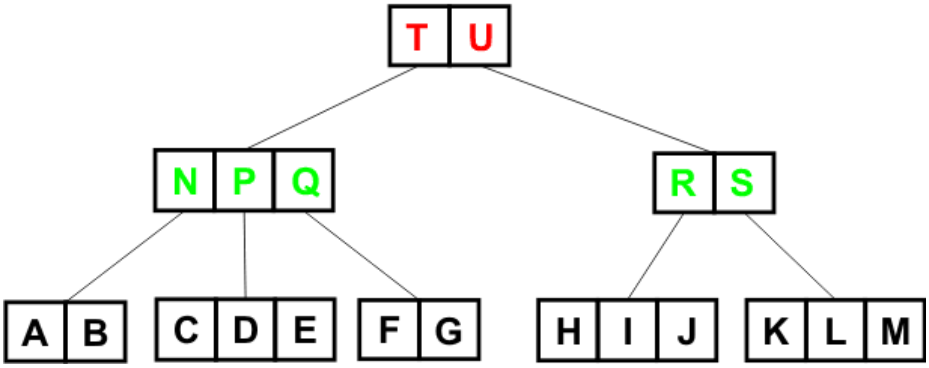
Dữ liệu địa lý truyền đạt các thông tin cần mô tả thông qua các kí hiệu, màu sắc, và các nhãn. Ví dụ:

- Đường sá được hiển thị dựa trên lớp đường. Ví dụ, kí hiệu dạng đường biểu diễn đường cao tốc được phân chia, đường chính, đường giao thông không lát đá, đường mòn…..
- Màu xanh dùng để thể hiện nước, dòng chảy.
- Đường phố thường được gán nhãn với tên của chúng và thường có thêm một số địa chỉ nào đó
- Các điểm đặc biệt và các kí hiệu dạng đường biểu thị các đối tượng cụ thể như các tuyến đường sắt, sân bay, trường học, bệnh viện …

2.1.5. R-tree

R-Tree thường được sử dụng để lập chỉ mục cho một đối tượng không có kích thước trong không gian nhiều chiều giống như tọa độ địa lý, hình chữ nhật hoặc đa giác. R tree được đề xuất bởi Antonin Guttman vào năm 1984 [8] và được sử dụng rộng rãi trong cả lý thuyết và ứng dụng thực tiễn. Cấu trúc chỉ mục này có thể giúp cập nhật đơn giản cho những chỉ mục điểm trong không gian nhiều chiều với một vài cải tiến nhỏ trong giải thuật chèn và tìm kiếm. Một thực tế hay sử dụng của R tree là có thể lưu trữ các đối tượng không gian như địa điểm, nhà hàng, hoặc những đa giác chúng tạo thành bản đồ: đường, nhà, hồ, bờ biển, vv… và có thể tìm kiếm 8 chúng một cách nhanh chóng với các truy vấn như “tìm tất cả các bảo tàng trong vòng bán kính 2 km từ vị trí hiện tại của tôi”.





2.2. Thuật toán tìm đường đi ngắn nhất

Trong lý thuyết đồ thị, bài toán đường đi ngắn nhất nguồn đơn là bài toán tìm một đường đi giữa hai đỉnh sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất. Bài toán được mô tả như sau: cho trước một đồ thị có trọng số (một tập đỉnh V, một tập cạnh E, và một hàm trong số có giá trị thực f : E → R), cho trước một đỉnh v thuộc V, tìm một đường đi P từ v tới mỗi đỉnh v' thuộc V sao cho:

$$\sum_{p \in P} f(p)$$

là nhỏ nhất trong tất cả các đường nối từ v tới v'

2.2.1. Thuật toán Dijkstra

Thuật toán Dijkstra, mang tên của nhà khoa học máy tính người Hà Lan Edsger Dijkstra, là một thuật toán giải quyết bài toán đường đi ngắn nhất từ một đỉnh đến các

đỉnh còn lại của đồ thị có hướng, trọng số không âm. Thuật toán thường được sử dụng trong định tuyến với một chương trình con trong các thuật toán đồ thị hay trong công nghệ Hệ thống định vị toàn cầu GPS.

