CS2102: Database Systems

Tutorial #5: *SQL (Part 3)*

Week 7 Guide

AY 2022/23 Sem 2

1 Discussions

The following questions are to be discussed during tutorial. All answers will be released with explanation.

1. **(Equivalence)** Consider the following relational schema:

```sql
CREATE TABLE R (
  a  INTEGER PRIMARY KEY,
  b  INTEGER,
  c  INTEGER
);

CREATE TABLE S (
  x  INTEGER PRIMARY KEY,
  y  INTEGER REFERENCES R(a)
);
```

Are these two queries *equivalent*?

(a) Q_1

```sql
SELECT   COUNT(c)
FROM     R
WHERE    a = 10;
```

(b) Q_2

```sql
SELECT   COUNT(c)
FROM     R
WHERE    a = 10
GROUP BY a;
```

Suggested Guide:

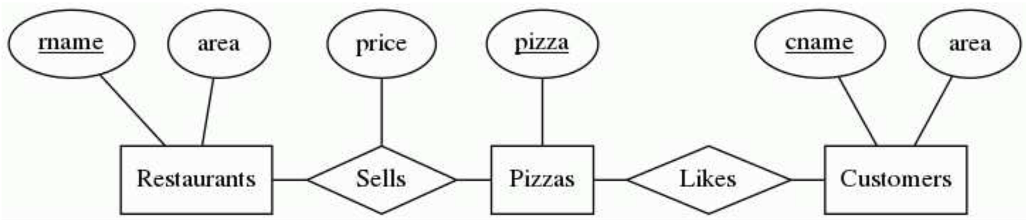
Q_1 and Q_2 are **NOT** equivalent queries. Consider the case where there is no record in `R` with `a = 10`.

(a) Q_1: The `WHERE` clause will evaluate to an *empty table* and the aggregate function `COUNT(c)` will evaluate to a *single-row* table with a *single-column* value of 0.

(b) Q_2: The `WHERE` clause will evaluate to an *empty table* so the `GROUP BY` clause will **NOT** produce any group. As such, the `COUNT(c)` will **NOT** be evaluated at all. As a result, the query result will be an *empty table*.

2. **(Un-Aggregate)** This question is based on the pizza database schema shown below.

Answer each of the following queries using SQL. For parts (a) to (e), remove duplicate records from all query results.



(a) Find the most expensive pizzas and the restaurants that sell them (*at the most expensive price*). Do **NOT** use any aggregate function in your answer.

(b) Find all restaurant pairs (R_1, R_2) such that the price of the most expensive pizza sold by R_1 is *higher* than than that of R_2. Exclude restaurant pairs where R_2 do not sell any pizza.

(c) For each restaurant that sels some pizza, find the restaurant name and the average price of its pizzas if its average price is higher than \$22. Do **NOT** use the `HAVING` clause in your answer.

(d) For each restaurant R that sells some pizza, let $totalPrice(R)$ denote the total price of all the pizzas sold by R. Find all pairs $(R, totalPrice(R))$ where $totalPrice(R)$ is *higher* than the average of $totalPrice()$ over all the restaurants.

(e) Find the customer pairs (C_1, C_2) such that $C_1 < C_2$ and they like *exactly* the same pizzas. Exclude customer pairs that do not like any pizza. Do **NOT** use the `EXCEPT` operator in your answer.

Suggested Guide:

(a) **Solution 1**:

```
SELECT pizza, rname
FROM   Sells
WHERE  price >= ALL (SELECT price FROM Sells);
```

Solution 2:

```
SELECT pizza, rname
FROM   Sells S1
WHERE  NOT EXISTS (
  SELECT 1
  FROM   Sells S2
  WHERE S2.price > S1.price
);
```

(b) **Solution 1**:

```sql
SELECT R1.rname, R2.rname
FROM   Restaurants R1, Restaurants R2
WHERE  (SELECT MAX(price) FROM Sells
        WHERE rname = R1.rname)
       >
       (SELECT MAX(price) FROM Sells
        WHERE rname = R2.rname);
```

Solution 2:

```sql
WITH RestMaxPrice AS (
  SELECT rname, (SELECT MAX(price) FROM Sells
                 WHERE rname = R.rname) as maxPrice
  FROM   Restaurants R
)
SELECT R1.rname, R2.rname
FROM   RestMaxPrice R1, RestMaxPrice R2
WHERE  R1.maxPrice > R2.maxPrice;
```

(c) A possible solution

```sql
WITH RestAvgPrice AS (
  SELECT   rname, AVG(price) as avgPrice
  FROM     Sells
  GROUP BY rname
)
SELECT *
FROM   RestAvgPrice
WHERE  avgPrice > 22;
```

