CS2102: Database Systems

Tutorial #4: *SQL (Part 2)*

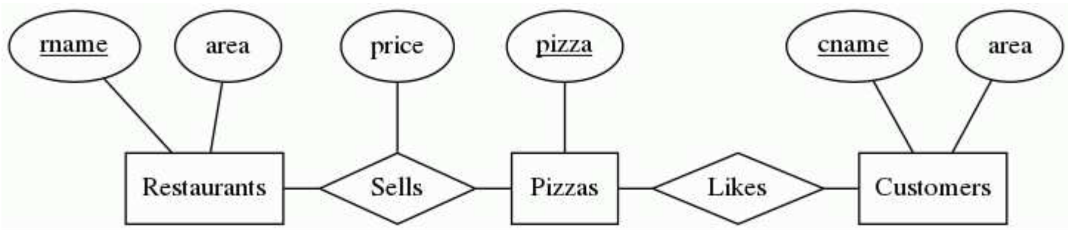
Week 6 Guide

AY 2022/23 Sem 2

1 Discussions

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

This tutorial discussion questions are based on the following pizza database schema. The ER diagram is shown below.



The ER diagram produces the following schemas:

Relation	Description
$Pizzas(\underline{pizza})$	All the pizzas of interest.
$Customers(\underline{cname}, area)$	The name and location of each customer.
$Restaurants(\underline{rname}, area)$	The name and location of each restaurant.
$Recipes(\underline{pizza, ingredients})$	The ingredients used in each pizza.
$Sells(\underline{rname, pizza}, price)$	Pizzas sold by restaurants and the prices.
$Likes(\underline{cname, pizza})$	Pizzas that customers like.

Additionally, we have the following foreign key constraints on the database schema:

- $(Recipes.pizza) \rightsquigarrow (Pizzas.pizza)$
- $(Sells.rname) \rightsquigarrow (Restaurants.rname)$
- $(Sells.pizza) \rightsquigarrow (Pizzas.pizza)$
- $(Likes.cname) \rightsquigarrow (Customers.cname)$
- $(Likes.pizza) \rightsquigarrow (Pizzas.pizza)$

1. **(Simple Query)** For each of the following queries, write an *equivalent* SQL query that does **not** use any subquery. Note that we do not consider set operation (*e.g., in* `Q1 UNION Q2`, *neither* `Q1` *nor* `Q2` *are considered subqueries*).

   (a) Query A

```sql
SELECT DISTINCT cname
FROM   Likes L
WHERE  EXISTS (
  SELECT 1
  FROM   Sells S
  WHERE  S.rname = 'Corleone Corner'
    AND  S.pizza = L.pizza
);
```

   (b) Query B

```sql
SELECT cname
FROM   Customers C
WHERE  NOT EXISTS (
  SELECT 1
  FROM   Likes L, Sells S
  WHERE  S.rname = 'Corleone Corner'
    AND  S.pizza = L.pizza
    AND  C.cname = L.cname
);
```

   (c) Query C

```sql
SELECT DISTINCT rname
FROM   Sells
WHERE  rname <> 'Corleone Corner'
  AND  price > ANY (
    SELECT price
    FROM   Sells
    WHERE  rname = 'Corleone Corner'
);
```

   (d) Query D

```sql
SELECT rname, pizza, price
FROM   Sells S
WHERE  price >= ALL (
  SELECT S2.price
  FROM   Sells S2
  WHERE  S2.rname = S.rname
    AND  S2.price IS NOT NULL
);
```

Suggested Guide:

A possible way to solve this kind of question is to translate the query into an equivalent English statement and then attempt to solve problem again without subquery.

