

MSiA-413 Introduction to Databases and Information Retrieval

Homework 6: Accessing large real-world databases

Name 1: Alicia Burris

NetID 1: anb0847

Name 2: Naomi Kaduwela

NetID 2: nak133

Instructions

You should submit this homework assignment via Canvas. Acceptable formats are word files, text files, and pdf files. Paper submissions are not allowed and they will receive an automatic zero.

As explained during lecture and in the syllabus, assignments are done in groups. The groups have been created and assigned (randomly) by the instructors. Each group needs to submit only one assignment (i.e., there is no need for both partners to submit individually the same homework assignment).

Each group can submit solutions multiple times (for example, you may discover an error in your earlier submission and choose to submit a new solution set). We will grade only the last submission and ignore earlier ones.

Make sure you submit your solutions before the deadline. The policies governing academic integrity, tardiness and penalties are detailed in the syllabus.

To help you prepare for the final exam, we plan to release the solutions as soon as the assignment's deadline is over, so you will have at least a day to study it. Thus, please note that **there will be no grace period for this assignment**. Late submissions will receive an automatic zero.

Due Date: Tuesday December 4, 11:59 pm – No grace period

General Instructions

In this assignment you will write queries on a large, real-world dataset stored in a Postgres database server. To connect to this server you will use your MSiA credentials. If you have trouble logging into the database server, please email the MSiA root, Lewis Meineke, at meineke@northwestern.edu and cc the instructor.

To connect, you'll have to be on a Northwestern network, either on campus or through the NU VPN (virtual private network). Northwestern has several WiFi networks—you need to use the one called “Northwestern” so you can authenticate with your NetID. You can find instructions for setting up the VPN for off-campus access at <http://www.it.northwestern.edu/oncampus/vpn/>.

One way to connect to a database hosted on the Postgres server at MSiA is to login to Northwestern VPN and then **ssh** to **app1.lab.analytics.northwestern.edu** with your MSiA cluster credentials. Then, you can login to the database by issuing the “**psql -d databaseName**” command. For example:

```
$ ssh username@app1.lab.analytics.northwestern.edu
username@app1.lab.analytics.northwestern.edu's password: .....
[username@app1 ~]$ psql -h pg.analytics.northwestern.edu -d yelp
Password for user username: .....
psql (9.6.10, server 10.5)
WARNING: psql major version 9.6, server major version 10.
Some psql features might not work.
Type "help" for help.

yelp=>
```

You can also use the remote desktop server at **ts2.lab.analytics.northwestern.edu** to connect to the Postgres server. You should be able to connect to that with your username written as **MCC\username**. It has several programs installed that can access the Postgres database server. During bootcamp you used the official graphical interface client for PostgreSQL, PgAdmin. However, this client is still buggy. Thus, unlike the previous homeworks where you used a well-developed graphical user interface for SQLite, in this assignment it is recommended that you work with the command-line interface of Postgres.

You may find the following Postgres commands useful. Some of these commands were also covered at bootcamp.

The command “**\d**” presents a list of the relations in the database:

```
yelp=> \d
List of relations
Schema | Name      | Type  | Owner
-----+-----+-----+-----
public | attribute | table | nha224
public | business  | table | nha224
public | category  | table | nha224
public | checkin   | table | nha224
public | elite_years | table | nha224
public | friend     | table | nha224
public | hours      | table | nha224
public | photo      | table | nha224
public | review     | table | nha224
public | tip        | table | nha224
public | user       | table | nha224
(11 rows)
```

The command “**\d tableName**” presents a description of table with name **tableName**, including the column names and types, default values, foreign keys, and available indexes:

```
yelp=> \d user
```

Table "public.user"				
Column	Type	Collation	Nullable	Default
id	character varying(22)		not null	
name	character varying(255)			NULL::character varying
review_count	integer			
yelping_since	time without time zone			
useful	integer			
funny	integer			
cool	integer			
fans	integer			
average_stars	double precision			
compliment_hot	integer			
compliment_more	integer			
compliment_profile	integer			
compliment_cute	integer			
compliment_list	integer			
compliment_note	integer			
compliment_plain	integer			
compliment_cool	integer			
compliment_funny	integer			
compliment_writer	integer			
compliment_photos	integer			

Indexes:

"user_pkey" PRIMARY KEY, btree (id)

Referenced by:

TABLE "elite_years" CONSTRAINT "fk_elite_years_user1" FOREIGN KEY (user_id) REFERENCES "user"(id)