(d) **Solution 1**:

```sql
WITH RestTotalPrice AS (
  SELECT   rname, SUM(price) as totalPrice
  FROM     Sells
  GROUP BY rname
)
SELECT rname, totalPrice
FROM   RestTotalPrice
WHERE  totalPrice > (SELECT AVG(totalPrice) FROM RestTotalPrice);
```

Solution 2:

```sql
SELECT   rname, SUM(price) as totalPrice
FROM     Sells S
GROUP BY rname
HAVING   SUM(price) > (SELECT SUM(price) / COUNT(DISTINCT rname) FROM
    Sells);
```

WRONG ANSWER 1: The following query is an **invalid** query

```sql
SELECT   rname, SUM(price) as totalPrice
FROM     Sells S
GROUP BY rname
HAVING   totalPrice > (SELECT SUM(price) / COUNT(DISTINCT rname) FROM
    Sells);
```

`totalPrice` is ***undefined*** in the `HAVING` clause as the `SELECT` clause is conceptually evaluated *after* the `HAVING` clause.

WRONG ANSWER 2: The following query produce the wrong result

```sql
SELECT   rname, SUM(price)
FROM     Sells
GROUP BY rname
HAVING   SUM(price) > SUM(price) / COUNT(*);
```

The query above is ***incorrect*** because both *SUM(price)* and *COUNT(*)* in the HAVING clause are computed w.r.t. a group.

(e) **Solution 1**: Customer C! and C2 like exactly the same pizzas if and only if (a) for every pizza that C1 likes, C2 also likes that pizza, and (b) for every pizza that C2 likes, C1 also likes that pizza. Conditions (a) holds if there does not exists any pizza that C1 likes that C2 does not like.

```sql
SELECT C1.cname, C2.cname
FROM   Customers C1, Customers C2
WHERE  C1.cname < C2.cname
  AND  EXISTS (SELECT 1 FROM Likes L WHERE L.cname = C1.cname)
  AND  NOT EXISTS (
         SELECT 1
         FROM   Likes L1
         WHERE  L1.cname = C2.cname
           AND  NOT EXISTS (
                  SELECT 1
                  FROM   Likes L2
                  WHERE  L2.cname = C2.cname
                    AND  L2.pizza = L1.pizza
                )
       )
  AND  NOT EXISTS (
         SELECT 1
         FROM   Likes L2
         WHERE  L2.cname = C2.cname
           AND  NOT EXISTS (
                  SELECT 1
                  FROM   Likes L1
                  WHERE  L1.cname = C1.cname
                    AND  L1.pizza = L2.pizza
                )
       );
```

Solution 2: This solution is based on the property that if customer C1 likes the set of pizzas $S1$ and customer C2 likes the set of pizzas $S2$, then $S1 = S2$ iff $|S1 \cap S2| = |S1| = |S2|$ where $|X|$ denote the cardinality of set X. Alternatively, you can also use the following equality $|S1 \cap S2| = |S1 \cup S2|$.

Here, we use CTE to solve the problem but a solution without CTE is possible.

```sql
WITH NumLike AS (
  SELECT   cname, COUNT(*) AS num FROM Likes L
  GROUP BY cname
), NumBothLike AS (
  SELECT   L1.cname AS cname1, L2.cname AS cname2, COUNT(*) AS num
  FROM     Likes L1, Likes L2
  WHERE    (L1.pizza = L2.pizza) AND (L1.cname < L2.cname)
  GROUP BY L1.cname, L2.cname
)
-- continue on the next page

```

```sql
SELECT cname1, cname2
FROM   NumBothLike B
WHERE  num = (SELECT num FROM NumLike N WHERE N.cname = B.cname1)
  AND  num = (SELECT num FROM NumLike N WHERE N.cname = B.cname2);
```

Solution 3: This solution is based on the property that if customers C1 and C2 like exactly the same pizzas, then for the subset of all tuples $S \subseteq$ `Likes` that are associated with C1 or C2, there must be exactly two tuples in S associated with each distinct pizza in S.

```sql
SELECT C1.cname, C2.cname
FROM   Customers C1, Customers C2
WHERE  C1.cname < C2.cname
  AND  C1.cname IN (SELECT cname FROM Likes)
  AND  NOT EXISTS (
         SELECT   1
         FROM     Likes
         WHERE    cname IN (C1.cname, C2.cname)
         GROUP BY pizza
         HAVING   COUNT(*) <> 2
       );
```