(a) The query can be translated into English roughly as:

> *Find all distinct customer names that likes at least one pizza that is sold by Corleone Corner. Include only customers that likes at least one pizza. (Alternatively, exclude customers that do not like any pizza.)*

This can be solved without subquery by using join operation (*e.g., Cartesian product, inner join, etc*).

```sql
SELECT DISTINCT cname
FROM   Likes L, Sells S
WHERE  S.rname = 'Corleone Corner'
  AND  S.pizza = L.pizza;
```

(b) The query can be translated into English roughly as:

> *Find all distinct customer names that do not like any pizza that is sold by Corleone Corner. Include customers that do not like any pizza.*

This can be solved without subquery by decomposing the problem into two parts:

- Find all customers that likes at least one pizza (a).
- Remove these customers from the collection of all customers (*i.e., set difference*).

```sql
SELECT cname FROM Customers
EXCEPT
SELECT cname
FROM   Likes L, Sells S
WHERE  S.rname = 'Corleone Corner'
  AND  S.pizza = L.pizza;
```

(c) The query can be translated into English roughly as:

> *Find all restaurant names (besides Corleone Corner) that sells at least one pizza more expensive than any pizza sold by Corleone Corner.*

This can be solved without subquery by using join operation (*e.g., Cartesian product, inner join, etc*).

```sql
SELECT DISTINCT S.rname
FROM   Sells S, Sells S2
WHERE  S.rname <> 'Corleone Corner'
  AND  S2.rname = 'Corleone Corner'
  AND  S.price > S2.price;
```

(d) The query can be translated into English roughly as:

> *Find all distinct restaurant names (besides Corleone Corner), the pizza they sell, and the price of the pizza where the pizza is at least as expensive as the most expensive pizza sold by Corleone Corner.*

This can be solved without subquery by decomposing the problem into two parts:

- Find all triples (restaurant names, pizza, price) where the price is cheaper than any pizza sold by Corleone Corner.
- Remove these triples from the collection of triples (restaurant names, pizza, price).

```sql
SELECT rname, pizza, price
FROM   Sells
WHERE  price IS NOT NULL
EXCEPT
SELECT S.rname, S.pizza, S.price
FROM   Sells S, Sells S2
WHERE  S.rname = S2.rname
  AND  S.price < S2.price;
```

We need the condition `price IS NOT NULL` here because these triple will not be removed by the second query. When `price` is `NULL`, `S.price < S2.price` is `NULL` so the triple is not in the second query.

Similarly, in the original `ALL` subquery, the result of the subquery will be `NULL` so the triple containing `NULL` price is not going to be in the result.

2. **(SQL Query)** Write an SQL query to answer each of the following questions on the pizza database *without using aggregate functions*. Remove duplicate records from all query results.

   (a) Find pizzas that Moe likes but is not liked by Lisa.

   (b) Find pizzas that are sold by at most one restaurant in each area; exclude pizzas that are not sold by any restaurant.

   (c) Find all tuples (A, P, P_{min}) where P is a pizza that is available in area A (*i.e.*, there is some restaurant in area A selling pizza P) and P_{min} is the *lowest* price of P in area A.

Suggested Guide:

For each question, we will discuss some possible solutions (*remember, there can be multiple solutions*) and potentially some solutions that may look correct but are actually wrong.

(a) **Solution 1**:

```
SELECT pizza FROM Likes
WHERE  cname = 'Moe'
  AND  pizza NOT IN (
    SELECT pizza FROM Likes
    WHERE cname = 'Lisa'
);
```

Solution 2:

```
SELECT pizza FROM Likes L1
WHERE  cname = 'Moe'
  AND  NOT EXISTS (
    SELECT 1 FROM Likes L2
    WHERE L2.cname = 'Lisa' AND L2.pizza = L1.pizza
);
```

Solution 3:

```
SELECT pizza FROM Likes
WHERE  cname = 'Moe'
  AND  NOT pizza = ANY (
    SELECT pizza FROM Likes
    WHERE cname = 'Lisa'
);
```

Note that if Lisa does not like any pizza, then the `ANY` subquery will evaluate to `FALSE` and `NOT FALSE` will evaluate to `TRUE`. Thus the query will return all the pizzas that Moe likes.

Solution 4: *Probably simplest*

```
SELECT pizza FROM Likes WHERE cname = 'Moe'
EXCEPT
SELECT pizza FROM Likes WHERE cname = 'Lisa';
```

WRONG ANSWER: *The following answer is incorrect*

```
SELECT pizza FROM Likes
WHERE  cname = 'Moe'
  AND  pizza <> ANY (
    SELECT pizza FROM Likes
    WHERE cname = 'Lisa'
);
```

This answer looks similar to **Solution 3** BUT it is *incorrect*. If Lisa does not like any pizza, then the `ANY` subquery will evaluate to `FALSE` and the query will return an *empty set*. This will be incorrect if Moe likes some pizza.