Mã giả của thuật toán Dijkstra như sau:

```
procedure uniform_cost_search(Graph, start, goal) is
    node ← start
    cost ← 0
    frontier ← priority queue containing node only
    explored ← empty set
    do
        if frontier is empty then
            return failure
        node ← frontier.pop()
        if node is goal then
            return solution
        explored.add(node)
        for each of node's neighbors n do
            if n is not in explored then
                frontier.add(n)
```

2.2.2. Thuật toán A*

Thuật toán A* là thuật toán cải tiến của với thuật toán Dijkstra, trong trường hợp ta có một hàm heuristic ước lượng độ dài đường đi từ một nút bất kỳ đến đích.

So với thuật toán Dijkstra, frontier không được duyệt tuần tự mà ưu tiên đỉnh n có hàm f(n) = g(n) + h(n) nhỏ nhất, trong đó: g(n) là chi phí từ nút gốc cho đến nút hiện tại n; h(n) (hàm heuristic) là chi phí ước lượng từ nút hiện tại n tới đích.

Đối với bài toán tìm đường đi ngắn nhất giữa hai điểm trên dữ liệu bản đồ, ta luôn có một heuristic đó là khoảng cách Euclide giữa hai điểm đó.

2.3. Tổng quan về PostgreSQL và PostGIS

2.3.1. PostgreSQL

PostgreSQL là một hệ thống cơ sở dữ liệu quan hệ đối tượng mã nguồn mở mạnh mẽ, tích hợp SQL với nhiều tính năng giúp lưu trữ và chia tỷ lệ một cách an toàn các khối lượng công việc dữ liệu phức tạp nhất. Nguồn gốc của PostgreSQL có từ năm 1986 như một phần của dự án POSTGRES tại Đại học California ở Berkeley.

PostgreSQL được chúng em sử dụng trong bài tập này vì phần mở rộng PostGIS rất mạnh mẽ và phù hợp cho việc lưu trữ và quản lý dữ liệu địa lý.

2.3.2. PostGIS

PostGIS là 1 phần mở rộng của hệ quản trị CSDL PostgreSQL được cung cấp miễn phí cho phép CSDL quản lý các đối tượng GIS. PostGIS cho phép quản trị CSDL

không gian dùng trong hệ thống thông tin địa lý (GIS). PostGIS tuân theo chuẩn OpenGIS, cho phép chạy các truy vấn vị trí trong SQL.

2.3.3. GIST và GIN

GiST (Generalized Search Tree) cho phép kết hợp B-tree, R-tree, và các kiểu đánh chỉ mục người dùng tự định nghĩa để tạo các chỉ mục tùy chỉnh với khả năng truy vấn tiên tiến. GiST đã được sử dụng trong PostGIS (nó đã được chúng tôi thực hiện chuẩn hóa với tất cả các triển khai PostgreSQL từ tháng Giêng), và OpenFTS (một cỗ máy tìm kiếm full text mã nguồn mở). PostgreSQL cũng hỗ trợ SP-GiST cho phép tạo phân vùng chỉ mục tìm kiếm để tăng tốc độ truy xuất.

GIN (Generalized Inverted Index) cho phép đánh chỉ mục các kiểu dữ liệu kết hợp. Các kiểu dữ liệu kết hợp cho phép bạn kết hợp các kiểu dữ liệu khác nhau theo nhiều cách để tạo ra một cái gì đó hoàn toàn tùy chỉnh.

Để tạo các chỉ mục GIST và GIN, ta dùng cú pháp là: CREATE INDEX... ON... USING GIST|GIN...

Trong PostgreSQL 9.5 (hiện tại đang là bản beta), BRIN (Block Range Index) sẽ được giới thiệu. BRIN cho phép chia các bảng lớn thành các khoảng dựa trên cột đã được đánh chỉ muc. Điều này có nghĩa là các kế hoạch truy vấn có thể quét trong khoảng đã được chỉ định bởi truy vấn. Vì thế, nhờ các khoảng chỉ mục, số lượng kích thước đĩa cần thiết cho các chỉ mục sẽ nhỏ hơn một chỉ mục chuẩn B-Tree.

