CS2102 Database Systems

Semester 1 2019/2020

Tutorial 05 (*Selected Answers*)

Quiz

Questions 1-6 uses the following database schema as Tutorial 03 Question 07. They are reproduced here for convenience.

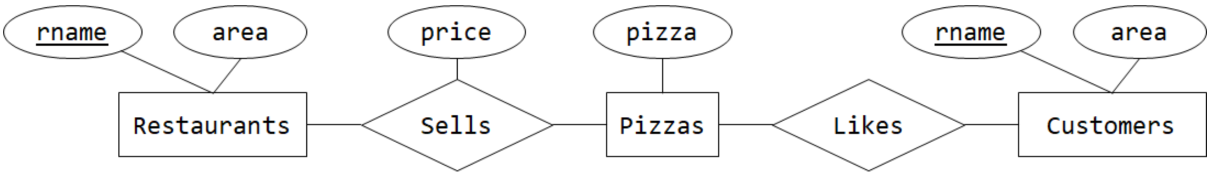
- **`Pizzas`** `(pizza)`
- **`Customers`** `(cname, area)`
- **`Restaurants`** `(rname, area)`
- **`Contains`** `(pizza, ingredient)`
- **`Sells`** `(rname, pizza, price)`
- **`Likes`** `(cname, pizza)`

`Pizzas` indicates all the pizzas of interest. **`Customers`** indicates the name and location of each customer. **`Restaurants`** indicates the name and location of each restaurant. **`Contains`** indicates the ingredients used in each pizza. **`Sells`** indicates the pizzas sold by restaurants and their prices. **`Likes`** indicates the pizzas that customers like.

The following are all the foreign key constraints on the database schema:

- `Contains.pizza` is a foreign key that refers to `Pizzas.pizza`
- `Sells.rname` is a foreign key that refers to `Restaurants.name`
- `Sells.pizza` is a foreign key that refers to `Pizzas.pizza`
- `Likes.cname` is a foreign key that refers to `Customers.pizza`
- `Likes.pizza` is a foreign key that refers to `Pizzas.pizza`

Furthermore, we provide the simplified ER diagram corresponding to the schema above minus `Contains`.



1. For each pizza, find the restaurant that sells it at the most expensive price. Your output should be `(rname, pizza, price)`. Exclude pizza that is not sold by any restaurants. *Can you do it without using any aggregate function?*

Solution:

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

Alternative solution:

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

2. Find all restaurant pairs `(rname1, rname2)` such that the price of the most expensive pizza sold by `rname1` is higher than that for `rname2`.

Solution:
```
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 );
```

Alternative solution:
```
WITH MaxPrice AS (
  SELECT rname, (SELECT MAX(price) FROM Sells WHERE rname = R.rname) AS maxP
  FROM   Restaurant R
)
SELECT R1.rname, R2.rname FROM MaxPrice R1 JOIN MaxPrice R2 ON R1.maxP > R2.maxP;
```

Alternative solution with restaurant not selling any pizza included with maxP = 0:
```
WITH MaxPrice AS (
  SELECT rname,
         COALESCE((SELECT MAX(price) FROM Sells WHERE rname = R.rname),0) AS maxP
  FROM   Restaurant R
)
SELECT R1.rname, R2.rname FROM MaxPrice R1 JOIN MaxPrice R2 ON R1.maxP > R2.maxP;
```

3. For each restaurant that sells some pizza, find the restaurant name and the average price of its pizza if its average price is higher than $22. *Can you do it without using `HAVING` clause in your answer?*

Solution:
```
WITH RestaurantAvgPrice AS
  ( SELECT rname, AVG(price) AS avgPrice FROM Sells GROUP BY rname )
SELECT * FROM RestaurantAvgPrice WHERE avgPrice > 22;
```

4. 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 `(rname, totalPrice(rname))` where `totalPrice(rname)` is higher than the average `totalPrice()` over all restaurants.