(b) A pizza is the output if there does not exist two distinct restaurants that are located in the same area selling that pizza.

```
SELECT DISTINCT pizza
FROM   Sells S3
WHERE  NOT EXISTS (
  SELECT 1
  FROM   Sells S, Restaurants R, Sells S2, Restaurants R2
  WHERE  S.rname = R.rname     AND  S2.rname = R2.rname
    AND  S.pizza = S2.pizza    AND  R.area = R2.area
    AND  R.rname <> R2.rname   AND  S.pizza = S3.pizza
);
```

WRONG ANSWER: *The following answer is incorrect*

```
SELECT DISTINCT pizza
FROM   Sells S, Restaurants R
WHERE  S.rname = R.rname
  AND  NOT EXISTS (
  SELECT 1
  FROM   Sells S2, Restaurants R2
  WHERE  S2.rname = R2.rname   AND  S.pizza = S2.pizza
    AND  R.area = R2.area      AND  R.rname <> R2.rname
);
```

This answer computes the pizzas that are sold by at most one restaurant in ***some*** area, which is a weaker condition than what is required by the question.

(c) A possible solution

```
SELECT DISTINCT R.area, S.pizza, S.price
FROM   Restaurants R, Sells S
WHERE  R.rname = S.rname
  AND  S.price <= ALL (
    SELECT S2.price
    FROM   Restaurants R2, Sells S2
    WHERE  R2.rname = S2.rname
      AND  R2.area = R.area
      AND  S2.pizza = S.pizza
);
```

3. **(Equivalence)** Consider the query to find distinct restaurants that are located in the East area. The following are two possible SQL answers (*denoted by* Q_1 *and* Q_2) for this query.

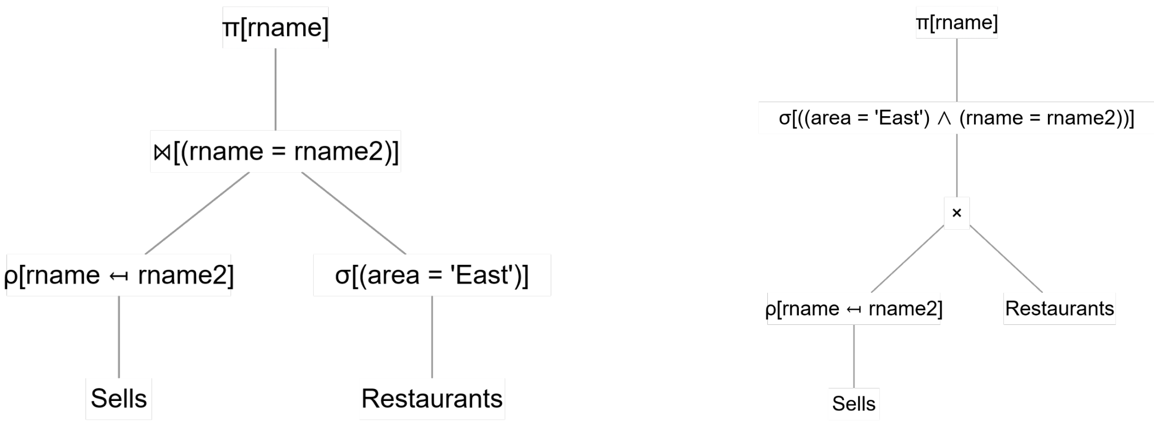
Q_1 Query 1

```sql
SELECT DISTINCT S.rname
FROM   Sells S JOIN Restaurants R
  ON   S.rname = R.rname AND R.area = 'East';
```

Q_2 Query 2

```sql
SELECT DISTINCT S.rname
FROM   Sells S, Restaurants R
WHERE  S.rname = R.rname
  AND R.area = 'East';
```

The semantics of these two SQL queries are defined by the relational algebra expressions shown below. Discuss whether Q_1 and Q_2 are equivalent queries.