3. Chương III: Triển khai

3.1. Thu thập và chuẩn bị dữ liệu

Quá trình thu thập và chuẩn bị dữ liệu bao gồm các bước sau:

- Import vào cơ sở dữ liệu
- Xử lý dữ liệu đã import

Các phần sau trong mục 3.1 sẽ trình bày khái quát các công việc trong 2 bước trên.

3.1.1. Import vào cơ sở dữ liệu

Dữ liệu địa lý được khai thác từ https://www.openstreetmap.org/, đây là một dịch vụ bản đồ thế giới trực tuyến có nội dung mở. OpenStreetMap nhằm mục đích cung cấp dữ liệu địa lý do nhiều người dùng cùng cộng tác với nhau trên hệ thống wiki. Dữ liệu từ OpenStreetMap sau khi export có dạng file đuôi osm. Để thực hiện các nghiệp vụ quản lý hệ thống bus Hà Nội, ta cần export bản đồ toàn bộ thành phố Hà Nội và các vùng lân cận.

Sau khi thu được file osm từ OpenStreetMap, ta tiến hành việc import dữ liệu vào cơ sở dữ liệu PostgreSQL đã tạo bằng công cụ osm2pgsql. Dữ liệu sau khi import bao gồm các bảng chính như sau:

- planet_osm_point: Lưu thông tin các point.
- planet_osm_line: Lưu thông tin các line và multiline.
- planet_osm_polygon: Lưu thông tin các polygon.

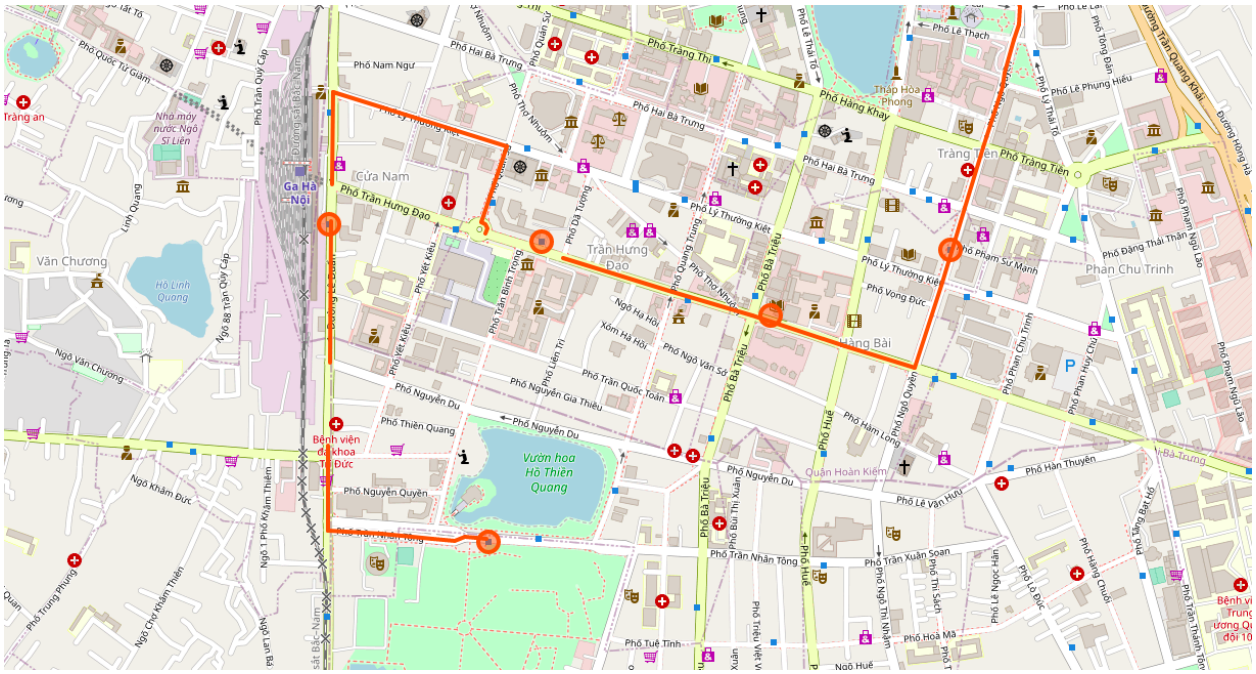
Dữ liệu ta cần quan tâm đến đó là thông tin các tuyến bus (tên, số hiệu, lộ trình, ...) và thông tin các điểm bus (tên, vị trí, ...) nằm lần lượt trong bảng planet_osm_line và planet_osm_point, ta có thể lấy các dữ liệu trên với truy vấn như sau:

```
-- thông tin tuyến bus
SELECT * FROM planet_osm_point WHERE highway = 'bus_stop';

-- thông tin điểm bus
SELECT * FROM planet_osm_line WHERE route = 'bus';
```