TABLE "friend" CONSTRAINT "fk_friends_user1" FOREIGN KEY (user_id) REFERENCES "user"(id)

TABLE "review" CONSTRAINT "fk_reviews_user1" FOREIGN KEY (user_id) REFERENCES "user"(id)

TABLE "tip" CONSTRAINT "fk_tip_user1" FOREIGN KEY (user_id) REFERENCES "user"(id)

Note that the database has a table named "**user**". However, "**user**" is a reserved keyword in the SQL standard, and thus also in PostgreSQL (as Postgres' variant of SQL is known). To access the table named "**user**" in the yelp database you need to use "**public.user**" instead, as in the example below:

```
SELECT id FROM public.user ORDER BY id LIMIT 10;
```

Postgres provides a facility to time the execution of queries. Turn on execution timing by issuing the command "**\timing**" while in the yelp database prompt. The database server will reply with "**Timing is on.**" to notify you that your command succeeded. Note that if you issue the command "**\timing**" again, timing will turn off.

```
yelp=> \timing
```

Timing is on.

```
yelp=> \timing
```

Timing is off.

Note that PostgreSQL adheres to the SQL standard a little more than many other database systems. For example, when a SQL statement contains a GROUP BY clause, the columns that appear in the SELECT clause should have a unique value within each group (and, hence, these columns should also appear in the GROUP BY clause). For example, the query below is correct and will return results:

```
SELECT id, name, COUNT(*) FROM public.user GROUP BY id, name;
```

However, the following query will fail with an error, because the **GROUP BY** clause does not include "**id**":

```
SELECT id, name, COUNT(*) FROM public.user GROUP BY name;
```

ERROR: column "user.id" must appear in the GROUP BY clause or be used in an aggregate function

LINE 1: SELECT id, name, COUNT(*) FROM public.user GROUP BY name;

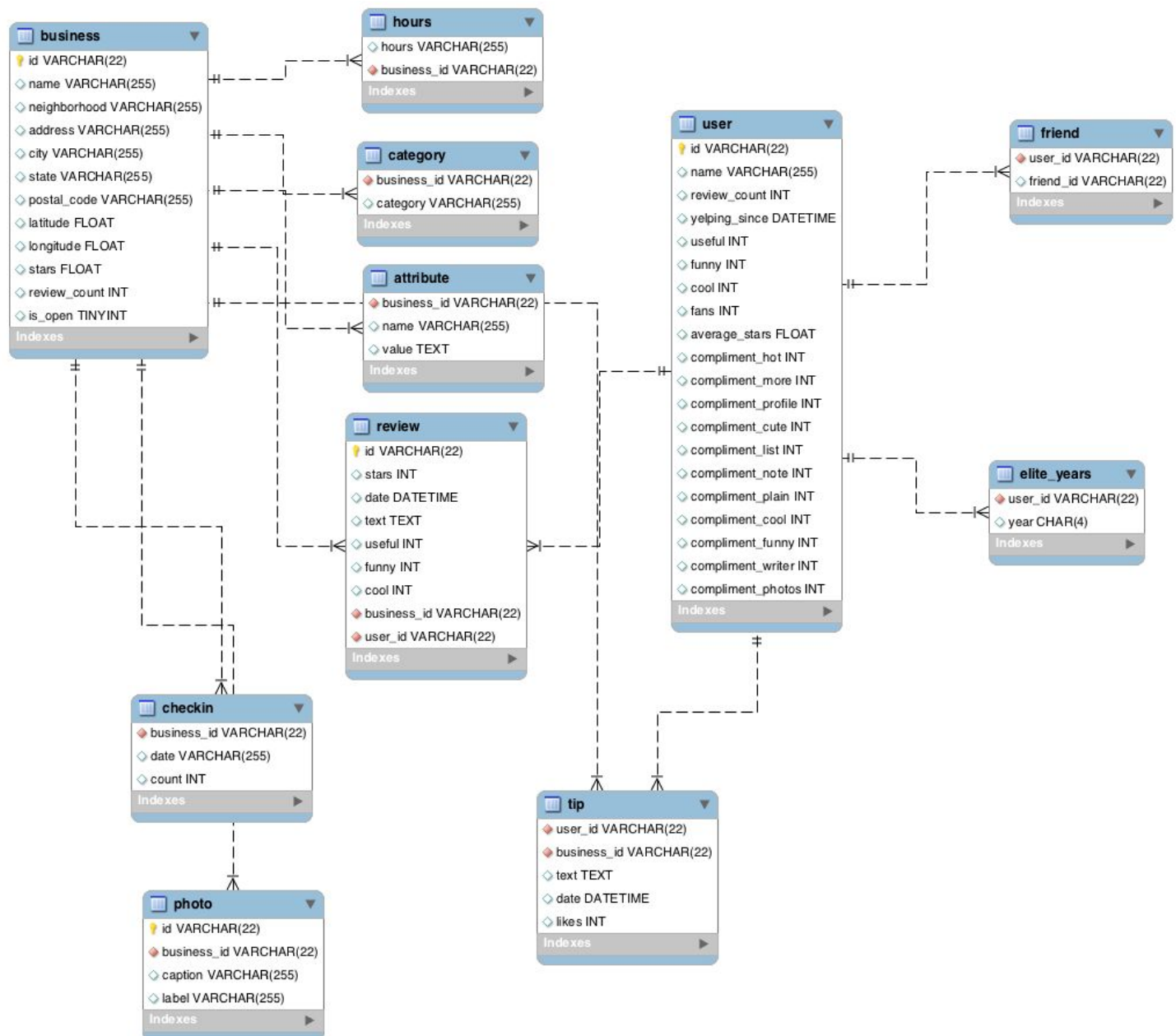
You can exit from the database by typing “\q” or by pressing **CONTROL-d**.

```
yelp=> \q  
[username@app1 ~]$
```

