Lab Project: OpenStreetMap

Zhenfeng Shi, Hongru Zhu, Chang Zhou

5130309777, 5130309784, 5130309787

Abstract

OpenStreetMap powers map data on thousands of web sites, mobile apps, and hardware devices. It is built by a community of mappers that contribute and maintain data about roads, trails, cafs, railway stations, and much more, all over the world. Our task is to load the OpenStreetMap data into MySQL database system, and make suitable tables, indexes and queries for them.

Keywords: OSM, Database

1. Usage

1.1. Environment

MySQL + Python 3 + pymysql + Firefox + Django + leaflet

1.2. File Structure

The following tree demonstrate the useful files in the folders, other files can be ignored.

```
.
├── DataInsertion
│   ├── dumpin.py
│   └── utils.py
├── Query
│   ├── calc_dist.py
│   ├── mysql2xml.py
│   ├── Query.py
│   └── utils.py
├── readme.md
├── Report
│   └── Group10_FinalReport.pdf
├── SZZ_install.py
├── TableCreation
│   ├── create_database.py
│   └── create_tables.py
└── XML
    ├── text2.xml
    └── text.xml
```

Figure 1: Tree for files

1.3. Install

Enter the root path of this project, run the following command in the shell:

```
python SZZ_install.py [-h] [-c host] [-u user] [-p passwd] [-n dbname] [-i input]

                      -c:  host connect, for instance 'localhost'
                      -u:  username for mysql, for instance 'root'
                      -p:  password for mysql, ignore this if no password
                      -n:  name for the new database
                      -i:  inputfile path, for instance '../shanghai_dump.osm'
```

For instance,

```
python SZZ_install -c localhost -u root -n OSM -i data/shanghai_dump.osm
```

1.4. Queries

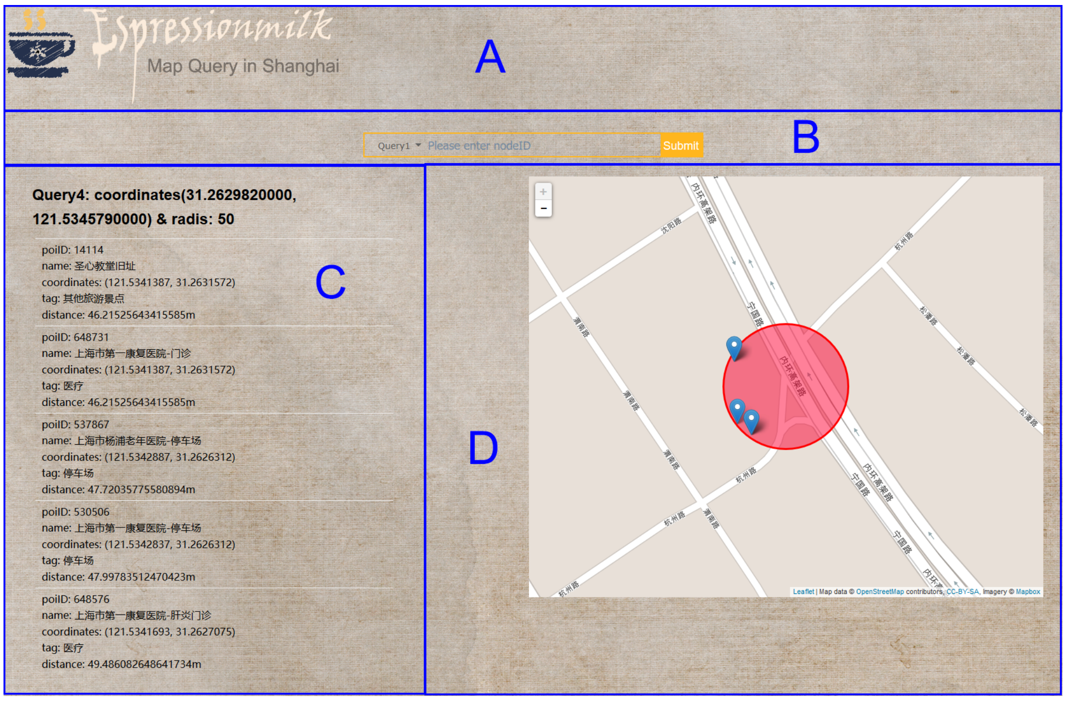
Website Overview

A part: logo part. Our website is called Espressionmilk having a wish that your experience in using our website could be as silky as sipping a cup of Espressionmilk.

