CSC 370

Activity Worksheet: Intro SQL

Mr. Yichun Zhao

Fall 2022

Notes

This worksheet provides a series of practice questions for writing and interpreting SQL queries. We will use a dataset extracted from an Anthropology research paper from 1977 that studied the bifurcation of a university's karate club's membership based on ideological and organisational disagreement. It is now a well-known dataset for social network analysis, called Zachary's karate club. I have converted it into a relational schema and set of tables below. I have constructed fictitious names for the club members, as the original dataset was anonymised. The graph is undirected, and I have represented only one canonical edge, i.e., the ordered pair (i, j), i < j. The "num_contexts" attributes refers to the number of other contexts outside karate club (e.g., at the bar or in classes together) in which that pair of members also socialised together.

In the first part of worksheet, you are given a SQL query and expected to describe in plain English the intent of the query and to show the output of running that query on those tables. In the second part of the worksheet, you are only given a plain English description of the query, and you should write it in SQL.

Schema

Member(id, name, faction, faction_strength, post_split_club)
Club(id, label)
Faction(id, label)
Relationship(member1, member2, num_contexts)

id	label
1	Mr. Hi
2	John

Table 1: Faction

id	label
1	Mr. Hi's
2	Officers'

Table 2: Club

id	name	faction	fs	psc	id	name	faction	fs	psc
1	Alice	1	strong	1	18	Werner	1	weak	1
2	Bob	1	strong	1	19	Xi	NULL	NULL	2
3	Carol	1	strong	1	20	Yuri	1	weak	1
4	Dave	1	strong	1	21	Zane	2	strong	2
5	Eve	1	strong	1	22	Antonio	1	weak	1
6	Farisha	1	strong	1	23	Babak	2	strong	2
7	Gregoire	1	strong	1	24	Claire	2	weak	2
8	Hamza	1	strong	1	25	Dierdre	2	weak	2
9	Ninad	2	weak	1	26	Einar	2	strong	2
10	Omar	NULL	NULL	2	27	Farouk	2	strong	2
11	Panagiotis	1	strong	1	28	Ghada	2	strong	2
12	Quinn	1	strong	1	29	Hiro	2	strong	2
13	Ravi	1	weak	1	30	Iniko	2	strong	2
14	Saalima	1	weak	1	31	Javier	2	strong	2
15	Tarik	2	strong	2	32	Kumar	2	strong	2
16	Ulysses	2	weak	2	33	Ludmila	2	strong	2
17	Vivienne	NULL	NULL	1	34	Manpreet	2	strong	2

Table 3: Member

m1	m2	nc	m1	m2	nc	m1	m2	nc	m1	m2	nc
1	2	4	1	3	5	1	4	3	1	5	3
1	6	3	1	7	3	1	8	2	1	9	2
1	11	2	1	12	3	1	13	2	1	14	3
1	18	2	1	20	2	1	22	2	1	32	2
2	3	6	2	4	3	2	8	4	2	14	5
2	18	1	2	20	2	2	22	2	2	31	2
3	4	3	3	8	4	3	9	5	3	10	1
3	14	3	3	28	2	3	29	2	3	33	3
4	8	3	4	13	3	4	14	3	5	7	2
5	11	3	6	7	5	6	11	3	6	17	3
7	17	3	9	31	3	9	33	4	9	34	3
10	34	2	14	34	3	15	33	3	15	34	2
16	33	3	16	34	4	19	33	1	19	34	2
20	34	1	21	33	3	21	34	1	23	33	2
24	26	5	24	28	4	24	30	2	24	33	5
24	34	4	25	26	2	25	28	3	25	32	2
26	32	7	27	30	4	27	34	2	28	34	4
29	32	2	29	34	2	30	33	3	30	34	2
31	33	3	31	34	3	32	33	4	32	34	4
33	34	5									

Table 4: Relationship

Questions

1. Interpreting SQL Queries

```
SELECT ‘name’
FROM ‘Member’
    JOIN ‘Faction’ ON (‘faction’ = ‘Faction’.‘id’)
WHERE ‘faction_strength’ = ‘weak’
    AND ‘label’ = ‘Mr. Hi’;
```

Solution: The methodical approach is to:

1. perform all the joins to get the set of candidate tuples.
2. remove all tuples that don’t match all the selection predicates.
3. remove attributes that are not in the projection; don’t forget to include when you show the result!

Then one should end up with:

name
Ravi
Saalima
Werner
Yuri
Antonio

Table 5: Result

This retrieves the names of everyone who is weakly associated with Mr. Hi’s faction.