Sau các bước trên, ta đã có dữ liệu địa lý của các đối tượng cần quan tâm (tuyến bus, điểm bus) và các đối tượng địa lý khác nằm trong bản đồ. Tuy nhiên, việc thu thập và import dữ liệu như trên có một số vấn đề, đó là:

- Cấu trúc bảng như trên không thể hiện được quan hệ giữa điểm bus và tuyến bus.
- Dữ liệu về lộ trình của tuyến bus chưa chính xác, đôi khi bị chia thành các đoạn nhỏ, các đoạn này sau khi import vào database sẽ nằm ở các bản ghi khác nhau. Điều này dẫn đến việc tồn tại nhiều bản ghi trong bảng planet_osm_line có id giống nhau, cùng chỉ một tuyến bus, đồng thời không bản ghi nào thể hiện đúng toàn bộ lộ trình của tuyến bus.



3.1.2. Xử lý dữ liệu đã import

Để dữ liệu có thể sử dụng đối với vấn đề đặt ra ban đầu, ta cần giải quyết các vấn đề liên quan đến import dữ liệu kể trên, đồng thời đưa dữ liệu trong database về dạng phù hợp và tiện lợi cho việc truy vấn và áp dụng các thuật toán tìm đường.

Trước tiên, để có thông tin về quan hệ giữa các tuyến bus và điểm bus, giải pháp là sử dụng API của OpenStreetMap để lấy lần lượt thông tin các điểm bus mà một tuyến bus đi qua. Cụ thể, ta sử dụng API: https://www.openstreetmap.org/api/0.6/relation/{bus_id}. Dữ liệu sau khi lấy về được đưa vào bảng bus_stop, gồm 2 trường: bus_id và stop_id.

Bước tiếp theo, để thuận tiện cho việc truy vấn dữ liệu, ta tạo các bảng bus_route và bảng stop lưu trữ thông tin các tuyến bus và điểm bus.

- Đối với bảng stop, ta lấy trong bảng planet_osm_point các đối tượng với điều kiện highway = 'bus_stop' đã nêu ở trên, sau đó insert vào bảng stop:

```
WITH stopbus(osm_id, name, tags, way)
AS (
        SELECT osm_id, name, tags, way
        FROM planet_osm_point WHERE highway = 'bus_stop'
)
INSERT INTO stop
SELECT osm_id, name, tags, way
FROM stopbus
```

```
ORDER BY osm_id;
```

- Đối với bảng bus_route, ta làm tương tự như trên, tuy vậy, cần ghép trường way (lưu dữ liệu địa lý) của các bản ghi có osm_id giống nhau bằng hàm ST_LineMerge thành một multiline duy nhất.

```
WITH busroute(osm_id, operator, ref, name, tags, way)
AS (
	SELECT osm_id, operator, ref, name, tags,
		ST_makeline(ST_LineMerge(ST_Collect(way)))
		AS way
	FROM planet_osm_line
	GROUP BY osm_id, operator, ref, name, tags
	WHERE route = 'bus';
)
INSERT INTO bus_merged_route(osm_id, operator, ref, name, tags, way)
SELECT osm_id, operator, ref, name, tags, way
FROM busroute;
```