B part: query type select & search input. Here you can select query type and input content

C part: result output in detail. It contains all the useful information in a clearer view.

D part: result output on map. It gives reader an intuitive view of the result on the map.


Espressionmilk
Map Query in Shanghai
A
Query1
Please enter nodeID
Submit
B
C
D
+
-
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Query4: coordinates(31.2629820000, 121.5345790000) & radis: 50

poiID: 14114
name: 圣心教堂旧址
coordinates: (121.5341387, 31.2631572)
tag: 其他旅游景点
distance: 46.21525643415585m

poiID: 648731
name: 上海市第一康复医院-门诊
coordinates: (121.5341387, 31.2631572)
tag: 医疗
distance: 46.21525643415585m

poiID: 537867
name: 上海市杨浦老年医院-停车场
coordinates: (121.5342887, 31.2626312)
tag: 停车场
distance: 47.72035775580894m

poiID: 530506
name: 上海市第一康复医院-停车场
coordinates: (121.5342837, 31.2626312)
tag: 停车场
distance: 47.99783512470423m

poiID: 648576
name: 上海市第一康复医院-肝炎门诊
coordinates: (121.5341693, 31.2627075)
tag: 医疗
distance: 49.486082648641734m

Figure 2: Website Overview

You need to install Django at first. Before using the website, you have to start the server:

```
python manage.py runserver
```

Then enter the address in your browser: http://localhost:8000/query-post.
You can use select menu to choose the query type.

Query 1
input format: nodeID
You can get a sentence indicate whether it is an intersection and a table of the wayID that contains this node.
Query 2
input format: wayID
You can get nodeID of the nodes along this way
Query 3
input format: string
You can get the road which has the name contains this string
Query 4
input format: longtitude/latitude/radius
Note that each value should be connected by /
You can get a list of POIs details meanwhile you can also have a direct view of their position on the online map.
Query 5
input format: longtitude/latitude
Note that each value should be connected by /
You can get the information of the closest road which is the nearest to your input position.
Query 6
input format: longtitude/latitude/ longtitude/latitude
Note that each value should be connected by /
After submit, we will process the xml file writing in the server and you can get a download link in our website.

