CS2102 Database Systems

Semester 1 2019/2020

Tutorial 05

Quiz

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

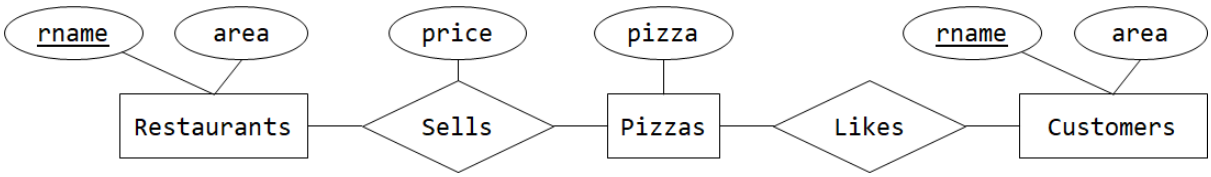
- **Pizzas** (pizza)
- **Customers** (cname, area)
- **Restaurants** (rname, area)
- **Contains** (pizza, ingredient)
- **Sells** (rname, pizza, price)
- **Likes** (cname, pizza)

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?*
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`.
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?*
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.
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?*
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

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.