Imperial College
London

CO130: Databases
2018-02-23

SQL

Tutorial II

Valentin CLÉMENT - vec16@ic.ac.uk

Department of Computing

Accessing the test server - reminder

You should all have read access to the following PostgreSQL database:

Host	: db.doc.ic.ac.uk
Port	: 5432
DB	: vec16
Username	: *Your college username*
Password	: Your ***Postgres*** password, which is ***not*** your college password! You can find it in your emails as **Your new PostgreSQL password and database for [...]**

In this DB, you only have access to selected *schemas*, such as *sn*.

- You have a similar DB called 'your username', everything else being similar. You only have very limited access on mine, you should have full access on yours, letting you go free
- If you want to copy one or more table, the following command might be helpful: CREATE TABLE staff AS TABLE vec16.doc.staff: it creates a table as the result of a query, in this case copying it.

Theoretical motivation

- Today we will tackle more intricate SQL with a small ’social network’ example
- Social Networks are interesting, as they examplify ***graphs***:

Definition

A graph is an ordered pair $G = (V, E)$ comprising a set V of vertices together with a set E of edges, which are 2-element subsets of V

- That sounds an awful lot like relations...

Theoretical motivation

- Today we will tackle more intricate SQL with a small 'social network' example
- Social Networks are interesting, as they examplify ***graphs***:

Definition

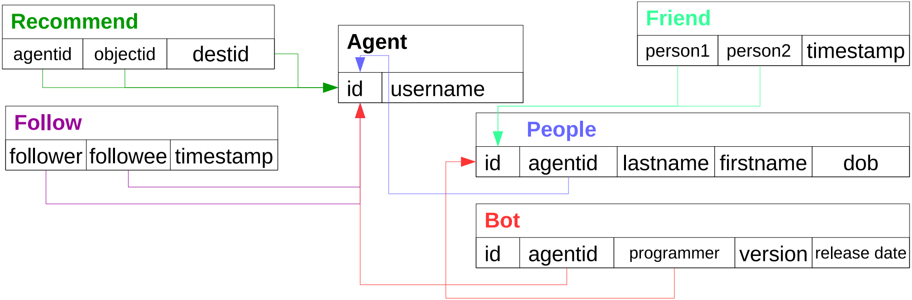
A graph is an ordered pair $G = (V, E)$ comprising a set V of vertices together with a set E of edges, which are 2-element subsets of V

- That sounds an awful lot like relations...
- And indeed, graphs are fundamentally isomorphic to relations $\mathit{vertex}(\underline{\mathit{id}}, \ldots, \mathit{vlab})$, $\mathit{edge}(\underline{\textbf{vertexid}_1}, \underline{\textbf{vertextid}_2}, \ldots, \mathit{elab})$

A toy social network

- The network represents the interaction of ***agent***(id, username)
- Of course, this is the 21^{st} century, so agents can be ***people***(id, **agentid**, lastname, firstname, dob) or ***bot***(id, **agentid**, **programmer**, version, releasedate)
- An agent can ***follow***(**follower**, **followee**, timestamp) another agent, human or not
- Two humans can be ***friend***(**person1**, **person2**, timestamp).
- Any agent can ***recommend***(**agentid**, **objectid**, **destid**) another agent to a third one.

Representation - *sn* schema



sn.agent (id, username)

sn.people (id, agentid(agent.id), lastname, firstname, dob)

sn.bot (id, agentid(agent.id), programmer(people.id), version, release)

sn.friend (person1(people.id), person2(people.id), timestamp)

sn.recommend (agentid(agent.id), objectid(agent.id), destid(agent.id))

sn.follow (follower(agent.id), followee(agent.id), timestamp)

Example 1

- List the first 50 *people* as: agent id, username, first name, last name and date of birth, ordered by *descending* date of birth.
- List, for each person, their first and last name and the number of bots they have programmed (different versions count as different bots), ordered by descending number of bots

Relevant tables:

sn.agent	(id, username)
sn.people	(id, agentid(agent.id), lastname, firstname, dob)
sn.bot	(id, agentid(agent.id), programmer(people.id), version, release)