2. Database Design

2.1. E-R Model

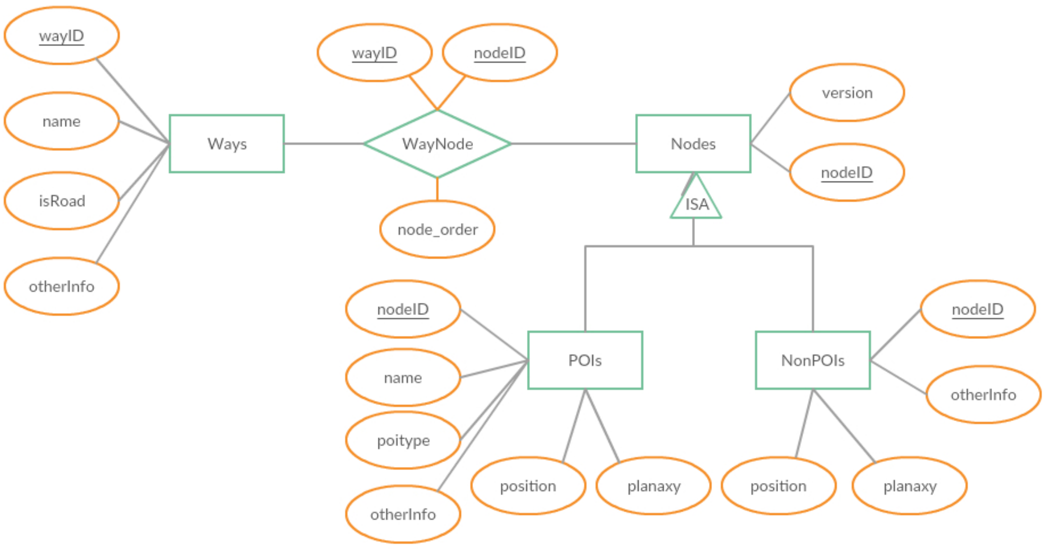


Figure 3: Entity Relationship Diagram

2.2. SQL For Table Creation

Codes are in 'OSM/TableCreation/create_tables.py'

```
CREATE TABLE ways(
      wayID VARCHAR(12),
          LineString LINESTRING,
          name VARCHAR(100), INDEX(name),
          isRoad VARCHAR(100),
          otherInfo TEXT,
          PRIMARY KEY(wayID)
        ) ENGINE=MyISAM

CREATE TABLE nodes(
          nodeID VARCHAR(12),
          version TINYINT(1), INDEX(version),
          version BOOLEAN,
          PRIMARY KEY(nodeID)
        ) ENGINE=MyISAM

CREATE TABLE POIs(
          nodeID VARCHAR(12),
          position POINT NOT NULL, SPATIAL INDEX(position),
          planaxy POINT NOT NULL, SPATIAL INDEX(planaxy),
          name VARCHAR(100), INDEX(name),
          poitype VARCHAR(100), INDEX(poitype),
          otherInfo TEXT,
          PRIMARY KEY(nodeID)
        ) ENGINE=MyISAM

create table nonPOIs(
          nodeID VARCHAR(12),
          position POINT NOT NULL, SPATIAL INDEX(position),
          planaxy POINT NOT NULL, SPATIAL INDEX(planaxy),
          otherInfo TEXT,
          PRIMARY KEY(nodeID)
        ) ENGINE=MyISAM

create table WayNode(
                    wayID VARCHAR(12), INDEX(wayID),
                    nodeID VARCHAR(12), INDEX(nodeID),
                    node_order INT(2),
                    FOREIGN KEY (nodeID) REFERENCES nodes(nodeID),
                    FOREIGN KEY (wayID) REFERENCES ways(wayID)
               ) ENGINE=MyISAM
```

2.3. Data Insertion

Codes are in 'OSM/DataInsertion/dumpin.py'

For the data we parsed from XML, we inserted them into corresponding fields of our created tables.

Notably, if we insert the data directly into the table, the insertion time complexity would be $O(log(N))$, where N is the entries already existed in the table, due to the index (primary key) building process.

Therefore, in order to speed up the insertion process, we disable all the keys before the insertion, and enable them after the insertion. This will ensure every row is inserted in time complexity $O(N)$.

The SQL code is as follows:

```
LOCK TABLE `nodes`, `pois`, `nonpois` WRITE;
ALTER TABLE `nodes` DISABLE KEYS;
ALTER TABLE `pois` DISABLE KEYS;
ALTER TABLE `nonpois` DISABLE KEYS;
/*...insertion...*/
ALTER TABLE `nodes` ENABLE KEYS;
ALTER TABLE `pois` ENABLE KEYS;
ALTER TABLE `nonpois` ENABLE KEYS;
UNLOCK TABLES;
```

The **LOCK TABLE** is to make sure no other users are writing at the same time.

2.4. Index

Besides index for primary keys, we built 8 indexes to accelerate the queries. Especially, in order to speed up the spatial queries, we applied Spatial Index in MySQL. For **MyISAM** tables, Spatial Index creates an R-tree index. The key idea of the R-tree is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree. For storage engines that support non-spatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values is useful for exact-value lookups, but not for range scans. In our cases, the R-tree is more suitable because required query 4, 5, 6 all include range scans.