Yelp Database (yelp)

The database “yelp” has data from the Yelp business review app (<http://yelp.com/>). It provides access to data from 12 metropolitan areas and 4.7 million reviews of 156,639 businesses. It also includes data on 1.1 million users and 1 million “tips” from these users. The database is about 10.8 GiB. This makes the yelp dataset a medium-sized database: large enough to require a well-engineered database server, but a dwarf compared to truly big data.

The database schema is provided below:



You will use this database to answer the following questions. Unless otherwise noted, for each question please provide:

- The query you constructed
- The output of that query
- Any other information requested by the question (e.g., timing results)

- 1) How many rows are in the *friend* table and how long (in seconds or ms) does it take to count them? In your answer provide (a) the query, (b) the output, and (c) the number of seconds it took you to run the query. *Note:* This query may take a long time.

Answer:

a.)

yelp=> \timing

Timing is on.

yelp=> select count(*) from friend;
count

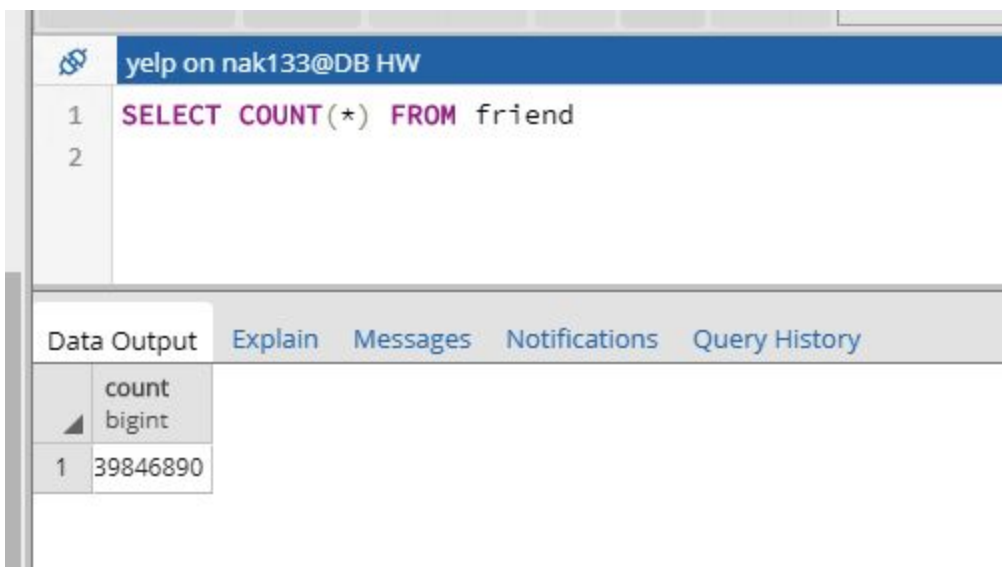
b.)

39846890

(1 row)

c.)