(a) Q_1 (b) Q_2

> **Suggested Guide:**
>
> Queries Q_1 and Q_2 are **equivalent**. Whether the selection predicate "`area = 'East'`" is evaluated *before* or *after* the join/cross product operation does not change the semantics of the query.

4. **(Equivalence)** Consider the query to find distinct restaurants that are located in the East area or restaurants that sell some pizza that Lisa likes, where the restaurants that do not sell any pizza are to be excluded. The following are two possible SQL answers (*denoted by* Q_1 *and* Q_2) for this query.

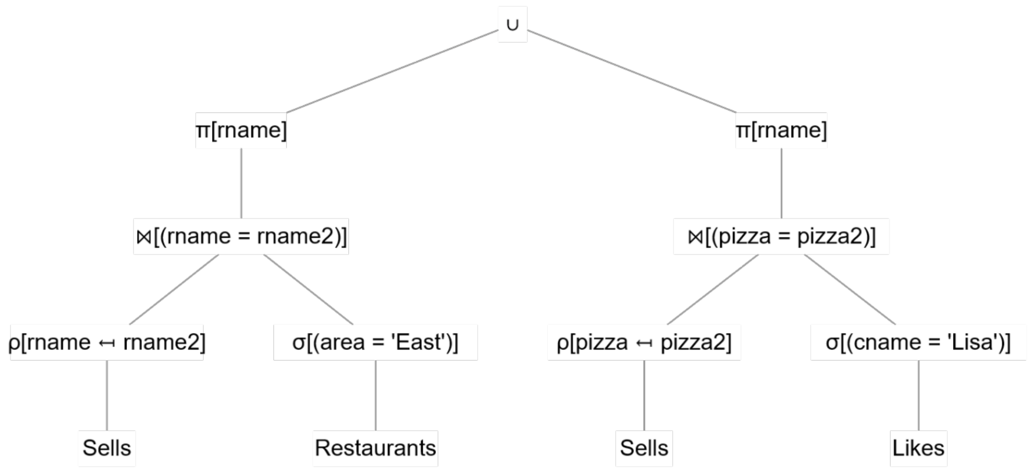
Q_1 Query 1

```sql
SELECT DISTINCT S.rname
FROM   Sells S JOIN Restaurants R
  ON   S.rname = R.rname AND R.area = 'East'
UNION
SELECT DISTINCT S.rname
FROM   Sells S JOIN Likes L
  ON   S.pizza = L.pizza AND L.cname = 'Lisa';
```

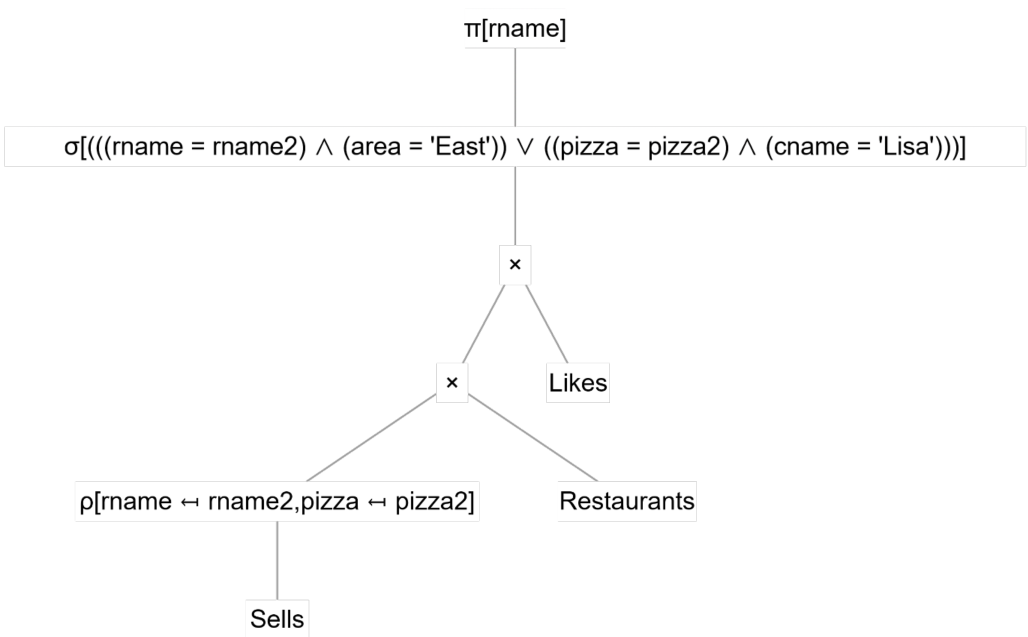
Q_2 Query 2

```sql
SELECT DISTINCT S.rname
FROM   Sells S, Restaurants R, Likes L
WHERE  (S.rname = R.rname AND R.area = 'East')
  OR   (S.pizza = L.pizza AND L.cname = 'Lisa');
```