Solution 1

- List people by descending DOB:

```
SELECT a.id, a.username, p.firstname,
          p.lastname, p.dob
FROM sn.agent a
JOIN sn.people p on p.agentid=a.id
ORDER BY p.dob DESC
LIMIT 50;
```

Solution 1

- List people by descending DOB:

```
SELECT a.id, a.username, p.firstname,
          p.lastname, p.dob
FROM sn.agent a
JOIN sn.people p on p.agentid=a.id
ORDER BY p.dob DESC
LIMIT 50;
```

- List people by bot programmed:

```
SELECT p.firstname, p.lastname, count(b.id)
FROM sn.people as p
JOIN sn.bot as b ON b.programmer = p.id
GROUP BY p.id
ORDER BY count(b.id) DESC;
```

Example 2

- List all the people’s firstname, lastname and bot username who have programmed a bot which followed them.
- For each *person*, list their last name, username and the total number of agents that *follow them*
- For each person, list their last name, username, the total number of people that follow them **and** the total number of agents that *they follow*
- Some people might have *no* followers and/or follow no one: make sure you still list them !

Relevant tables:

sn.agent (id, username)

sn.people (id, agentid(agent.id), lastname, firstname, dob)

sn.follow (follower(agent.id), followee(agent.id), timestamp)

Example 2

- List all the people’s firstname, lastname and bot username who have programmed a bot which followed them.
- For each *person*, list their last name, username and the total number of agents that *follow them*
- For each person, list their last name, username, the total number of people that follow them **and** the total number of agents that *they follow*
- Some people might have *no* followers and/or follow no one: make sure you still list them ! Hint: This means **[INNER] JOIN** will not be enough

Relevant tables:

sn.agent (id, username)

sn.people (id, agentid(agent.id), lastname, firstname, dob)

sn.follow (follower(agent.id), followee(agent.id), timestamp)

Solution 2 - i

- Users programming bots to follow themselves:

```
SELECT p.firstname, p.lastname, a.username
FROM sn.people p
JOIN sn.bot b ON b.programmer = p.id
JOIN sn.follow f ON f.follower = b.agentid
JOIN sn.agent a ON b.agentid=a.id
WHERE f.followee = p.agentid;
```

Solution 2 - ii

- Number of followers:

```
SELECT p.lastname, a.username,
         count(f.id) AS followers
FROM sn.agent a
JOIN sn.people p ON p.agentid = a.id
LEFT JOIN sn.follow f ON f.followee = a.id
GROUP BY a.id, a.username, p.lastname
ORDER BY a.id;
```

- Note the **LEFT JOIN**, necessary to account for people with no followers/following
- We **GROUP BY** the id and use it for ordering out of convenience: username/lastname are *not guaranteed* to be unique, whereas ids will provide us reliable and reproducible results.

Solution 2 - iii

- Followers and following:

```
SELECT p.lastname, a.username,
          count(distinct f1.id) as followers,
          count(distinct f2.id) as following
FROM sn.agent a
JOIN sn.people p ON p.agentid = a.id
LEFT JOIN sn.follow f1 ON f1.followee = a.id
LEFT JOIN sn.follow f2 ON f2.follower = a.id
GROUP BY a.id, a.username, p.lastname
ORDER BY a.id;
```

- Approach is very similar to the previous one
- Note we now need to count **distinct** ids: this is because the two full joins return all valid pairs

Example 3

- List all the friends of John Doe, in format lastname, firstname.
- Remember: friend relation can go either way: either *A* is a friend of *B*, or *B* is a friend of *A*
- There are duplicates in the database

Relevant tables:

sn.people (id, agentid(agent.id), lastname, firstname, dob)

sn.friend (person1(people.id), person2(people.id), timestamp)

Solution 3 - i

- Find the ID of John Doe:

```
SELECT p.id FROM sn.people p
WHERE p.lastname = ’Doe’ AND p.firstname = ’John’;
```

- Find his friends:

```
SELECT p.lastname, p.firstname
FROM sn.people p
WHERE p.id IN (
  SELECT DISTINCT p.id
  FROM sn.people p
  WHERE p.id IN (
    SELECT f.person1 FROM sn.friend f WHERE f.person2 = 1
  ) OR p.id IN (
    SELECT f.person2 FROM sn.friend f WHERE f.person1 = 1
  )
);
```