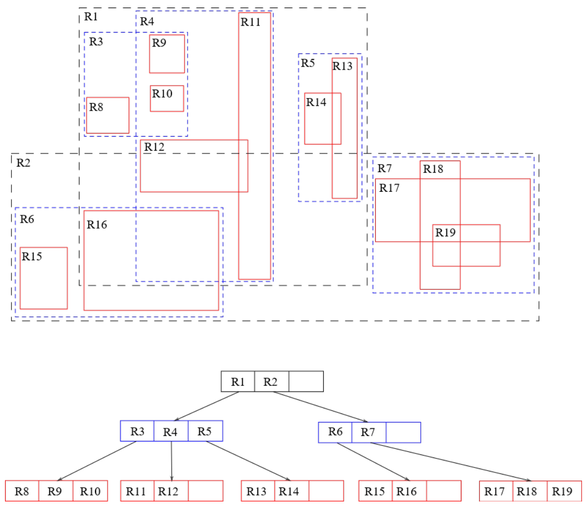


Figure 4: R-tree in 2 dimention

3. Point Mapping

The longtitude and latitude are used as the absolute coordinates. However, when calculating the distance between two points of given longtitude and latitude, we have to take spherical properties into consideration.

For instance, the distance between (30.4, 122.1) and (30.4, 122.6) is 48.0073 Km. The distance between (32.4, 122.1) and (32.4, 122.6) is 46.995 Km. There would be a error about 1 Km if we ignore the spherical properties of the Earth.

3.1. Different Ways of Calculating Distances Between Two Given Coordinates

a) Vincenty's formulae are two related iterative methods used in geodesy to calculate the distance between two points on the surface of a spheroid, developed by Thaddeus Vincenty (1975). They are based on the assumption that the figure of the Earth is an oblate spheroid, and hence are more accurate than methods that assume a spherical Earth. For simplicity and focus on the course related work, here we only provide the link to the Vincentys paper without further explanation. (`http://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf`)

b) Another approach is to map latitude-longitude coordinates to plana coordinates and then calculate the distance in between. We used the definition of Millers cylindrical projection, which is more accurate near the equator. Further about the derivation please see the original paper. (`http://www.jstor.org/stable/210384`)

3.2. Implementation

We randomly sampled the start and the destination and got the distribution of the distance error derived using two methods. We concluded that the Vincenty distance, which is more accurate, was 0.66 1.15 times of the Miller distance. This is further explored in our Query 4 and Query 5 design.

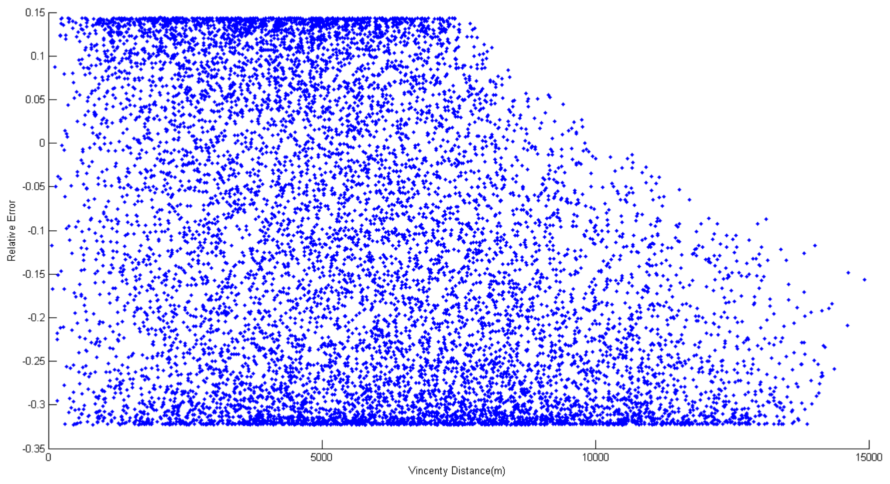


Figure 5: Relative error between miller distance and vincenty distance

4. Solution to Required Queries

Codes are in 'OSM/Query/Query.py'

1. Given a node, return all ways that contain it, and infer whether the node is an intersection of roads, i.e., a crossroad.
   Solution: This query is simple. Based on the given node ID, we go to table 'waynode' to find the corresponding way ID, and extract information about this way ID in table 'ways'.
   We have compared the speed of two queries:

```
SELECT * FROM ways
    WHERE wayID IN
    (SELECT wayID FROM waynode WHERE nodeID=givenNodeID);
or
SELECT wayID, LineString, name, isRoad, otherInfo
    FROM ways NATURAL JOIN waynode
    WHERE waynode.nodeID=givenNodeID;
```

   The corresponding runtimes are $0.72s$ and $8.29s$. Therefore, we choose to use the first solution instead of the second one.