Time: 1439.280 ms



The screenshot shows a database interface with a query editor at the top and a results pane below. The query editor contains the SQL query: `SELECT COUNT(*) FROM friend`. The results pane has tabs for "Data Output", "Explain", "Messages", "Notifications", and "Query History". The "Data Output" tab is selected, showing a table with one column named "count" of type "bigint" and one row with the value 39846890.

	count bigint
1	39846890



The screenshot shows the same database interface as above, but with the "Messages" tab selected in the results pane. It displays the message: "Successfully run. Total query runtime: 1 secs 509 msec. 1 rows affected."

Data Output	Explain	Messages	Notifications	Query History
Successfully run. Total query runtime: 1 secs 509 msec. 1 rows affected.				

- 2) Find the name of the businesses for which there is a review that contains the case-insensitive text string "wing" at least 25 times in the same review. *Hint 1:* You do not have to search for complete words but only for **text strings** that are not case sensitive, i.e., "sunwing", "wing", "winging", "Wings", "WiNg" are all hits. *Hint 2:* The

regular expressions format in PostgreSQL is different than the MySQL variant we discussed in class. PostgreSQL does pattern matching with regular expressions using the ***SIMILAR TO*** operator, where “***_***” matches any character and “***%***” matches any sequence of zero or more characters. The remaining rules are similar to the ones we learned in class, e.g., parentheses “***()***” are used to group items together into a single logical item, brackets “***[]***” are used to denote a class of characters, angled brackets “***{ }***” are used to denote repetition, etc. The regular expressions syntax rules for PostgreSQL you need for this question can be found at <https://www.postgresql.org/docs/9.3/functions-matching.html>.

Answer:

```
yelp=> select name from business join review on business.id = review.business_id where review.id in (select id
from review where text ~* '(*wing.*){25,}');
      name
```

```
-----
The Firehall Cool Bar Hot Grill
Wingstop
Buffalo Wild Wings
Puck'n Wings
Wing Time
(5 rows)
```

38 select name from business join review on business.id = review.business_id
39 where review.id in (select id from review where text ~* '(*wing.*){25,}')

40

Query History
Messages
Explain
Data Output

	name
1	Wing Time
2	Puck'n Wings
3	Wingstop
4	The Firehall Cool Bar Hot G...
5	Buffalo Wild Wings

- Find the top 3 users that have provided the largest number of reviews of businesses within a range of 0.1 degrees from latitude 36.0 and longitude -115.0. For each one of these users, provide in your answer the name and the number of reviews that user provided for businesses within that range. *Hint:* You can use the Pythagorean theorem to find businesses within the requested range. For example, locations within d degrees from latitude X and longitude Y satisfy the formula ***$\text{sqrt}(\text{power}(\text{longitude} - y, 2.0) + \text{power}(\text{latitude} - x, 2.0)) \leq d$*** .

Answer:


```

19 select public.user.name, count(review.user_id)
20 from public.user join review on public.user.id = review.user_id
21 join business on business.id = review.business_id
22 where (sqrt(power(business.longitude + 115, 2.0) +
23 power(business.latitude - 36, 2.0)) <= .1)
24 group by public.user.name, review.user_id
25 order by count(review.user_id) desc limit 3;
26
27

```

Data Output Explain Messages Notifications Query History

	name character varying (255)	count bigint
1	Bonnie	227
2	Shirley	213
3	Bethany	209

- 4) What is the name, address (including city, state, postal code), and average rating of the highest-rated restaurant with “McDonald” in its name? *Hint 1:* You must use the category named “**Restaurants**”, otherwise you’ll get results for other types of businesses with “**McDonald**” in the name. *Hint 2:* We are asking for the restaurant with the highest **average** rating. Many such restaurants have at least one 5-star rating, but only one location has a star rating **average** close to 5. *Hint 3:* You do not need to concatenate the address into a single string. It is OK for the address, city, state and postal code to occupy a separate column each in your resulting table.

Answer:

```

yelp=> select avg(review.stars), business.address, business.name, business.city, business.state, business.postal_code
from business join category on business.id = category.business_id join review on review.business_id = business.id
where category = 'Restaurants' and business.name like '%McDonald%' group by review.business_id,
business.address, business.name, business.city, business.state, business.postal_code order by avg(review.stars) desc
limit 1;

```

```

yelp on nak133@DB HW
1 SELECT AVG(review.stars), b.name, b.address, b.city, b.state, b.postal_code
2 FROM business b
3 JOIN category ON b.id = category.business_id
4 JOIN review ON review.business_id = b.id
5 WHERE category = 'Restaurants' AND b.name LIKE '%McDonald%'
6 GROUP BY review.business_id, b.name, b.city, b.state, b.postal_code, b.address
7 ORDER BY AVG(review.stars) DESC LIMIT 1;
8

