CSC 370

Activity Worksheet: Grouping & Sorting

Mr. Yichun Zhao

Fall 2022

Notes

This worksheet provides a series of practice questions for writing and interpreting SQL queries that plausibly involve sub-queries. We will continue to use the Zachary's karate club dataset from the previous worksheets, which has been copied again here for convenience.

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 and Simplifying SQL Queries

```
SELECT 'label' AS 'club', COUNT(*)
FROM 'Member'
    JOIN 'Club'
        ON ('post_split_club' = 'Club'.'id')
GROUP BY 'label';
```

Solution:

2. Interpreting and Simplifying SQL Queries

```
SELECT 'Club'.'label' AS 'club', COUNT(*)
FROM 'Member'
    JOIN 'Club'
        ON ('post_split_club' = 'Club'.'id')
    Join 'Faction'
        ON ('faction' = 'Faction'.'id')
WHERE ('Club'.'label' LIKE '%Hi%' AND 'Faction'.'label' NOT LIKE
    '%Hi%' )
    OR ('Club'.'label' NOT LIKE '%Hi%' AND 'Faction'.'label' LIKE
        '%Hi%' )
GROUP BY 'Club'.'label';
```

Solution:

3. Interpreting and Simplifying SQL Queries

```
SELECT 'name'
FROM 'Member' AS 'M1'
WHERE EXISTS (
      SELECT *
      FROM 'Relationship'
          JOIN 'Member' AS 'M2'
               ON ('id' = 'member1')
      WHERE 'M1'.'id' = 'member2'
           AND 'name' = 'Javier');
```

Solution:

4. Interpreting and Simplifying SQL Queries

```
SELECT 'name', 'label' AS 'faction', COUNT(*)
FROM 'Member' AS 'M1'
    JOIN (
        SELECT 'member1' AS 'u', 'member2' AS 'v'
        FROM 'Relationship'
            UNION ALL
        SELECT 'member2' AS 'u', 'member1' AS 'v'
        FROM 'Relationship')
    ON ('u' = 'id')
    JOIN 'Faction'
    ON ('faction' = 'Faction'.'id')
GROUP BY 'name'
HAVING COUNT(*) >= 5;
```

Solution:

5. Writing SQL Queries: For each club member, determine the maximum strength connection that they have.

   Solution:

6. Writing SQL Queries: For each club member, determine with whom they have the maximum strength connection of all their connections.

   Solution:

7. Writing SQL Queries: For each unique, ordered pair of clubs and factions, determine the total number of weak and strong faction associations, ordered by the sum of those.

   Solution:

8. Writing SQL Queries: A triad is a 3-clique or 3-cycle in a graph. For each club member, determine the number of unique triads they are in.

   Solution:

9. Writing SQL Queries: Triadic closure refers to a real-world phenomenon in social networks where the friends of your friends are usually also your friends. This can be interpreted as a large percentage of connected vertex triplets that you are in are triads. Determine for each person:

   - The number of connected vertex triplets that include them in the Relationships graph.
   - The number of triads that include them.
   - The ratio of triads to vertex triplets above (i.e., the triadic closure ratio), which you should use as the sort key for the query result.

   Solution:

Solutions

Question 1

Logically, we apply grouping, aggregation and having clauses prior to projection, and then apply sorting last:

1. Here we join club to member, using the foreign key in member. This will yield as many tuples as members with non-NULL clubs and the sum of their attributes, i.e., 34 tuples and 7 attributes.
2. There is no selection in this query.
3. Now we group by 'label'; i.e., we create one output tuple for each unique 'label' (or "equivalence class") in our intermediate solution. There are only two of these. We assign each intermediate tuple to one of those groups so that each group has a set of tuples (17 each).
4. We project onto 'label' (renamed as 'club') and use the count aggregation function on its group: this simply returns the group size.

club	COUNT(*)
Mr. Hi's	17
Officers'	17

Table 5: Result

This query retrieves the name of each club and the number of members in it.