2. Given a way, return all the nodes along the way.
   Solution: To realise this query, we designed a solution to lookup the tables multiple times. Firstly, we go to table 'waynode' to find the corresponding node ID based on the given way ID. Then find the detail information for these nodes in 'POIs' and 'Non-POIs'. Similar to query 1, we find the solution using the following SQL is faster than using JOIN.

```
tmpResult$\leftarrow$SELECT * FROM nodes
                        WHERE nodeID IN
                        (SELECT nodeID
                                  FROM waynode
                                  WHERE wayID=givenWayID);
    result=[];
    for row in tmpResult:
      if(row['version']==1):
        tmp$\leftarrow$SELECT nodeID, AsText(position) AS position, name, poitype, otherinfo
                            FROM pois
                            WHERE nodeID=row['nodeID'];
        result.append(tmp);
      else:
        tmp$\leftarrow$SELECT nodeID, AsText(position) AS position, otherInfo
                            FROM nonpois
                            WHERE nodeID=row['nodeID'];
        result.append(tmp);
```

3. Search the name of the road and return information of those matched.
   Solution: This query only involve one table 'ways'. The SQL is simple as follows:

```
    SELECT * FROM ways WHERE name LIKE ('%input_string%');
```

   In order to speed up the query process, we built index for 'name' in 'ways'.
4. Query the POIs within a radius of a given location (Longtitude-latitude coordinates).
   Solution: In this query, we first convert given coordinate to 'planaxy', and draw a circle with a larger radius than required, namely 1.33 times of the original radius. This is to ensure that all nodes within the radius under Vincenty distance will always be included. According to our simulation above. Later we use spatial index to scan for all nodes in a polygon that perfectly circumscribe the desired circle. After we got all possible nodes, we use a linear time examination to find all nodes within the original radius. The core SQL commands responsible for this query are:

```
    SET @poly='Polygon((x-rad, y+rad,
                        x+rad, y+rad,
                        x+rad, y-rad,
                        x-rad, y-rad,
                        x-rad, y+rad))';
    SELECT nodeID, ST_AsText(position), name, poitype
            FROM POIs
            WHERE MBRContains(ST_GeomFromText(@poly), planaxy);
```

5. Find the closest road to a given GPS coordinate.
   Solution: In this query, we first convert given coordinate to 'planaxy', and draw a circle with a larger radius than required simiar to Query 4. Next we use a iterative method to find at least on NONPOI point which is closest to the tartget GPS coordinate. To improve efficiency and fast our search, we use an exponentially growing raidus, starting from 10 meters as the initial radius. In the i^{th} attempt, we will search in a circle with radius $10e^{i-1}$ meters. We use the same technique as the above to use the spatial index in a polygon and run filters to get desired answer. The core SQL commands responsible for this query are:

```
    SET @poly='Polygon((x-rad, y+rad,
                        x+rad, y+rad,
                        x+rad, y-rad,
                        x-rad, y-rad,
                        x-rad, y+rad))';
    SELECT nodeID, ST_AsText(position)
        FROM nonPOIs
        WHERE MBRContains(ST_GeomFromText(@poly), planaxy);
    SELECT ways.wayid, ways.name, ways.isRoad, ways.otherInfo
        FROM waynode, ways
        WHERE waynode.nodeid=NID and waynode.wayid=ways.wayid and ways.isroad<>'0';
```