Solution 3 - ii

In all its horrible (but side-effect free !) glory

- List all friends of John Doe

```
SELECT p.lastname, p.firstname FROM sn.people p WHERE p.id IN (
  SELECT DISTINCT p.id FROM sn.people p WHERE p.id IN (
    SELECT p.id FROM sn.people p
    JOIN sn.friend f on p.id = f.person1
    JOIN sn.people q on q.id = f.person2
    WHERE q.id IN (
      SELECT p.id FROM sn.people p
      WHERE p.lastname = ’Doe’ AND p.firstname = ’John’
    )
  ) OR p.id IN (
    SELECT p.id FROM sn.people p
    JOIN sn.friend f on p.id = f.person2
    JOIN sn.people q on q.id = f.person1
    WHERE q.id IN (
      SELECT p.id FROM sn.people p
      WHERE p.lastname = ’Doe’ AND p.firstname = ’John’
    )
  ));
```

Solution 3 - iii

A better compromise: WITH

```
WITH jd(id) AS (
   SELECT p.id FROM sn.people p WHERE p.lastname = ’Doe’ AND p.firstname = ’John’
), friendsofjohndoe(id) AS (
   SELECT DISTINCT p.id FROM sn.people p
   WHERE p.id IN (
      SELECT f.person1 FROM sn.friend f WHERE f.person2 IN (SELECT id FROM jd)
   ) OR p.id IN (
      SELECT f.person2 FROM sn.friend f WHERE f.person1 IN (SELECT id FROM jd)
   )
)
SELECT p.lastname, p.firstname FROM sn.people p
JOIN friendsofjohndoe fojd ON p.id = fojd.id;
```

Example 4

- For each *person* who has been the destination of a recommendation at least once, list all recommending agents' username, along with their number of recommendations
- Same as above, but limit only to recommendations by other *people*
- Advanced level: Same as above, but limit only to recommendation by their *friends*
- SQL Grandmaster level: for each *person*, find a single *person*, who is not their friend, and is most recommended by their friends

Relevant tables:

sn.agent	(id, username)
sn.people	(id, agentid(agent.id), lastname, firstname, dob)
sn.recommend	(agentid(agent.id), objectid(agent.id), destid(agent.id))
sn.friend	(person1(people.id), person2(people.id), timestamp)

Imperial College London

Solution 4 - i

- Let’s unpack the question: ’for each person [...], list all [...] username’

Solution 4 - i

- Let’s unpack the question: ’for each person [...], list all [...] username’
- This means generating a pairs (person, username), linking the person and the recommender, something like:

```
SELECT p.lastname, p.firstname, a.username
FROM sn.people p
JOIN sn.agent a ON ???
WHERE ???
```

Solution 4 - i

- Let’s unpack the question: ’for each person [...], list all [...] username’
- This means generating a pairs (person, username), linking the person and the recommender, something like:

```
SELECT p.lastname, p.firstname, a.username
FROM sn.people p
JOIN sn.agent a ON ???
WHERE ???
```

- ’[...] along with their number of recommendations’:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.agent a ON ???
JOIN sn.recommend r ON ???
WHERE ???
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

Solution 4 - ii

- All that remains is the joining to perform. . . This is driven by the *recommendations*, with the person being the recipient and the agent the sender:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
WHERE ???
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

Solution 4 - ii

- All that remains is the joining to perform...This is driven by the *recommendations*, with the person being the recipient and the agent the sender:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
WHERE ???
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

- Actually, no filtering (WHERE) is needed! So we get the final query:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

Solution 4 - iii

- Same as above, but limit only to recommendations by other *people*

Solution 4 - iii

- Same as above, but limit only to recommendations by other *people*
- Option 1: simply add a filter:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
WHERE a.id IN (SELECT agentid FROM sn.people)
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

Solution 4 - iii

- Same as above, but limit only to recommendations by other *people*
- Option 1: simply add a filter:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
WHERE a.id IN (SELECT agentid FROM sn.people)
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