Dữ liệu ta thu được sau các bước ở trên đã có đầy đủ các thông tin cần thiết của hệ thống bus. Tuy vậy, để áp dụng các thuật toán tìm đường, dữ liệu cần được biểu diễn bằng đồ thị dưới dạng danh sách cạnh, tức là bằng một bảng edge_table. Khi một tuyến bus cho phép ta đi từ điểm dừng A đến điểm dừng B, ta thêm vào đồ thị một cạnh một chiều có điểm đầu là A, điểm cuối là B. Bảng edge_table được tạo như sau:

```
WITH busstop(source, target, cost, bus_id, change_bus_cost, distance_cost, way, x1,
y1, x2, y2)
AS (
	SELECT
		bs1.stop_id as source,
		bs2.stop_id as target,
		bs1.bus_id as bus_id,
		1 as change_bus_cost,
		ST_Distance(stop1.way, stop2.way) as distance_cost,
		ST_LineSubstring(bus_route.way, ST_LineLocatePoint(bus_route.way,
stop1.way), ST_LineLocatePoint(bus_route.way, stop2.way)) as way,
		ST_X(stop1.way) as x1,
		ST_Y(stop1.way) as y1,
		ST_X(stop2.way) as x2,
		ST_Y(stop2.way) as y2

	FROM bus_stop_v2 as bs1
	JOIN stop as stop1
```

```
            ON bs1.stop_id = stop1.osm_id
        JOIN bus_merged_route as bus_route
            ON bs1.bus_id = bus_route.osm_id
            AND bus_route.osm_id NOT IN (
                SELECT osm_id FROM planet_osm_line
                GROUP BY osm_id HAVING COUNT(*) > 1
                WHERE route = 'bus'
            )
        JOIN bus_stop_v2 as bs2
            ON bs1.bus_id = bs2.bus_id
        JOIN stop as stop2
            ON bs2.stop_id = stop2.osm_id
            AND ST_LineLocatePoint(bus_route.way, stop1.way) <
ST_LineLocatePoint(bus_route.way, stop2.way)
)
INSERT INTO edge_table_v2(source, target, cost, bus_id, change_bus_cost,
distance_cost, way, x1, y1, x2, y2)
SELECT source, target, cost, bus_id, change_bus_cost, distance_cost, way, x1, y1, x2,
y2
FROM busstop;
```

Trong bước xử lý dữ liệu ở trên, đối với các tuyến bus bị chia nhỏ thành nhiều lộ trình không liên tiếp, câu lệnh ST_LineLocatePoint sẽ báo lỗi. Đây là vấn đề liên quan đến dữ liệu đầu vào, vì vậy, trong phạm vi báo cáo này, chúng em sẽ bỏ qua các tuyến bus như vậy.

3.2. Bài toán 1: Tìm điểm bus lân cận một điểm

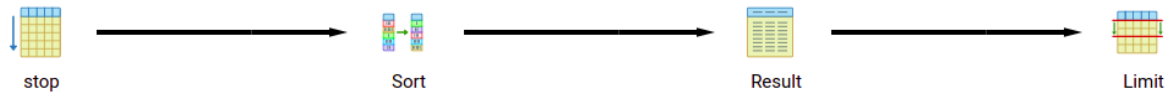
Để xác định k điểm bus gần nhất so với một tọa độ cho trước chúng ta sử dụng hàm ST_DISTANCE để xác định khoảng cách giữa điểm hiện tại tới lần lượt các điểm bus rồi lấy ra k điểm gần nhất.