6. Implement an API to return the XML in osm format defined in the wiki page, given a rectangular area bounding box (x1, y1, x2, y2) as parameters.
   Solution: In this query, we firstly get all the nodes information from 'nodes', 'pois' and 'nonpois'. Then we go to table 'waynode' and 'ways' to find information of included ways.

```
        SET @poly='Polygon((x1, y1,
                            x1, y2,
                            x2, y2,
                            x2, y1,
                            x1, y1))';
        SELECT nodeID, AsText(position) as position, name, poitype, otherInfo
            FROM pois
            WHERE MBRContains(GeomFromText(@poly), position);
        SELECT nodeID, AsText(position) as position, otherInfo
            FROM nonpois
            WHERE MBRContains(GeomFromText(@poly), position);

        SELECT wayID, name, isRoad, otherInfo
            FROM ways
            WHERE wayID in
            (SELECT DISTINCT wayID
                FROM nonpois
                NATURAL JOIN waynode
                WHERE MBRContains(GeomFromText(@poly), position));
        SELECT wayID, name, isRoad, otherInfo
            FROM ways
            WHERE wayID in
            (SELECT DISTINCT wayID
                FROM pois
                NATURAL JOIN waynode
                WHERE MBRContains(GeomFromText(@poly), position));
```

Then we follow the format defined in the wiki page to output the xml file. In this process, we query the node IDs corresponding a way ID.

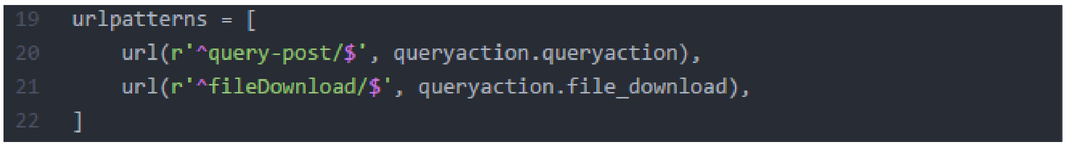
```
        SELECT nodeID, node_order FROM waynode
            WHERE wayID=givenWayID order by node_order;
```

5. Interface

5.1. Django Framework

We use Django Framework to design website. Django is an excellent Python Web framework based on MVC model which can be convenient to both query function and interface designing. Here we mainly talk about how our website communicate with query functions in the server and how our website make use of these result to create pretty appearance.

a. We only have one web page. All the actions are performed in one page.



```
urlpatterns = [
    url(r'^query-post/$', queryaction.queryaction),
    url(r'^fileDownload/$', queryaction.file_download),
]
```

Figure 6: Controller

One is for our homepage while the other is for our download link which will be included in Download Function

b. Basic interaction variable
We mainly have two input variable and two output variable.
The input variable contains query type and query content. Combined these two variables, the server could know which query to choose and execute it with the corresponding input.
The output variable contains query result in string type and query result in number type. The string type result can be used in C part. To pursue a clearer view, we rearrange the result so that their attributes can be stick out a mile. The number type results are mainly made up of coordinates which are used to represent the points and areas on the map plugin.