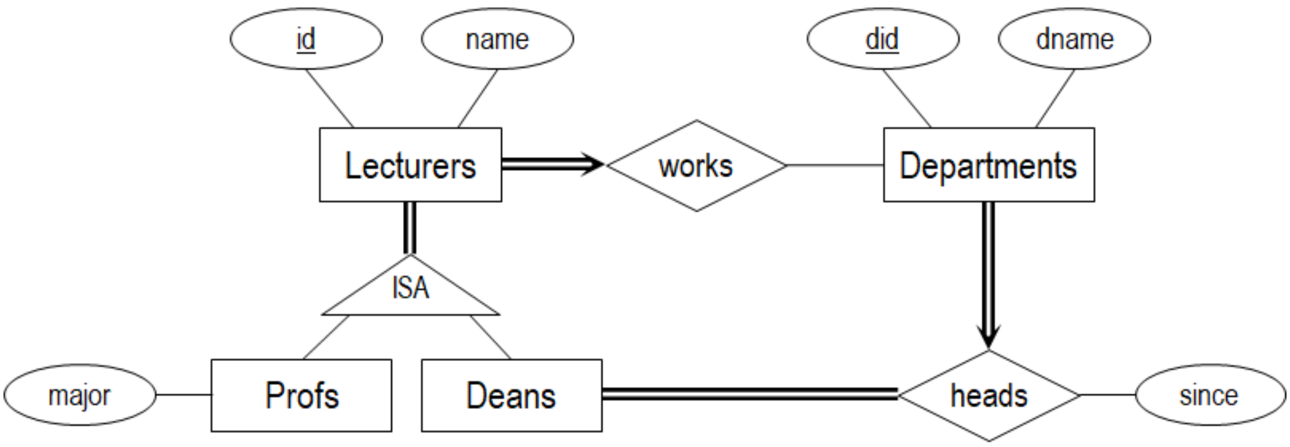
(f) **Solution**: This is simply done using the `UPDATE` statement with `CASE` expression.

```sql
UPDATE Sells S
SET    price = CASE (
                 SELECT area
                 FROM   Restaurants
                 WHERE  rname = S.rname
               )
                 WHEN 'Central' THEN price + 3
                 WHEN 'East' THEN price + 2
                 ELSE price + 1
               END;
```

2 Challenge

The answers to the following questions is given without explanation. Please discuss them on Canvas.

1. **(CTE)** Consider the following ER diagram.



We assume that we have the following schema:

- `Lecturers(id, name, did)`
- `Profs(id, major)`
- `Deans(id)`
- `Departments(did, dname, id)`

We say that a dean is ***important*** if the dean heads a department where either:

- There are at least 20 professors working in the department excluding the dean, OR
- There are at least 5 professors with different majors working in the department

Find all the ***non-important*** dean. **Hint:** You may use CTE to simplify the problem.

Suggested Guide:

Our solution does not actually use CTE. But you may use CTE to simplify the problem.

```sql
SELECT DISTINCT D.lid, D.name
FROM   Deans D, Departments H, Profs P
WHERE  D.id = H.id AND  H.did = P.did
  AND ((SELECT COUNT(DISTINCT P1.id)
        FROM   Deans D1, Departments H1, Profs P1
        WHERE  D1.id = H1.id AND H1.did =  P1.did
          AND  D1.id = D.id  AND D1.id <> P1.id
        GROUP BY D1.ld) > 20
        OR
       (SELECT COUNT(DISTINCT major)
        FROM   Deans D1, Departments H1, Profs P1
        WHERE  D1.id = H1.id AND H1.did = P1.did
          AND  D1.id = D.lid
        GROUP BY D1.id) > 5
      );
```

2. **(Universal Quantification)** Consider the following schema:

```sql
CREATE TABLE Students (
  matric  VARCHAR(9) PRIMARY KEY,
  sname   VARCHAR(50)
);
CREATE TABLE Projects (
  pid    VARCHAR(9) PRIMARY KEY,
  pname  VARCHAR(50)
);
CREATE TABLE Workings (
  pid     VARCHAR(9) REFERENCES Projects(pid),
  matric  VARCHAR(9) REFERENCES Students(matric),
  since   DATE,
  PRIMARY KEY(pid, matric)
);
CREATE TABLE Category (
  pid    VARCHAR(9) REFERENCES Projects(pid),
  cname  VARCHAR(9),
 PRIMARY KEY(pid, cname)
);
```

Find all pair of distinct projects' `pid (p1, p2)` such that the two projects have exactly the same set of categories.

Suggested Guide:

Here we use the cardinality method. The two sets must be identical so the intersection and the union must have the same size.

```
SELECT P1.pid, P2.pid
FROM   Projects P1, Projects P2
WHERE  P1.pid <> P2.pid
  AND  ( SELECT COUNT(*)
         FROM ( SELECT cname FROM Categories C0 WHERE P1.pid = C0.pid
                INTERSECT  -- alternatively UNION
                SELECT cname FROM Categories C0 WHERE P2.pid = C0.pid ) AS T1 )
       =
       ( SELECT COUNT(*)
         FROM ( SELECT cname FROM Categories C0 WHERE P1.pid = C0.pid
                UNION      -- alternatively INTERSECT
                SELECT cname FROM Categories C0 WHERE P2.pid = C0.pid ) AS T2 )
    ;
```