```

Data Output Explain Messages Notifications Query History

	avg numeric	name character varying (255)	address character varying (255)	city character varying (255)	state character varying (255)	postal_code character varying (255)
1	4.833333333333333	McDonald's McCafe	100 King Street W, Exchang...	Toronto	ON	M5X 2A2

5) What are the names of the businesses for which there are at least 5 reviews where each one of these reviews contains the text “**barf**”? *Hint:* Similarly to question 2, you do not have to match individual words, but only sub-strings. For example, “barf”, “barfing” and “barfday” should all be considered hits.

Answer:

```
yelp=> select business.name from business join review on business.id = review.business_id where review.text like '%barf%' group by business.name, review.business_id having count(review.id) >=5;
      name
```

Spirit Airlines
Wicked Spoon

(2 rows)

```
6 SELECT b.name, COUNT(r.id)
7 FROM business b
8 JOIN review r on r.business_id = b.id
9 WHERE r.text LIKE '%barf%'
10 GROUP BY b.name
11 HAVING COUNT(r.id) >=5 ;
```

Data Output	Explain	Messages	Notifications	Query History
	name character varying (255)	rcount bigint		
1	Spirit Airlines	6		
2	Wicked Spoon	5		

6) With execution timing on, find the name of the user with id '**CxDOIDnH8gp9KXzpBHJYXw**'. Include the time it took to execute the query in your answer.

Answer:

```
yelp=> select name from public.user where id = 'CxDOIDnH8gp9KXzpBHJYXw';
      name
```

Jennifer
(1 row)

Time: 1.145 ms

```

4 SELECT public.user.name
5 FROM public.user
6 WHERE public.user.id LIKE 'CxDOIDnH8gp9KXzpBHJYXw'

```

Data Output Explain Messages Notifications Query History

	name
	character varying (255)
1	Jennifer

```

4 SELECT public.user.name
5 FROM public.user
6 WHERE public.user.id LIKE 'CxDOIDnH8gp9KXzpBHJYXw'

```

Data Output Explain Messages Notifications Query History

Successfully run. Total query runtime: 46 msec.
1 rows affected.

- 7) With execution timing on, find the name of the user with 3336 compliment_plain compliments. Include the time it took to execute the query in your answer.

Answer:

yelp=> select name from public.user where compliment_plain = 3336;

name

Jennifer
(1 row)

Time: 92.471 ms

4	SELECT public.user.name
5	FROM public.user
6	WHERE public.user.compliment_plain = 3336

Data Output	Explain	Messages	Notifications	Query History
-------------	---------	----------	---------------	---------------

	name
	character varying (255)
1	Jennifer

4	SELECT public.user.name
5	FROM public.user
6	WHERE public.user.compliment_plain = 3336

Data Output	Explain	Messages	Notifications	Query History
-------------	---------	----------	---------------	---------------

Successfully run. Total query runtime: 137 msec.
1 rows affected.

- 8) Which query is faster, query 6 or query 7, and by how much? Also, please explain why it is faster. *Note:* this question does not ask you to write a query or provide a query's output. Simply provide your answers below.

Answer: Query 6 is faster than Query 7 by 91 milliseconds. This is because there is an index on id which is the column that query 6 is searching on. Because there is no index on compliment_plain, query 7 takes longer because that is the column the query is searching and then filtering on.

- 9) Find the name of the user that has given the largest number of useful reviews to closed businesses. Print both the user name and the number of such reviews the user has given.

Answer:

yelp on nak133@DB HW

```

1 SELECT u.name, count(r.useful) AS numUseful FROM public.review r
2 LEFT JOIN public.business b ON b.id=r.business_id
3 LEFT JOIN public.user u ON u.id=r.user_id
4 WHERE b.is_open=0 and r.useful>=1
5 GROUP BY r.user_id, u.name
6 ORDER BY numUseful DESC
7 LIMIT 10;

```

Data Output

[Explain](#)

[Messages](#)

[Notifications](#)

[Query History](#)

	name character varying (255)	numuseful bigint
1	Jennifer	716
2	Stefany	309
3	Norm	271
4	Gabi	260
5	Michael	232
6	J	227
7	Emily	220
8	Misti	209
9	Georgia	203
10	Jade	202