Question 2

Logically, we apply grouping, aggregation and having clauses prior to projection, and then apply sorting last:

1. Here we join club and faction to member, using the foreign keys in member. This will again yield as many tuples as members with non-NULL clubs and the sum of their attributes, i.e., 31 tuples and 9 attributes.
2. We filter those results to only those tuples in which the club and faction labels for the tuple have exactly one appearance of the string '%Hi%'. This leaves only one tuple: {(9, 'Ninad', 2, 'weak', 1, 1, 'Mr. Hi's, 2, 'John')}.
3. Now we group by 'Club'.'label'; i.e., we create one output tuple for each unique 'label' (or "equivalence class") in our intermediate solution. This time, there is only one distinct club label. We assign each (i.e., the only) intermediate tuple to that group.
4. We project onto 'Club'.'label' (renamed as 'club') and use the count aggregation function on its group: this simply returns the group size.

club	COUNT(*)
Mr. Hi's	1

Table 6: Result

This query retrieves the name of each club and the number of members that defected to it from the other faction.

Question 3

Logically, we apply grouping, aggregation and having clauses prior to projection, and then apply sorting last:

1. Here we join two instances of member to relationship, using its two foreign keys to member. Finally, we join faction to the first instance of member, using member's foreign key to faction. This will yield as many tuples as relationships that involve a first member with a non-NULL faction and the sum of their attributes, i.e., 74 tuples and 15 attributes.
2. We filter out tuples where the relationship is between two members from the same faction. Only 9 tuples remain.
3. Now we group the remaining tuples by the id of the first instance of member; i.e., we create one output tuple for each unique tuple from Member that appears in the first instance in our intermediate solution. There are only five of these. We assign each intermediate tuple to one of those groups so that each group has a set of tuples.
4. We apply a second filter to groups where we discard groups that don't have at least 2 tuples assigned to them. This discards the groups for member id's 2, 14, and 20, leaving only those for id's 1 and 3 left.
5. We project onto the group's key ('M1'.'id'), 'M1'.'name' (which is functionally determined by the group's key), the label of the first member's faction (which is also functionally determined by the group's key), and finally use the count aggregation function on its group: this simply returns the group size.
6. Finally, we sort the output in descending order of COUNT(*), i.e., the group size.

'M1'.'id'	'M1'.'name'	faction	COUNT(*)
3	Carol	Mr. Hi	4
1	Alice	Mr. Hi	2

Table 7: Result

This query retrieves the name and faction of each club member who has external social connections with at least two club members from the other faction, also reporting the tally as a descending sort key. It considers only directed edges.

Question 4

This problem involves a sub-query as a table in our join clause, so let's resolve that first:

1. This takes a union of two results, so it is effectively two queries. However, each is simpler than cases that we have already solved on this worksheet. It joins two instances of member to Relationship using the two foreign keys, then only retains the member id keys. The second query does the same but reverses the order of the ids. Each one will produce 77 tuples with 2 attributes. The UNION ALL clause will perform a bag union, so we end up with 154 tuples over 2 attributes in total. This corresponds to taking every edge (u, v) and adding edge (v, u). It enables us to no longer worry about whether one club member has a lower id than another one or not.

Logically, we apply grouping, aggregation and having clauses prior to projection, and then apply sorting last:

1. We join member to the first id of the sub-query result, leveraging that the sub-query has returned foreign keys to member from relationship. This will yield as many tuples as the sub-query result, i.e., 154 tuples, over 7 attributes. We also join to this the faction, which adds two attributes but loses the 6 tuples involving a member with a NULL faction. This leaves 148 tuples.
2. There are no selection predicates.
3. Now we group the remaining tuples by the name of the member. Recall that 3 members were discarded because they had NULL factions; so, there will be 31 groups.
4. We apply a second filter to groups where we discard groups that don't have at least 5 tuples assigned to them. This leaves only 10 groups.
5. Finally, we project onto the member's name, the label of the member's faction, and finally use the count aggregation function on its group: this simply returns the group size. Observe that faction is not determined by name, though! So this query will throw an error! Nonetheless, I have included below a result for a modification to the query that uses MAX('faction') to see how the logic plays out.