- Option 2: join

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
JOIN sn.people q ON a.id = q.agentid
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

Solution 4 - iv

- Advanced level: same as above, but limit to recommendations by that person’s *friends*

Solution 4 - iv

- Advanced level: same as above, but limit to recommendations by that person’s *friends*
- Need to join in the ***friend***(person, person) many-to-many relationship…

Solution 4 - iv

- Advanced level: same as above, but limit to recommendations by that person's *friends*
- Need to join in the ***friend***(person, person) many-to-many relationship. . .
- Using the second option from above, that is actually straightforward:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
JOIN sn.people q ON a.id = q.agentid
JOIN sn.friend f ON f.person1 = p.id
WHERE f.person2 = q.id
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

- Or is it. . . ?

Solution 4 - iv

- Advanced level: same as above, but limit to recommendations by that person’s *friends*
- Need to join in the ***friend***(person, person) many-to-many relationship. . .
- Using the second option from above, that is actually straightforward:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
JOIN sn.people q ON a.id = q.agentid
JOIN sn.friend f ON f.person1 = p.id
WHERE f.person2 = q.id
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

- Or is it. . . ?
- Try inverting ’person1’ and ’person2’ in the query above. . .
- The non-oriented relationship raises its ugly head again!

Solution 4 - v

- Doing a big WITH extravaganza as in previous example is possible...

Solution 4 - v

- Doing a big WITH extravaganza as in previous example is possible...
- But there is another solution for these kind of problems: tuple comparison:

```
SELECT p.lastname, p.firstname, a.username, COUNT(r.id)
FROM sn.people p
JOIN sn.recommend r ON r.destid = p.agentid
JOIN sn.agent a ON r.agentid = a.id
JOIN sn.people q ON a.id = q.agentid
WHERE (
   (p.id, q.id) IN (SELECT f.person1, f.person2 FROM sn.friend f) OR
   (p.id, q.id) IN (SELECT f.person2, f.person1 FROM sn.friend f)
)
GROUP BY p.id, p.lastname, p.firstname, a.id, a.username
```

Solution 4 - vi

- Grandmaster level: for each *person*, find the *person*, who is not their friend, and is most recommended by their friends

Solution 4 - vi

- Grandmaster level: for each *person*, find the *person*, who is not their friend, and is most recommended by their friends
- Proceeding methodically: WHAT are we trying to find ?

```
SELECT p.firstname, p.lastname, q.firstname, q.lastname
FROM sn.people p
JOIN sn.people q ON ???
WHERE ???
```

Solution 4 - vii

- Let’s make our life simpler first...

```
CREATE TABLE allfriends(p1, p2, i1, i2) AS (
   SELECT f.person1, f.person2, p.agentid, q.agentid
   FROM sn.friend f
   JOIN sn.people p ON p.id = f.person1
   JOIN sn.people q ON q.id = f.person2
);
INSERT INTO allfriends(p1, p2, i1, i2) (
   SELECT f.person2, f.person1, q.agentid, p.agentid
   FROM sn.friend f
   JOIN sn.people p ON p.id = f.person1
   JOIN sn.people q ON q.id = f.person2
);
SELECT p.firstname, p.lastname, q.firstname, q.lastname
FROM sn.people p, sn.people q
WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends)
```

Solution 4 - viii

- Let’s start putting in the recommendations

```
SELECT p.firstname person, p.lastname,
        q.firstname recommendation, q.lastname, COUNT(r.id)
FROM sn.people p, sn.people q
JOIN sn.recommend r ON r.objectid = q.agentid
WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends) AND
       (p.id, r.agentid) IN (SELECT p1, i2 FROM allfriends)
GROUP BY (p.firstname, p.lastname, q.firstname, q.lastname)
```

Solution 4 - viii

- Let’s start putting in the recommendations

```
SELECT p.firstname person, p.lastname,
        q.firstname recommendation, q.lastname, COUNT(r.id)
FROM sn.people p, sn.people q
JOIN sn.recommend r ON r.objectid = q.agentid
WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends) AND
       (p.id, r.agentid) IN (SELECT p1, i2 FROM allfriends)
GROUP BY (p.firstname, p.lastname, q.firstname, q.lastname)
```

- Looks good! But all these aggregations sound really tricky to put in now. . . It is time for a step back.