5.2. Optimized Input Form

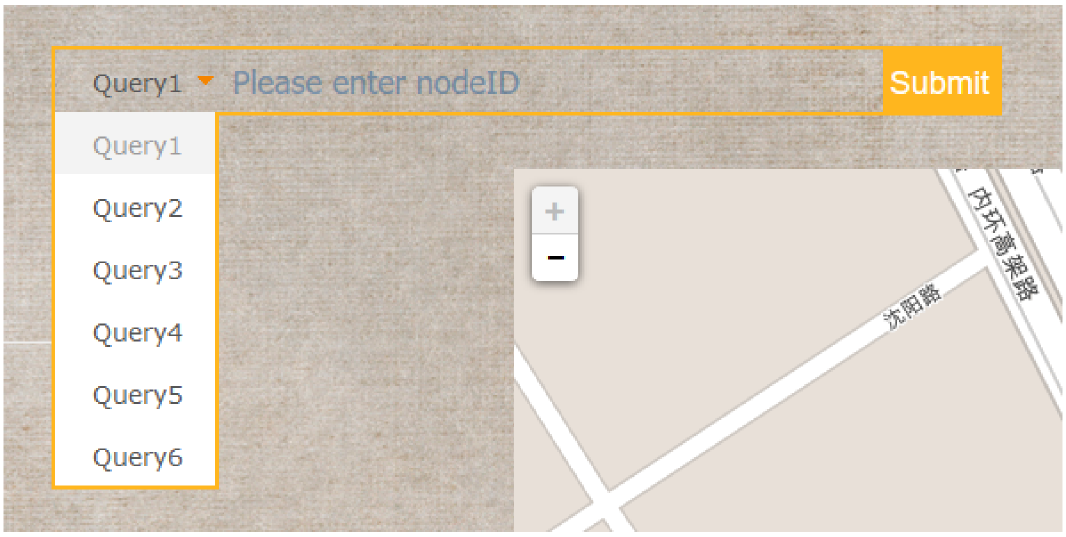


Figure 7: Select menu

Our input form is characteristic. All the query inputs are entered in the same form. The query type can be selected in the select form. When you select one type, the place holder will also change to remind your input format.

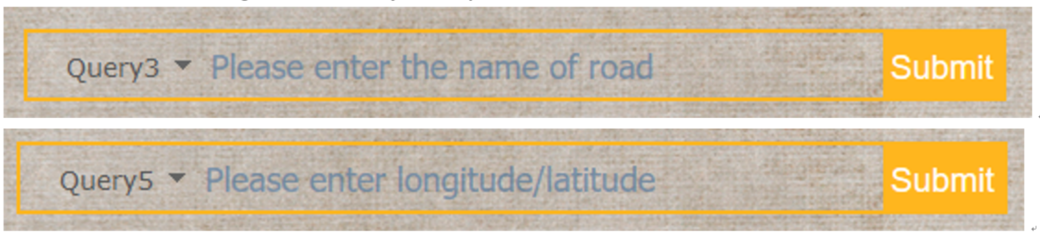
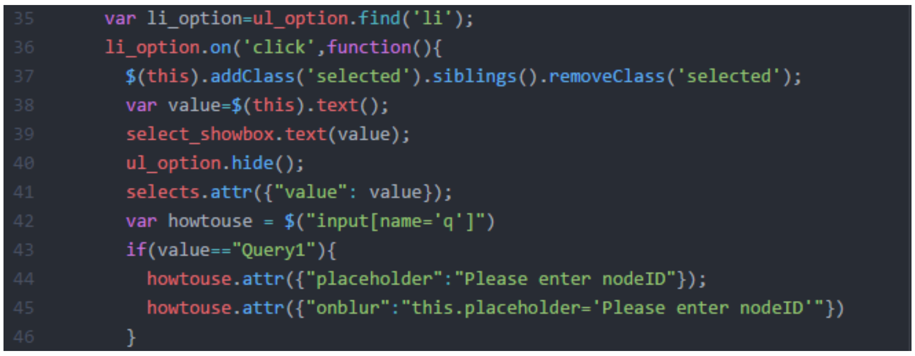

Query3
Please enter the name of road
Submit
Query5
Please enter longitude/latitude
Submit

Figure 8: Query3 and Query4 indication

We use JQuery to manage its action. When user select a query type, it will trigger a series of actions: change the value of query type, change the content of placeholder.



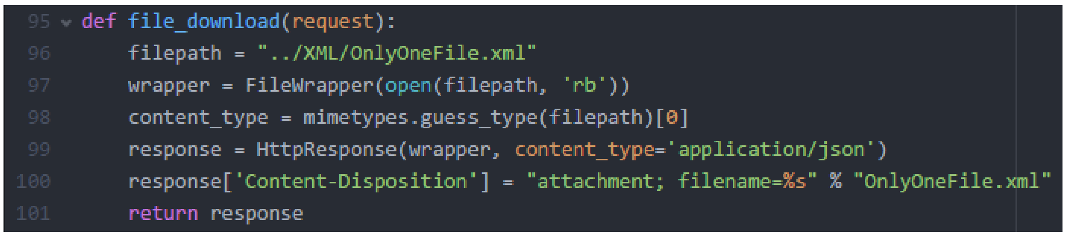
```
    var li_option=ul_option.find('li');
    li_option.on('click',function(){
      $(this).addClass('selected').siblings().removeClass('selected');
      var value=$(this).text();
      select_showbox.text(value);
      ul_option.hide();
      selects.attr({"value": value});
      var howtouse = $("input[name='q']")
      if(value=="Query1"){
        howtouse.attr({"placeholder":"Please enter nodeID"});
        howtouse.attr({"onblur":"this.placeholder='Please enter nodeID'"})
      }
```

Figure 9: JQuery

5.3. *Download Function*

The query 6 requires us to writing a xml file. In order to present our work in website, we produce download function. The rough idea is to write the xml file and then use wrapper function to send it to the user.