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

Alternative solution:
```
WITH TotalPrice AS (
  SELECT rname, SUM(price) AS totalP FROM Sells GROUP BY rname
)
SELECT rname, totalP FROM TotalPrice
WHERE  totalP > (Select AVG(totalP) FROM TotalPrice);
```

Invalid solution:

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

NOTE: `totalPrice` is `undefined` in the `HAVING` clause as the `SELECT` clause is evaluated after the `HAVING` clause.

Invalid solution:

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

NOTE: all three aggregate functions are computed w.r.t. a group

5. Find the customer pair `(cname1, cname2)` such that `cname1 < cname2` and they like exactly the same pizzas. Exclude customer pairs that do not like any pizza. *Can you do it without using `EXCEPT` operator?*

Solution:

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

NOTE: C1 and C2 likes *exactly* the same pizza if and only if (a) for all pizza that C1 likes, C2 likes the pizza, and (b) for all pizza that C2 likes, C1 also likes that pizza. Using universal quantification trick of $\neg\neg(\forall F) \equiv \neg(\exists\neg F)$.

Alternative solution:

```
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 JOIN Likes L2
             ON ( L1.pizza = L2.pizza ) AND ( L1.cname < L2.cname )
    GROUP BY L1.cname, L2.cname )
```

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

NOTE: Let S_1 be the set of pizza that C1 likes and S_2 be the set of pizza that C2 likes. Then $S_1 = S_2$ if and only if $|S_1 \cap S_2| = |S_1| = |S_2|$ where $|S|$ denotes the cardinality of the set S.

6. Write an SQL statement to increase the selling prices of pizzas as follows:
   i. Increase by $3 if the restaurant is located in 'Central'
   ii. Increase by $2 if the restaurant is located in 'East'
   iii. Otherwise, increase by $1

Solution:

```
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;
```

Tutorial Questions *[Discussion: 7(a), 7(b), 7(c), 7(d), 7(e)]*

7. Consider the following relational schema.

```
CREATE TABLE Students (
  sid      integer PRIMARY KEY,
  name     varchar(50) NOT NULL
);
CREATE TABLE Presenters (
  week     integer CHECK (week > 0),
  qnum     integer NOT NULL CHECK (qnum > 0),
  sid      integer REFERENCES Students (sid),
  PRIMARY KEY (week, sid)
);
```

The `Students` relation maintains information about students, and the `Presenters` relation maintains information about students who have presented solutions for tutorial questions. Specifically, a tuple `(w,q,s)` in `Presenters` relation means that the student with `sid s` presented tutorial questions `q` in week `w`. For each week, a student can present at most one question.

Assume that if the maximum week value in `Presenters` relation is `W`, then there is at least one record in `Presenters` for each week value from `{1, 2, ..., W}`, and the next tutorial class will be in Week `W+1`.

Write an SQL query for each of the following questions. Remove duplicate records from all results. Try not to use `DISTINCT` keywords unless necessary.

a) Find the `sid` of all students who have presented the most often.
b) Find all `sid` pairs `(s1,s2)` such that `s1 < s2` and both students have presented in the same week for at least 5 different weeks.
c) Find all students who did not present for any three consecutive weeks. For example, if Alice presented only twice so far in week 2 and 6, then Alice should be in the query result as she did not present in weeks 3, 4, and 5.

d) This question considers how to choose presenters for the next tutorial. Given a student with `sid s`, let `numQ(s)` denote the total number of questions that `s` has presented so far. Let `lastWk(s)` denote the most recent week number that `s` has presented. If `s` has not presented at all, then `numQ(s) = 0` and `lastWk(s) = 0`.