name	MAX('label')	COUNT(*)
Alice	Mr. Hi	16
Bob	Mr. Hi	9
Carol	Mr. Hi	10
Dave	Mr. Hi	6
Ninad	John	5
Saalima	Mr. Hi	5
Claire	John	5
Kumar	John	6
Ludmila	John	12
Manpreet	John	16

Table 8: Result

This query retrieves the name, faction, and number of connections for everyone with at least five connections. It processes edges bidirectionally.

Question 5

The "for each" phrasing is a strong indication that we will need grouping. The "maximum" term clearly requires aggregation. Let's apply the same general approach as before:

1. The attributes that we will need are distinct club member ids and maximum strength.
2. Both of these attributes come from Relationship. The only catch is that the member id could appear in either of two foreign keys: member1 or member2. We can resolve this using the UNION ALL approach from the previous question.
3. There are no selection predicates.
4. Since we want an aggregate value defined per club member, it makes sense to group by this attribute.
5. Finally, we project onto member id and take MAX(nc) from the set of tuples in that group.

```
SELECT 'member1', MAX('num_contexts')
FROM (SELECT 'member1', 'member2', 'num_contexts' FROM '
    Relationships'
    UNION ALL SELECT 'member2', 'member1', 'num_contexts' FROM '
        Relationships')
GROUP BY 'member1';
```

Question 6

This is a trickier one than the previous question, because we cannot say that there will be only one connection per club member for whom they have the maximum strength. Thus, we need to set this up as a sub-query with a sub-goal to retrieve the maximum strength connection (i.e., the previous question) and a main goal of retrieving all connections that club member has with that connection strength.

```
SELECT *
FROM Member AS 'M1'
    JOIN          (SELECT 'member1', 'member2', 'num_contexts' FROM '
        Relationship'
         UNION ALL SELECT 'member2', 'member1', 'num_contexts' FROM '
             Relationship')
         ON ('M1'.'id' = 'member1')
WHERE 'num_contexts' = (
    SELECT MAX('num_contexts')
    FROM 'Relationship'
    WHERE 'member1' = 'M1'.'id'
        OR 'member2' = 'M1'.'id');
```

Question 7

Again, we have the "for each" phrasing, this time of the pair (club, faction), and, like question (a), all the attributes in the projection are unique for that pair. Thus, we will use a group by here again:

1. We need club id and faction id, to identify the groups; we also need faction_strength for the aggregation attributes.
2. All of this information is available in the Member table.
3. However, our aggregations have filters in them (just "weak" or just "strong"). We could write these as correlated sub-queries in the select statement, but a cleaner choice is probably to write them as sub-queries in the FROM clause and use a join instead of a correlated sub-query.
4. We also need to order by an aggregation function.

```
SELECT club
      , faction
      , COALESCE('num_weak', 0) AS 'WeakAssocations' -- replaces NULL
        from L.O.J. with 0
      , COALESCE('num_strong', 0) AS 'StrongAssociations' -- replaces
        NULL from L.O.J. with 0
FROM 'Member'
    NATURAL LEFT OUTER JOIN (
        SELECT 'club', 'faction', COUNT(*) AS 'num_weak'
        FROM 'Member'
        WHERE 'faction_strength' = 'weak'
        GROUP BY 'club', 'faction') AS 'weaks'
    NATURAL LEFT OUTER JOIN (
        SELECT 'club', 'faction', COUNT(*) AS 'num_strong'
        FROM 'Member'
        WHERE 'faction_strength' = 'strong'
        GROUP BY 'club', 'faction') AS 'strongs'
GROUP BY 'club', 'faction'
ORDER BY 'WeakAssocations' + 'StrongAssociations';
```

Question 8

Note that these questions are becoming quite difficult. We are performing rudimentary social network analysis in a relational database.