Ví dụ với tọa độ (Nhà C1, Đại học Bách khoa Hà Nội) có tạo độ như sau: '0101000020110F0000D3B15F901F79664188E511C711414241':

```
SELECT
    name,
    ST_Transform(way, 4326),
    ST_DISTANCE(way,
'0101000020110F0000D3B15F901F79664188E511C711414241') as dist
FROM stop
ORDER BY dist
```

```
LIMIT 5;
```

Cách làm trên cho ta biết chính xác k điểm bus gần nhất với một tọa độ cho trước. Tuy nhiên, việc tính khoảng cách từ tọa độ trên tới tất cả các điểm bus có chi phí cao về mặt tính toán.



Trên thực tế, với các tình huống yêu cầu thời gian tính toán nhanh, như phục vụ cho việc tìm đường, việc duyệt qua tất cả các điểm bus là không hợp lý. Tuy nhiên, các bài toán này không quan tâm đến việc tìm chính xác k điểm bus, mà quan tâm đến việc tìm các điểm bus cách tọa độ cho trước một khoảng chấp nhận được (để có thể đi bộ tới vị trí đó - cỡ 300m). Để giải quyết tình huống này, ta có thể tìm điểm bus nằm trong một hình tròn bán kính 300m quanh tọa độ đã cho.

```
SELECT name, way
FROM stop
WHERE ST_WITHIN(way,
ST_Buffer('0101000020110F0000D3B15F901F79664188E511C711414241', 300))
```

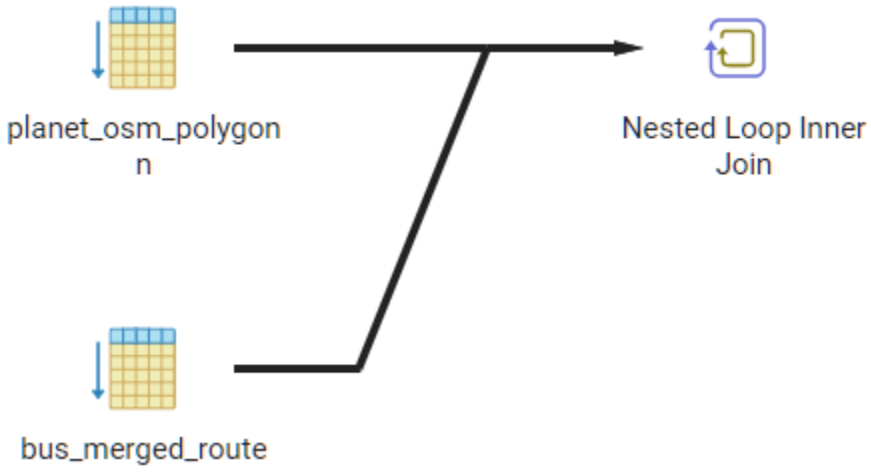
Với cách làm này, ta có thể tận dụng việc đánh index cho trường dữ liệu địa lý của của stop để giảm chi phí tính toán.

3.3. Bài toán 2: Tìm các tuyến bus đi qua một khu vực

Để xác định các tuyến bus đi qua một khu vực ta sử dụng hàm ST_CROSSES. Ví dụ xác định các tuyến bus đi qua quận Hà Đông (có osm_id = -9424587):

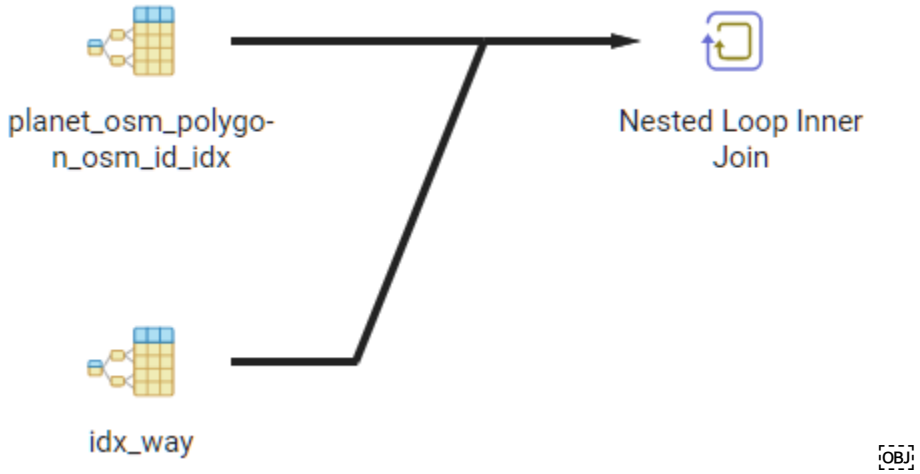
```
SELECT bus_merged_route.name, bus_merged_route.ref, bus_merged_route.operator
FROM bus_merged_route, planet_osm_polygon
WHERE ST_Crosses(bus_merged_route.way,planet_osm_polygon.way)
AND planet_osm_polygon.osm_id = '-9424587'
```