- 10) You are tasked with doing some city planning, which requires that you find clusters of businesses that are physically located very close to each other. Your first task is to find the IDs, names and GPS coordinates (latitude, longitude) of businesses that are clustered around McDonald's at address re. A business is considered part of the cluster if it is within 0.005 degrees away from any other business in the cluster. *Hint 1:* When you need to include an apostrophe as part of a text string in PostgreSQL, you need to escape it with another apostrophe, e.g., to find all McDonald's you need a query like **SELECT * FROM business WHERE name='McDonald's'**; *Hint 2:* You can use the Pythagorean theorem to find businesses within the requested range like in question 3.


```

yelp=> WITH RECURSIVE
yelp-> cluster(id,name,latitude,longitude)
yelp-> AS (
yelp(> (select id,name,latitude,longitude from business where address = 'Av. Maip 2779' and name = 'McDonald's')
yelp(> UNION
yelp(> SELECT b1.id,b1.name,b1.latitude,b1.longitude
yelp(> FROM business b1, business b2
yelp(> JOIN cluster
yelp(> ON b2.id=cluster.id
yelp(> where
yelp(> sqrt(power(b2.longitude - b1.longitude, 2.0) + power(b2.latitude - b1.latitude, 2.0)) <= .005
yelp(> )
yelp-> SELECT * FROM cluster;
      id      |      name      | latitude | longitude
-----+-----+-----+-----

```

softZjpREG65wpAns2FaWA	McDonald's	-34.51	-58.4911
bGxzQDGOTpab_6hdqsv9g	Burger King	-34.5089	-58.4919
i1e8KsIy1ELvI7G6mvvZkw	Havanna	-34.5133	-58.4894
m-SUr48X9gMHTwvraM-KmA	Compaa del Sol	-34.5134	-58.4896
WNsimvvr-0NimM57I5gj4A	Arnaldo	-34.5137	-58.4888
yadScsa2pShYsQAVXbNivw	La Farola de Olivos	-34.5108	-58.4908
YBaWP2r64BPJazkmyf1fig	Almacn de Pizzas	-34.5089	-58.4916
zMAiU0s8ScUYHwAESC8Qg	Prosciutto	-34.5122	-58.4898
4-xLjGavuWFqEfNuznxL3A	D' Lucky	-34.516	-58.4884
AwpX8mheEmMhaIuIqEhMkA	Estacin Mitre - Lnea Mitre	-34.515	-58.4897
Ss6J7HFhMCxoq7M8wXqc8A	Salve Bruna	-34.5159	-58.488

(11 rows)


yelp on nak133@DB HW

```

1 WITH RECURSIVE cluster(id,name,latitude,longitude) AS (
2   (SELECT id,name,latitude,longitude FROM business WHERE address = 'Av. Maip 2779' AND name = 'McDonald''s')
3   UNION
4   SELECT b1.id,b1.name,b1.latitude,b1.longitude
5   FROM business b1, business b2
6   JOIN cluster
7   ON b2.id=cluster.id
8   WHERE sqrt(power(b2.longitude - b1.longitude, 2.0) + power(b2.latitude - b1.latitude, 2.0)) <= .005)
9   SELECT * FROM cluster;

```

Data Output
[Explain](#)
[Messages](#)
[Notifications](#)
[Query History](#)

	id character varying (22)	name character varying (255)	latitude double precision	longitude double precision
1	softZjpREG65wpAns2FaWA	McDonald's	-34.51	-58.4911
2	bGxzQDGOTpab_6hdqsv9g	Burger King	-34.5089	-58.4919
3	i1e8KsIy1ELvI7G6mvvZkw	Havanna	-34.5133	-58.4894
4	m-SUr48X9gMHTwvraM-K...	Compaa del Sol	-34.5134	-58.4896
5	WNsimvvr-0NimM57I5gj4A	Arnaldo	-34.5137	-58.4888
6	yadScsa2pShYsQAVXbNivw	La Farola de Olivos	-34.5108	-58.4908
7	YBaWP2r64BPJazkmyf1fig	Almacn de Pizzas	-34.5089	-58.4916
8	zMAiU0s8ScUYHwAESC8...	Prosciutto	-34.5122	-58.4898
9	4-xLjGavuWFqEfNuznxL3A	D' Lucky	-34.516	-58.4884
10	AwpX8mheEmMhaIuIqEh...	Estacin Mitre - Lnea Mitre	-34.515	-58.4897
11	Ss6J7HFhMCxoq7M8wXq...	Salve Bruna	-34.5159	-58.488