Given two students with sid s1 and s2, s1 has a higher priority than s2 if one of the following conditions hold:

```
  i.   numQ(s1) < numQ(s2)
 ii.   (numQ(s1) = numQ(s2)) and (lastWk(s1) < lastWk(s2))
iii.   (numQ(s1) = numQ(s2)) and (lastWk(s1) = lastWk(s2)) and (s1 < s2)
```

Find all sets of two students `S` to be presenters for the next tutorial such that none of the students in `(Students - S)` has higher priority than any of the students in `S`. The schema of your query result should be a pair of `sid (s1,s2)`, where `s1 < s2` and each tuple in the query result represents a possible value for `S`.

e) Let W denote the maximum week value in `Presenters`. We say that the `Presenters` table is *consistent* if it satisfies both the following conditions:

i. There must exist at least one tuple `t` in `Presenters` with `t.week = w` for each value of `w` in `{1, 2, ..., W}`

ii. For each value `i` in `{1, 2, ..., W}`, if the maximum `qnum` value for week `i` in `Presenters` is `Q`, then there must exist at least one tuple `t` in `Presenters` with `t.week = i` and `t.qnum = j` for each value of j in `{1, 2, ..., Q}`.

Write SQL query to output 0 if `Presenters` is consistent; and 1 otherwise.

Solution:

a) Solution #1:

```
SELECT   sid FROM Presenters
GROUP BY sid
HAVING   COUNT(*) >= ALL
       ( SELECT COUNT(*) FROM Presenters GROUP BY sid );
```

Solution #2:

```
WITH   numPres AS
     ( SELECT sid, COUNT(*) AS num FROM Presenters GROUP BY sid )
SELECT sid FROM numPres WHERE num = ( SELECT MAX(num) FROM numPres );
```

b) SQL Query:

```
SELECT   P1.sid, P2.sid
FROM     Presenters P1 JOIN Presenters P2
         ON P1.sid < P2.sid AND P1.week = P2.week
GROUP BY P1.sid, P2.sid
HAVING   COUNT(*) >= 5;
```

c) Solution #1: If we have the students who did not present in any particular week, this problem is quite trivial. So what if we compute those first!

```
WITH NotPresent AS (
  SELECT sid, week
  FROM   ( SELECT sid FROM Students ) AS st_tbl,
         ( SELECT DISTINCT week FROM Presenters ) AS wk_tbl
  EXCEPT
  SELECT sid, week
  FROM   Presenters
)
SELECT DISTINCT P1.sid FROM NotPresent P1, NotPresent P2, NotPresent P3
```

```
WHERE  P1.sid = P2.sid AND P2.sid = P3.sid
 AND   P2.week = P1.week + 1 AND P3.week = P1.week + 2;
```

<u>Solution #2:</u> Let `W` denote the maximum week number from `Presenters`. A student with identifier `s` is in the output if `W >= 3` and there exists three consecutive weeks, say starting from week `w` to week `w+2` such that `s` did not present in these three weeks. More precisely, `s` is in the output if `W >= 3` and there exists some `w` in `{1, ..., W-2}` where `{w, w+1, w+2}` $\cap\ \pi_{week}(\sigma_{sid=s}(Presenters)) = \emptyset$.

```
WITH Weeks AS (
     SELECT DISTINCT Week FROM Presenters
     WHERE  week + 2 <= ( SELECT MAX(week) FROM Presenters )
)
SELECT S.sid FROM Students S
WHERE  ( SELECT MAX(week) FROM Presenters ) >= 3
  AND  EXISTS (
         SELECT 1 FROM Weeks W
         WHERE  NOT EXISTS (
                  SELECT week FROM Weeks W2
                  WHERE  W2.week >= W.week
                    AND  W2.week <= W.week + 2
                  INTERSECT
                  SELECT week FROM Presenters P
                  WHERE  P.sid = S.sid
                )
       );
```