Do không sử dụng index nên phải duyệt qua từng bản ghi trong các bảng dẫn đến tốc độ query không nhanh.




```
Successfully run. Total query runtime: 559 msec.
34 rows affected.
```

Khi ta sử dụng index thì sẽ không còn cần phải duyệt qua từng bản ghi trong các bảng nữa giúp tăng tốc độ query.




```
Successfully run. Total query runtime: 380 msec.
34 rows affected.
```

3.4. Bài toán 3: Tìm đường đi tối ưu giữa hai điểm bus

Bài toán tìm đường đi tối ưu giữa hai điểm bus, dựa trên bài toán đường đi ngắn nhất trên đồ thị, là bài toán cơ bản đối với hệ thống bus. Kết hợp bài toán này cùng bài toán 1, ta có thể đề xuất đường đi tối ưu bằng xe bus với 2 điểm bất kỳ trên bản đồ.

PostGIS có một bộ mở rộng giúp tìm đường đi ngắn nhất, đó là pgRouting. Trong bài tập này, em sử dụng hàm pg_KSP để tìm k đường đi ngắn nhất với điểm bus xuất phát và điểm bus đích cho trước, dựa trên dữ liệu của hệ thống bus. Hàm này dựa vào thuật toán Dijkstra.

Vấn đề đặt ra, đó là việc định nghĩa đường đi được coi là tối ưu. Thông thường khi chọn lộ trình bus phù hợp, các yếu tố được cân nhắc khi ta chỉ có dữ liệu địa lý các tuyến bus bao gồm số lần chuyển bus và độ dài đường đi.

Trong bảng edge_table, trường change_bus_cost dùng để xác định chi phí chuyển bus, trường geom cho phép ta tính độ dài đường đi. Như vậy, đường đi tối ưu và đường đi thỏa mãn cực tiểu được một hàm là tổng có trọng số của hai tham số trên. Đối với từng ngữ cảnh, ta có thể chọn các trọng số sao cho phù hợp.

Câu query dưới đây được thực hiện đối với điểm bắt đầu 738815106 (Viện Tin học Pháp ngữ - Đối diện SVĐ ĐHBK) và điểm đích 738813730 (ĐH Sư phạm Ngoại ngữ), chi phí được xác định bằng : distance_cost + change_bus_cost*1000.

```
select
        pgr_KSP.path_id, pgr_KSP.path_seq,
        source.name,
        ST_Transform(source.way, 4326) as source_location,
        target.name,
        ST_Transform(target.way, 4326) as target_location,
        bus_merged_route.ref,
        ST_Transform(bus_merged_route.way, 4326) as bus_route,
        ST_Transform(edge_table_v2.way, 4326) as way
from pgr_KSP('SELECT
        id,
        source,
        target,
        distance_cost + change_bus_cost*1000 as cost,
        x1,y1,x2,y2
        FROM edge_table_v2', 738815106, 738813730, 3)
join edge_table_v2 on pgr_KSP.edge = edge_table_v2.id
join bus_merged_route on edge_table_v2.bus_id = bus_merged_route.osm_id
join stop source on edge_table_v2.source = source.osm_id
join stop target on edge_table_v2.target = target.osm_id
```

