CSC 370

Activity Worksheet: Sub-Queries In SQL

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 worksheet, 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, show the output of running that query on those tables, and then, if possible, provide a simpler SQL query that is functionally equivalent. 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 twice: once with a sub-query and, if possible, once without. (Do not use outer joins on this worksheet).

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

Solution:

2. Interpreting and Simplifying SQL Queries

```
SELECT ‘name’
FROM ‘Member’
WHERE ‘faction’ = (
     SELECT ‘id’
     FROM ‘Faction’
     WHERE ‘label’ <> ‘Ravi’ )
AND ‘faction_strength’ = ‘weak’;
```

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’
FROM ‘Member’ AS ‘M1’
WHERE NOT EXISTS (
    SELECT *
    FROM ‘Relationship’
        JOIN ‘Member’ AS ‘M2’
            ON (‘id’ = ‘member1’)
    WHERE ‘M1’.‘id’ = ‘member2’
        AND ‘name’ = ‘Javier’);
```

Solution:

5. Writing SQL Queries: Retrieve the names of every club member who has the same number of external social contexts with Alice as Bob does.

   Solution:

6. Writing SQL Queries: Retrieve the names of everyone who is in the same faction as Panagiotis, but does not have any external social interactions with him.

   Solution:

7. Writing SQL Queries: Retrieve the faction of everyone named Ghada.

   Solution:

8. Writing SQL Queries: Retrieve the names of everyone who has an external social relationship with somebody from the other faction.

   Solution:

Solutions

Question 1

The queries in this worksheet all have sub-goals. To work through this systematically, we will first determine the result of the sub-query and then determine the result of the outer query.

To answer the sub-query, we take the same approach as the previous worksheet, working "inside-out" compared to when we construct the queries:

1. join tables: There is only one table, Faction. It has 2 tuples and 2 attributes.
2. apply selection predicates: We filter to the 1 tuple that has label 'Mr. Hi'.
3. We apply the projection to get the simple result $\{(1)\}$. Observe that this can cast to a scalar since it is a 1×1 table.

To answer the outer query, we can take the same approach and "plug in" the result of the sub-query:

1. join tables: There is only one table, Member. It has 34 tuples and 5 attributes.
2. apply selection predicates: we only retain tuples that have a 'weak' faction strength and a faction equal to 1. There are six such tuples.
3. We apply the projection to retrieve only the name attribute from those six tuples.

Then one should end up with:

name
Ninad
Ravi
Saalima
Werner
Yuri
Antonio

Table 5: Result

Note: that this sub-query was unnecessary and could have been achieved with a JOIN. We can recognise that because we only used conjunctions (no negations nor disjunctions) and used the '=' arithmetic operator with the sub-query result. To convert it, we promote the tables in the sub-query FROM clause to the outer query FROM clause and the predicates in the sub-query WHERE clause to the outer query WHERE clause. The sub-query turns into an ON clause.

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

Question 2

To answer the sub-query, we take the same approach as the previous question, working "inside-out" compared to when we construct the queries:

1. join tables: There is only one table, as before, Faction. It has 2 tuples and 2 attributes.
2. apply selection predicates: We filter where the label is not 'Ravi', which 2 tuples match.
3. We apply the projection to get the result $\{(1), (2)\}$. Observe that this cannot cast to a scalar since it is a 2×1 table. Because this sub-query is meant to retrieve a scalar for the '=' operator, we will get a run-time error.

Summary: it is not a valid query!

Question 3

This is going to be a difficult query to answer, because we can see a reference to an outer query table, namely M1, inside the sub-query. This is what is known as a correlated sub-query, because the answer to the sub-query changes for every tuple evaluated in the outer query. Often, there is no better alternative than to iterate tuples of the outer query and recompute the sub-query on each iteration.

Here, we can cheat a bit to save ourselves effort. We can defer calculating the first selection predicate and calculate a static result for the sub-query as follows:

1. join: We join Relationship to Member using the first foreign key. This produces 77 tuples and 8 attributes.
2. selection: We perform only those selections not involving the outer query. Thus, we filter the tuples by the member referenced in the first foreign key having name 'Javier' (id 31). There are only two such tuples.
3. projection: there is none here. We retrieve *.

For convenience, let's materialise the result of partially evaluating the sub-query:

m1	m2	nc	id	name	faction	faction_strength	post_split_club
31	33	3	31	Javier	2	strong	2
31	34	3	31	Javier	2	strong	2

Table 6: Intermediate Result

Now, we can evaluate the outer query. This simply retrieves the name of all members whose id leads to a non-empty result for the sub-query (EXISTS keyword). Note that the predicate that we left out was to bind the outer query's id to column 2 of the sub-query. Thus, the result of the outer query will correspond exactly to those tuples that appear in column 2 of the sub-query, namely 33 and 34. We can cheat again and skip iterating the entire tuple and just retrieve the names for those two member ids:

name
Ludmila
Manpreet

Table 7: Result

The fact that we were able to "cheat" like this, rather than actually iterating all tuples are computing the correlated sub-query is a strong indicator that the query could be simplified. Let's again try promoting everything to the other query, using the insight that a JOIN behaves like an EXISTS: we will only join a tuple if there exists something to join to. Indeed, we can instead write:

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

In summary: this was a very poorly written SQL query that could have potentially had much worse performance if the query optimiser was unable to simplify it the way that we did. The actual query intent was fairly easy, just not expressed well.

Question 4

For this query, we can use the same approach as above and calculate an intermediate result in which we defer applying the selection predicate from the outer query. The only difference is that instead of projecting on ‘m2’ to produce the result, we need to take ‘Member’ minus the projection onto ‘m2’. This yields the opposite result, which is the name of every member except Ludmila and Manpreet.

Unfortunately, this is a negation of an existential predicate, and it cannot be easily simplified.

Question 5

Let's first identify the task's main goal and its sub-goal (in reverse order):

- Retrieve the number of external social contexts that Bob (id 2) has with Alice (id 1). Call that number x.
- Retrieve the names of every club member who has x external social contexts with Alice (id 1).

Let's write each query in turn, then compose them together:

```
-- inner query
SELECT `num_contexts`
FROM `Relationship` AS `R1`
WHERE `member1` = 1
    AND `member2` = 2;

-- outer query
SELECT `name`
FROM `Member` AS `M1`
    JOIN `Relationship` AS `R1`
        ON (`M1`.`id` = `member2`)
WHERE `member1` = 1
    AND `num_contexts` = @x;

-- compose them
SELECT `name`
FROM `Member` AS `M1`
    JOIN `Relationship` AS `R1`
        ON (`M1`.`id` = `member2`)
WHERE `member1` = 1
    AND `num_contexts` = (
        SELECT `num_contexts`
        FROM `Relationship` AS `R1`
        WHERE `member1` = 1
            AND `member2` = 2);
```

This is an example of a query with a complex independent condition. We saw in Datalog that we could simply promote the sub-query into a join. That gives us:

```
SELECT `M1`.`name`
FROM `Member` AS `M1`
    JOIN `Relationship` AS `R1`
        ON (`M1`.`id` = `member2`)
    JOIN `Relationship` AS `R2`
        ON (`R1`.`num_contexts` = `R2`.`num_contexts`
            AND `R1`.`member1` = `R2`.`member1`)
WHERE `R2`.`member1` = 1
```

```
AND ‘R2’.‘member2’= 2;
```

Question 6

Again, let us first form the sub-goal(s) and the main goal:

- Retrieve the faction that Panagiotis is in. Call it x.
- Retrieve everyone who has a social interaction with Panagiotis. Call that set N.
- Retrieve the names of everyone in faction x who is not in N.

We saw in the previous question how easy it is to combine the first sub-goal with the main task. So, let us instead solve only the second sub-goal with a sub-query:

```
--inner query
SELECT 'id'
FROM 'Relationship'
WHERE 'member1' = 11
    OR 'member2' = 11;

-- outer query
SELECT 'name'
FROM 'Member' as 'M1'
    JOIN 'Member' AS 'M2' ON ('M1'.'faction' = 'M2'.'faction')
WHERE 'M2'.'name' = 'Panagiotis'
    AND 'M1'.'id' NOT IN @N;

-- composing the two queries:
SELECT 'name'
FROM 'Member' AS 'M1'
    JOIN 'Member' AS 'M2' ON ('M1'.'faction' = 'M2'.'faction')
WHERE 'M2'.'name' = 'Panagiotis'
    AND 'M1'.'id' NOT IN (
        SELECT 'id'
        FROM 'Relationship'
        WHERE 'member1' = 11 OR 'member2' = 11);
```

We recognise that this is the pattern of negating an existential predicate; so, it is not likely that we can simplify it further.

Question 7

Let us first break this up into a sub-goal and a main goal:

- Retrieve everyone named Ghada. Call that list N.
- Retrieve the faction of everyone in N.

```
-- inner query
SELECT 'id'
FROM 'Member'
WHERE 'name' = 'Ghada';

-- outer query
SELECT 'M1'.'faction'
FROM 'Member' AS 'M1'
WHERE 'M1'.'id' IN @N;

-- compose them together
SELECT 'faction'
FROM 'Member'
WHERE 'id' IN (
    SELECT 'id'
    FROM 'Member'
    WHERE 'name' = 'Ghada');
```

This example was meant to show how we can take simple queries and write unnecessarily complex solutions. This could have been solved with the trivial query:

```
SELECT 'faction'
FROM 'Member'
WHERE 'name' = 'Ghada';
```

Question 8

Let us again break this up into a sub-goal and a main goal:

- Retrieve the ids of everyone from “the other faction”. Call that list N.
- Retrieve the names of everyone who has a relationship with somebody in N.

Observe that our sub-goal is correlated with the main goal; we cannot know what is “the other faction” without first knowing the tuple referred to in the main goal. To make this work, let’s first fix that faction as x and then change that when we compose the queries.

```
-- inner query
SELECT ‘id’
FROM ‘Member’
WHERE ‘faction’ <> @x;

-- outer query
SELECT DISTINCT ‘name’
FROM ‘Member’ AS ‘M1’
    JOIN ‘Relationship’ ON (‘M1’.‘id’ = ‘member1’ OR ‘M1’.‘id’ = ‘
        member2’)
WHERE ‘member1’ IN @N
    OR ‘member2’ IN @N;

-- composing them together, we make @x refer to the outer query and
   @N refer to the inner query
SELECT DISTINCT ‘name’
FROM ‘Member’ AS ‘M1’
    JOIN ‘Relationship’
        ON (‘M1’.‘id’ = ‘member1’ OR ‘M1’.‘id’ = ‘member2’)
WHERE ‘member1’ IN (
        SELECT ‘id’
        FROM ‘Member’ AS ‘M2’
        WHERE ‘M1’.‘faction’ <> ‘M2’.‘faction’)
    OR ‘member2’ IN (
        SELECT ‘id’
        FROM ‘Member’ AS ‘M2’
        WHERE ‘M1’.‘faction’ <> ‘M2’.‘faction’);
```

This looks like something that can be simplified as it fits the pattern of being two related but complex predicates. Let us promote the sub-query into a join:

```
SELECT DISTINCT ‘M1’.‘name’
FROM ‘Member’ AS ‘M1’
    JOIN ‘Relationship’
        ON (‘M1’.‘id’ = ‘member1’ OR ‘M1’.‘id’ = ‘member2’)
    JOIN ‘Member’ AS ‘M2’
```

```
        ON (`M2`.`id` = `member1` OR `M2`.`id` = `member2`)
WHERE `M1`.`faction` <> `M2`.`faction`;
```

This is clearly a vastly simpler solution than the prior one with a sub-query.