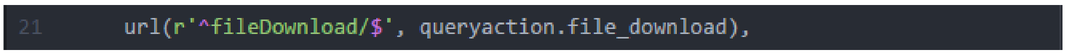
Model part of MVC: it mainly deals with the process of reading and wrapping. It finally returns a download response. Controller part of MVC: it monitors the front operations and when it detects the prescribed link, it will tell the model part



```python
def file_download(request):
    filepath = "../XML/OnlyOneFile.xml"
    wrapper = FileWrapper(open(filepath, 'rb'))
    content_type = mimetypes.guess_type(filepath)[0]
    response = HttpResponse(wrapper, content_type='application/json')
    response['Content-Disposition'] = "attachment; filename=%s" % "OnlyOneFile.xml"
    return response
```

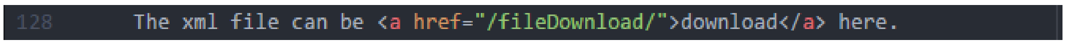
Figure 10: Model of MVC

to execute the corresponding function. View part of MVC: a quite simple hyperlink.



```python
    url(r'^fileDownload/$', queryaction.file_download),
```

Figure 11: Controll of MVC

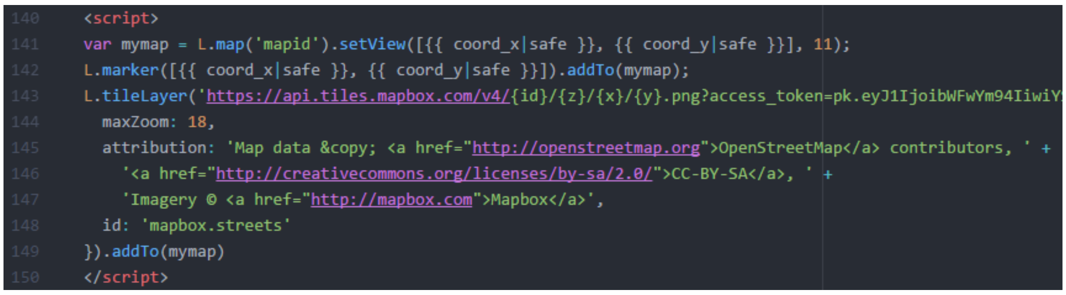


```html
    The xml file can be <a href="/fileDownload/">download</a> here.
```

Figure 12: View of MVC

5.4. *Map Plugin*

We use leaflet, an open-source web map plugin in javascipt to produce our map function. The main function is to denote the positions and areas on the map. We first create a map view centered with the corresponding coordinates. Then we pick out the positions according to the result. In the end, we cant forget to add the authority so that the map function can be loaded.



```html
<script>
var mymap = L.map('mapid').setView([{{ coord_x|safe }}, {{ coord_y|safe }}], 11);
L.marker([{{ coord_x|safe }}, {{ coord_y|safe }}]).addTo(mymap);
L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token=pk.eyJ1IjoibWFwYm94IiwiY
  maxZoom: 18,
  attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributors, ' +
    '<a href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, ' +
    'Imagery © <a href="http://mapbox.com">Mapbox</a>',
  id: 'mapbox.streets'
}).addTo(mymap)
</script>
```

6. Division of Work

Zhenfeng Shi: Schema designing; Database, table, index creation; Required query 1, 2, 3, 6.

Hongru Zhu: Algorithms dealing with geo-spatial data designing; Required query 4, 5.

Chang Zhou: Demonstration and interfaces.