Kết quả thu được như sau:

path. integ	path_ integ	name text	source_locati geometry	name text	target_locati geometry	ref text	bus_route geometry	way geometry
1	1	Viện Tin học Pháp ngữ	010100002...	Cổng Parabol Đại học Bách khoa (cột trước)	01010000...	26	0102000020E6100...	01020000...
1	2	Cổng Parabol Đại học Bách khoa (cột trước)	010100002...	Đối diện Bến xe khách Mỹ Đình (cột trước)	01010000...	21B	0102000020E6100...	01020000...
1	3	Đối diện Bến xe khách Mỹ Đình (cột trước)	010100002...	ĐH Sư phạm Ngoại ngữ	01010000...	60A	0102000020E6100...	01020000...
2	1	Viện Tin học Pháp ngữ	010100002...	109 Phạm Ngọc Thạch	01010000...	26	0102000020E6100...	01020000...
2	2	109 Phạm Ngọc Thạch	010100002...	Đối diện Bến xe khách Mỹ Đình (cột trước)	01010000...	21B	0102000020E6100...	01020000...
2	3	Đối diện Bến xe khách Mỹ Đình (cột trước)	010100002...	ĐH Sư phạm Ngoại ngữ	01010000...	60A	0102000020E6100...	01020000...
3	1	Viện Tin học Pháp ngữ	010100002...	Công ty in thương mại & dịch vụ ngân hàng	01010000...	26	0102000020E6100...	01020000...
3	2	Công ty in thương mại & dịch vụ ngân hàng	010100002...	Đối diện Bến xe khách Mỹ Đình (cột trước)	01010000...	21B	0102000020E6100...	01020000...
3	3	Đối diện Bến xe khách Mỹ Đình (cột trước)	010100002...	ĐH Sư phạm Ngoại ngữ	01010000...	60A	0102000020E6100...	01020000...

Trên hình có 3 đường đi được trả về:

- Đường đi thứ nhất:
    1. Đi xe số 26: Viện Tin học Pháp ngữ -> Cổng Parabol - Đại học Bách Khoa (cột trước).
    2. Đi xe số 21B: Cổng Parabol - Đại học Bách Khoa (cột trước) -> Đối diện Bến xe khách Mỹ Đình (cột trước).
    3. Đi xe số 60A: Đối diện Bến xe khách Mỹ Đình (cột trước) -> ĐH Sư phạm Ngoại ngữ.
- Đường đi thứ 2:
    1. Đi xe số 26: Viện Tin học Pháp ngữ -> 109 Phạm Ngọc Thạch.
    2. Đi xe số 21B: 109 Phạm Ngọc Thạch -> Đối diện Bến xe khách Mỹ Đình (cột trước).
    3. Đi xe số 60A: Đối diện Bến xe khách Mỹ Đình (cột trước) -> ĐH Sư phạm Ngoại ngữ.
- Đường đi thứ 3:
    1. Đi xe số 26: Viện Tin học Pháp ngữ -> Công ty thương mại và dịch vụ Ngân hàng.
    2. Đi xe số 21B: Công ty thương mại và dịch vụ Ngân hàng -> Đối diện Bến xe khách Mỹ Đình (cột trước).
    3. Đi xe số 60A: Đối diện Bến xe khách Mỹ Đình (cột trước) -> ĐH Sư phạm Ngoại ngữ.

4. Chương IV: Kết luận

Trong bài tập này, chúng em đã hoàn thành công việc xây dựng một cơ sở dữ liệu quản lý hệ thống xe bus, đồng thời thực hiện một số bài toán ứng dụng cơ bản trên cơ sở dữ liệu trên, bao gồm: tìm các điểm bus lân cận một điểm, tìm các tuyến bus đi qua một khu vực, và tìm đường tối ưu giữa hai điểm bus.

Vấn đề còn tồn tại liên quan đến dữ liệu đầu vào có thể được khắc phục bằng việc chỉnh sửa dữ liệu từ file osm đầu vào.

Qua bài tập này, chúng em đã ứng dụng các lý thuyết cơ bản về cơ sở dữ liệu địa lý để xây dựng một cơ sở dữ liệu thực tế, từ đó củng cố và nắm vững các kiến thức học được.