2. Interpreting SQL Queries

```
SELECT ‘M2’.‘name’
FROM ‘Member’ AS ‘M1’
   , ‘Member’ AS ‘M2’
WHERE ‘M1’.‘name’ = ‘Einar’
  AND ‘M1’.‘faction’ = ‘M2’.‘faction’;
```

Solution:

3. Interpreting SQL Queries

```
SELECT `M1`.`name`, `M2`.`name`
FROM `Member` AS `M1`
    JOIN `Member` AS `M2`
        ON (`M1`.`faction` = `M2`.`faction`)
WHERE `M1`.`post_split_club` <> `M2`.`post_split_club`;
```

Solution:

4. Interpreting SQL Queries

```
SELECT 'M1'.'name', 'M2'.'name'
FROM 'Member' AS 'M1'
    JOIN 'Relationship'
        ON ('M1'.'id' = 'member1')
    JOIN 'Member' AS 'M2'
        ON ('M2'.'id' = 'member2')
WHERE 'num_contexts' > 5;
```

Solution:

5. Interpreting SQL Queries

```
SELECT ‘M2’.‘name’
FROM ‘Member’ AS ‘M1’
    JOIN ‘Relationship’
        ON (‘M1’.‘id’ = ‘member1’)
    JOIN ‘Member’ AS ‘M2’
        ON (‘M2’.‘id’ = ‘member2’)
WHERE ‘M1’.‘name’ LIKE ‘f%’;
```

Solution:

6. Writing SQL Queries: Retrieve the names of every unique pair of club members who are weakly associated to the same faction.

   Solution:

7. Writing SQL Queries: Retrieve the names of everyone who is in the same post-split club as Werner.

   Solution:

8. Writing SQL Queries: Retrieve the names of everyone with whom Manpreet shares at least two other social contexts outside the karate club. (You can exploit that she has the highest id).

   Solution:

9. Writing SQL Queries: Retrieve all triplets of names of club members in the same faction who are all in at least three external social contexts with each other.

   Solution:

Solutions

Question 1

The methodical approach is to:

1. perform all the joins to get the set of candidate tuples.
2. remove all tuples that don't match all the selection predicates.
3. remove attributes that are not in the projection; don't forget to include when you show the result!

Then one should end up with:

name
Ravi
Saalima
Werner
Yuri
Antonio

Table 6: Result

This retrieves the names of everyone who is weakly associated with Mr. Hi's faction.

Question 2

Again, let's go through this slowly. We take the opposite approach as to when constructing a query.

1. First, we join together all the tables. As we have a comma, not a JOIN clause, this is a cross product between Member and itself. So we will get each member paired with each other member (including a copy of itself), leading to 34(34) = 1156 tuples, each with 10 attributes.
2. Next, we can apply the selection predicates. This will remove all the tuples where the name on the left hand side is not Einar and the faction on the left hand side and right hand side do not match. In other words, we are only left with the pairs of members that are in the same faction as Einar and were the left hand side is Einar. This leaves sixteen tuples, still defined over ten attributes.
3. Finally, we can apply the projection operator and discard all attributes except the name on the right-hand side. Note that duplicates will not necessarily be removed like they would be in datalog or relational algebra.

The final result is thus as below:

name
Ninad
Tarik
Ulysses
Zane
Babak
Claire
Dierdre
Einar
Farouk
Ghada
Hiro
Iniko
Javier
Kumar
Ludmila
Manpreet

Table 7: Result

This query retrieves the names of everyone in the same faction as 'Einar'.

Question 3

Again, let's go through this slowly. We take the opposite approach as to when constructing a query.

1. First, we join together all the tables. This time we have a JOIN clause, again between Member and itself. The join key is 'factions'; so we will get each member paired with each other member in the same (non-NULL) faction (including a copy of itself), leading to 15(15) + 16(16) = 481 tuples, each with 10 attributes.
2. Next, we can apply the selection predicates. This will remove all the tuples where the post_split_club is the same. This leaves only 2(15) tuples, namely the tuple with Ninad on one side and everyone else in faction 2 on the other. Ninad is no longer paired with himself because of the inequality on post_split_club.
3. Finally, we can apply the projection operator and discard all attributes except the names on each side.

The final result is thus as below:

This query retrieves the ordered pairs of names of everyone in the same faction but different clubs, i.e., the "defectors".

M1.name	M2.name
Ninad	Tarik
Ninad	Ulysses
Ninad	Zane
Ninad	Babak
Ninad	Claire
Ninad	Dierdre
Ninad	Einar
Ninad	Farouk
Ninad	Ghada
Ninad	Hiro
Ninad	Iniko
Ninad	Javier
Ninad	Kumar
Ninad	Ludmila
Ninad	Manpreet
Tarik	Ninad
Ulysses	Ninad
Zane	Ninad
Babak	Ninad
Claire	Ninad
Dierdre	Ninad
Einar	Ninad
Farouk	Ninad
Ghada	Ninad
Hiro	Ninad
Iniko	Ninad
Javier	Ninad
Kumar	Ninad
Ludmila	Ninad
Manpreet	Ninad

Table 8: Result

Question 4

Again, let's go through this slowly. We take the opposite approach as to when constructing a query.

1. First, we join together all the tables. This time we JOIN three tables, including Member to itself. The join key is both of the foreign key sin Relationship to the primary keys of the two respective instances of Member; so we will get one join tuple for each tuple of Relationship, i.e., 77 tuples. These will have 5 + 3 + 5 = 13 attributes.
2. Next, we can apply the selection predicates. This will discard all tuples in which the num_contexts attribute has a value of 5 or less, leaving just 2 tuples.
3. Finally, we can apply the projection operator and discard all attributes except the names on each side.