In the previous worksheet, we calculated triads: all the filtration was done by join predicates. We need only group and count them here.

```
SELECT `M1`.`id`, COUNT(*) AS `NumTriads`
FROM `Member` AS `M1`
    JOIN `Relationship` AS `R1` ON (`M1`.`id` = `R1`.`member1` OR `M1
        `.`id` = `R1`.`member2`)
    JOIN `Member` AS `R2` ON (`M2`.`id` = `R1`.`member1` OR `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 (`M3`.`id` = `R3`.`member1` OR `M3
        `.`id` = `R3`.`member2`)
WHERE `M1`.`id` <> `M2`.`id`
  AND `M1`.`id` <> `M3`.`id`
  AND `M2`.`id` <  `M3`.`id`
  AND (`M1`.`id` = `R3`.`member1` OR `M1`.`id` = `R3`.`member2`)
GROUP BY `M1`.`id`;
```

Observe the trick of fixing the order of M2.id and M3.id to ensure that we do not return duplicate triads.

Question 9

This question combines part (d) and part (c). The projection is onto attributes that are determined from different sets of tuples. Also note that there are two different isomorphic paths of length two where a vertex could be: either on the endpoint or in the middle. For the first value here, we need to take into account both of those graph isomorphism. We will avoid outer joins on 'NumLength2Paths' to avoid div-by-zero errors.

```
SELECT 'id', 'NumLength2Paths', 'NumTriads', 'NumTriads' / '
   NumLength2Paths' AS 'Ratio'
FROM 'Member'
    NATURAL JOIN (
        SELECT 'M1'.'id', COUNT(*) AS 'NumTriads'
        FROM 'Member' AS 'M1'
            JOIN 'Relationship' AS 'R1' ON ('M1'.'id' = 'R1'.'member1
                ' OR 'M1'.'id' = 'R1'.'member2')
            JOIN 'Member' AS 'R2' ON ('M2'.'id' = 'R1'.'member1' OR '
               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 ('M3'.'id' = 'R3'.'member1
                ' OR 'M3'.'id' = 'R3'.'member2')
        WHERE 'M1'.'id' <> 'M2'.'id'
          AND 'M1'.'id' <> 'M3'.'id'
          AND 'M2'.'id' <  'M3'.'id'
          AND ('M1'.'id' = 'R3'.'member1' OR 'M1'.'id' = 'R3'.'
              member2')
        GROUP BY 'M1'.'id')
    NATURAL JOIN (
        SELECT 'M1'.'id', COUNT(*) AS 'NumLength2Paths'
        FROM 'Member' AS 'M1'
            JOIN 'Relationship' AS 'R1' ON ('M1'.'id' = 'R1'.'member1
                '
                                        OR 'M1'.'id' = 'R1'.'member2
                                    ')
            JOIN 'Member' AS 'R2' ON ('M2'.'id' = 'R1'.'member1'
                                    OR 'M2'.'id' = 'R1'.'member2')
            JOIN 'Relationship' AS 'R2' ON ('M2'.'id' = 'R2'.'member1
                '
                                        OR 'M1'.'id' = 'R2'.'member1'
                                        OR 'M1'.'id' = 'R2'.'member2'
                                    )
            JOIN 'Member' AS 'M3' ON ('M3'.'id' = 'R2'.'member1'
                                    OR 'M3'.'id' = 'R2'.'member2')
        WHERE 'M1'.'id' <> 'M2'.'id'
          AND 'M1'.'id' <> 'M3'.'id'
```

```
            AND `M2`.`id` <  `M3`.`id`
          GROUP BY `M1`.`id`)
ORDER BY `Ratio`;
```