Solution 4 - ix

- There is an obvious FD from people’s id to their names, so let’s only get that
- We will be able to sort ourselves once we have all the good ids.

```
SELECT p.id person, q.id recommendation, COUNT(r.id) num
FROM sn.people p, sn.people q
JOIN sn.recommend r ON r.objectid = q.agentid
WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends) AND
        (p.id, r.agentid) IN (SELECT p1, i2 FROM allfriends)
GROUP BY (p.id, q.id)
```

Solution 4 - xi

- Actually, having numerical IDs, we can now easily get a representative of each number of recommendations:

```
SELECT person, MAX(recommendation), num FROM (
  SELECT p.id person, q.id recommendation, COUNT(r.id) num
  FROM sn.people p, sn.people q
  JOIN sn.recommend r ON r.objectid = q.agentid
  WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends) AND
        (p.id, r.agentid) IN (SELECT p1, i2 FROM allfriends)
  GROUP BY (p.id, q.id)
) AS tmp
GROUP BY person, num
```

- Just have to add something like:

```
WHERE num = maxnum
```

But how ?

Solution 4 - xii

- We want $\mathit{person}, \mathit{any}(\mathit{recommendation}), \mathit{num}$ and the corresponding $\mathit{person}, \mathit{max}(\mathit{num})\ldots$

Solution 4 - xii

- We want $person, any(recommendation), num$ and the corresponding $person, max(num) \ldots$
- We can generate both, and then JOIN them !

```
WITH tmp(person, rec, num) AS (
   SELECT p.id person, q.id recommendation, COUNT(r.id) num
   FROM sn.people p, sn.people q
   JOIN sn.recommend r ON r.objectid = q.agentid
   WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends) AND
            (p.id, r.agentid) IN (SELECT p1, i2 FROM allfriends)
   GROUP BY (p.id, q.id)
), r(p, q, num) AS (
   SELECT person, max(rec), num FROM tmp GROUP BY person, num
), m(p, maxnum) AS (
   SELECT person, max(num) FROM tmp GROUP BY person
)
SELECT r.p, r.q
FROM r
JOIN m ON r.p = m.p
WHERE r.num = m.maxnum
```

Solution 4 - xiii

It’s over now, I swear!

- Let’s finally put it back together:

```
WITH tmp(person, rec, num) AS (
   SELECT p.id person, q.id recommendation, COUNT(r.id) num
   FROM sn.people p, sn.people q
   JOIN sn.recommend r ON r.objectid = q.agentid
   WHERE (p.id, q.id) NOT IN (SELECT p1, p2 FROM allfriends) AND
            (p.id, r.agentid) IN (SELECT p1, i2 FROM allfriends)
   GROUP BY (p.id, q.id)
), r(p, q, num) AS (
   SELECT person, max(rec), num FROM tmp GROUP BY person, num
), m(p, maxnum) AS (
   SELECT person, max(num) FROM tmp GROUP BY person
), top_rec(p, q) AS (
   SELECT r.p, r.q FROM r JOIN m ON r.p = m.p WHERE r.num = m.maxnum
)
SELECT p.firstname, p.lastname, q.firstname rec_first, q.lastname rec_last
FROM sn.people p JOIN top_rec r ON r.p = p.id JOIN sn.people q ON r.q = q.id
ORDER BY p.lastname, p.firstname
```

Imperial College London

Conclusion

- You won’t have any monster like that on the exam
- But this should show you that ***SQL is very powerful*** (and quite rich)
- Several sites (most notably Stack Overflow) run on pretty much nothing more than SQL
- Train, experiment and build your intuition! These tutorials give you examples and syntax, but only practice makes perfect
- ***Use your DoC database***; most languages have fairly straightforward PostgreSQL bindings, which lets you use your DoC database to as a globally accessible store for any project you have
- Build your queries in steps - what do you need; what combination of sources will get you that; what then needs to be filtered. And finally, in what order do you want the results.

Questions ?