<u>Solution #3:</u> Case analysis; case (4) are saying that for any 2 presentations (p1,p2), the distance is >= 3 weeks and there are no other presentations (p3) in between for ALL (p1,p2). The rest are simply edge cases.

```
-- Case 1: not presented and W >= 3
SELECT sid FROM Students
WHERE  sid NOT IN (SELECT sid FROM Presenters)
  AND  (SELECT MAX(week) FROM Presenters) >= 3
UNION
SELECT sid FROM Presenters P1
WHERE
-- Case 2: first presentation is week 4 or after
( SELECT MIN(week) FROM Presenters WHERE sid = P1.sid ) >= 4
OR
-- Case 3: last presentation is week W-3 or before
( SELECT MAX(week) FROM Presenters ) -
( SELECT MAX(week) FROM Presenters WHERE sid = P1.sid ) >= 3
OR
-- Case 4: forall(P1,P2), not-exists(P3)
--     or, forall(P1), exists(P2), not-exists(P3)
EXISTS (
  SELECT 1 FROM Presenters P2
  WHERE  P1.sid = P2.sid AND P2.week
    AND  NOT EXISTS (
      SELECT 1 FROM Presenters P3
      WHERE  P3.sid = P1.sid AND P1.week < P3.week AND P3.week < P2.week
  )
);
```

<u>Wrong Answer #1:</u>

```
SELECT sid FROM Students
WHERE  ( SELECT MAX(week) FROM Presenters ) >= 3
EXCEPT
SELECT sid FROM Students S
WHERE  EXISTS (
         SELECT 1 FROM Presenters P1, Presenters P2, Presenters P3
         WHERE  P1.sid = S.sid AND P2.sid = S.sid AND P3.sid = S.sid
           AND  P2.week = P1.week + 1 AND P3.week = P1.week + 2
       );
```

The answer above is *incorrect* because a student who has presented for 3 consecutive weeks (e.g., in week 1, 2 and 3) but did not present in the next 3 consecutive weeks (e.g., in week 4, 5, and 6) would be incorrectly excluded from the output.

d) <u>Solution #1:</u>

```
WITH StudentInfo AS (
     SELECT   sid, COUNT(*) AS numQ, MAX(week) AS lastWk
     FROM     Presenters
     GROUP BY sid
     UNION
     SELECT   sid, 0, 0
     FROM     Students
     WHERE    sid NOT IN ( SELECT sid FROM Presenters )
),
     StudentSet AS (
     SELECT   sid FROM StudentInfo
     ORDER BY numQ, lastWk, sid
     LIMIT    2
)
SELECT MIN(sid) AS sid1, MAX(sid) AS sid2 FROM StudentSet;
```

<u>Solution #2:</u>

```
WITH StudentInfo AS (
     SELECT   P.sid, COUNT(*) AS numQ, MAX(week) AS lastWk
     FROM     Presenters P
     GROUP BY sid
     UNION
     SELECT   sid, 0, 0
     FROM     Students
     WHERE    sid NOT IN ( SELECT sid FROM Presenters )
)
SELECT S1.sid, S2.sid
FROM   StudentInfo S1 JOIN StudentInfo S2 ON S1.sid < S2.sid
WHERE  NOT EXISTS (
         SELECT 1 FROM StudentInfo S3
         WHERE  S3.sid <> S1.sid AND S3.sid <> S2.sid
           AND  (  ( S3.numQ < S1.numQ )
                OR ( S3.numQ < S2.numQ )
                OR ((S3.numQ = S1.numQ) AND (S3.lastWk < S1.lastWk))
                OR ((S3.numQ = S2.numQ) AND (S3.lastWk < S2.lastWk))
                OR ((S3.numQ = S1.numQ) AND (S3.lastWk = S1.lastWk)
                                        AND (S3.sid < S1.sid))
                OR ((S3.numQ = S2.numQ) AND (S3.lastWk = S2.lastWk)
                                        AND (S3.sid < S2.sid))
                )
```

```
);
```

e) <u>SQL Query:</u>

```
WITH Status AS (
     SELECT 1 AS num FROM Presenters
      WHERE ( SELECT MAX(week) FROM Presenters )
            <>
            ( SELECT COUNT(DISTINCT week) FROM Presenters )
         OR EXISTS (
              SELECT   1 FROM Presenters
              GROUP BY week
              HAVING   MAX(qnum) <> COUNT(DISTINCT qnum)
            )
)
SELECT COALESCE(MIN(num), 0) FROM Status;
```