The final result is thus as below:

M1.name	M2.name
Bob	Carol
Einar	Kumar

Table 9: Result

This query retrieves the names of unordered pairs of club members who have at least six social connections outside of karate club.

Question 5

Again, let's go through this slowly. We take the opposite approach as to when constructing a query.

1. First, we join together all the tables. This is the same join as in the previous question; so we will get one join tuple for each tuple of Relationship, i.e., 77 tuples, with 5 + 3 + 5 = 13 attributes.
2. Next, we can apply the selection predicates. This will discard all tuples in which the name of the first person does not match the case-insensitive pattern starting with an 'f'. Farisha (id 6) and Farouk (id 27) are the only names that match this pattern. There are only 3 + 2 = 5 tuples in the join result that match.
3. Finally, we can apply the projection operator and discard all attributes except the second name.

The final result is thus as below:

M2.name
Gregoire
Panagiotis
Vivienne
Iniko
Manpreet

Table 10: Result

This query retrieves the names of connections with a higher id of anyone who has a name starting with 'F'.

Question 6

To answer this systematically, we take the same approach to defining the logic as we did with Datalog:

1. We identify the attributes that are needed for the final projection or to satisfy selection predicates: here we need two club member names, a faction, and a faction strength.
2. We identify the tables that those attributes are found in: here we need two instances of Member.
3. We determine how to join those tables together in a way consistent with the question: here we have a self-join with a requirement for unique pairs, so we will join on equality of faction, equality of faction strength, and a strict inequality on id.
4. We determine any remaining selection predicates that are needed: here we need to select on faction_strength being ‘weak’.
5. Finally, we project onto the final set of attributes: here, it is just the names.

```
SELECT `M1`.`name`, `M2`.`name`
FROM `Member` AS `M1`
    JOIN `Member` AS `M2`
        ON (`M1`.`faction` = `M2`.`faction`
          AND `M1`.`faction_strength` = `M2`.`faction_strength`
          AND `M1`.`id` < `M2`.`id`)
WHERE `M1`.`faction_strength` = 'weak';
```

Question 7

To answer this systematically, we take the same approach to defining the logic as we did with Datalog:

1. attributes: member name, post_split_club for two copies of Member.
2. tables: two instances of Member.
3. join: self-join Member on post_split_club and inequality on id.
4. selection: the first Member's name should be 'Werner'.
5. Finally, we project onto the final set of attributes: here, it is just the second name (i.e., not 'Werner').

```
SELECT `M2`.`name`
FROM `Member` AS `M1`
    JOIN `Member` AS `M2`
        ON (`M1`.`post_split_club` = `M2`.`post_split_club`
          AND `M1`.`id` <> `M2`.`id`)
WHERE `M1`.`name` = 'Werner';
```

Question 8

To answer this systematically, we take the same approach to defining the logic as we did with Datalog:

1. attributes: member name for two copies of Member, also num_contexts.
2. tables: two instances of Member and Relationship.
3. join: We can join each instance of member using the foreign keys member1 and member2 in Relationship. We will exploit the id here and assume that Manpreet is member2 in a connection.
4. selection: the second Member's name should be 'Manpreet' and num_contexts should be >= 2.
5. Finally, we project onto the final set of attributes: here, it is just the first name (i.e., not 'Manpreet').

```
SELECT `M1`.`name`
FROM `Member` AS `M1`
    JOIN `Relationship`
        ON (`M1`.`id` = `member1`)
    JOIN `Member` AS `M2`
        ON (`M2`.`id` = `member2`)
WHERE `M2`.`name` = `Manpreet`
    AND `num_contexts` >= 2;
```

Question 9

To answer this systematically, we take the same approach to defining the logic as we did with Datalog:

1. attributes: member name for three copies of Member, also num_contexts for three copies of Relationship.
2. tables: three instances of Member and three instances of Relationship.
3. join: We can join each instance of member using the foreign keys member1 and member2 in Relationship. We will do this three times to get all three relationship context counts for each pair of the three club members.
4. selection: num_contexts should be >= 3 for all three relationships.
5. Finally, we project onto the final set of attributes: here, it is just the three member names.

```
SELECT 'M1'.'name', 'M2'.'name', 'M3'.'name'
FROM 'Member' AS 'M1'
    JOIN 'Relationship' AS 'R1'
        ON ('M1'.'id' = 'R1'.'member1')
    JOIN 'Member' AS 'M2'
        ON ('M2'.'id' = 'R1'.'member2')
    JOIN 'Relationship' AS 'R2'
        ON ('M2'.'id' = 'R2'.'member1')
    JOIN 'Member' AS 'M3'
        ON ('M3'.'id' = 'R2'.'member2')
    JOIN 'Relationship' AS 'R3'
        ON ('M1'.'id' = 'R3'.'member1'
            AND 'M3'.'id' = 'R3'.'member2')
WHERE 'R1'.'num_contexts' >= 3
    AND 'R2'.'num_contexts' >= 3
    AND 'R3'.'num_contexts' >= 3;
```