The semantics of these two SQL queries are defined by the relational algebra expressions shown below. Discuss whether Q_1 and Q_2 are equivalent queries.



(a) Q_1



(b) Q_2

Suggested Guide:

Queries Q_1 and Q_2 are **NOT equivalent**. Observe that if the `Likes` relation is *empty*, then query Q_1 simplifies to

$$\pi_{\texttt{rname}}(\texttt{Sells} \bowtie \sigma_{\texttt{area='East'}}(\texttt{Restaurants}))$$

BUT the result of Q_2 is always an empty set due to

$$\texttt{Sells} \times \texttt{Restaurants} \times \emptyset$$

2 Challenge

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

1. **(SQL Query)** Write an SQL query to answer each of the following questions on the pizza database *without using aggregate functions*. Remove duplicate records from all query results.

   (a) Find all tuples (A, P, P_{min}, P_{max}) where P is a pizza that is available in area A (*i.e.*, there is some restaurant in area A selling pizza P), P_{min} is the *lowest* price of P in area A and P_{max} is the *highest* price of P in area A.

Suggested Guide:

(a) You should recognize that this query is simply an extension of Question 2(c) requiring an additional information (*i.e.*, highest selling price) for each area-pizza pair. For a given area-pizza pair (A, P), the following query will compute the highest price of pizza P in area A.

```sql
SELECT DISTINCT S2.price
FROM   Restaurants R2, Sells S2
WHERE  R2.rname = S2.rname
  AND  R2.area = A  AND  S2.pizza = P
  AND  R2.price >= ALL (
    SELECT S3.price
    FROM   Restaurants R3, Sells S3
    WHERE  R2.rname = S2.rname
      AND  R3.area = A
      AND  S3.pizza = P
);
```

Since the above query will return a *single one-column tuple*, it can be used as a **scalar** subquery to extend the previous question's solution as follows:

```sql
SELECT DISTINCT R.area, S.pizza, S.price AS minPrice, (
    SELECT DISTINCT S2.price
    FROM   Restaurants R2, Sells S2
    WHERE  R2.rname = S2.rname
      AND  R2.area = R.area  AND  S2.pizza = S.pizza
      AND  R2.price >= ALL (
```

```
        SELECT S3.price
        FROM   Restaurants R3, Sells S3
        WHERE  R2.rname = S2.rname
          AND  R3.area = R.area
          AND  S3.pizza = S.pizza
) AS maxPrice
FROM   Restaurants R, Sells S
WHERE  R.rname = S.rname
  AND  S.price <= ALL (
    SELECT S2.price
    FROM   Restaurants R2, Sells S2
    WHERE  R2.rname = S2.rname
      AND  R2.area = R.area
      AND  S2.pizza = S.pizza
);
```

We will learn about other (*simpler and more elegant*) solutions for such queries later in class.

2. **(Update)** Consider *again* the following relational schema discussed in Tutorial 2 Question 2 (*Discussions*).

```
CREATE TABLE Offices (
  office_id    INTEGER,
  building     TEXT NOT NULL,
  level        INTEGER NOT NULL,
  room_number  INTEGER NOT NULL,
  area         INTEGER,
  PRIMARY KEY (office_id),
  UNIQUE (building, level, room_number)
);

CREATE TABLE Employees (
  emp_id      INTEGER,
  name        TEXT NOT NULL,
  office_id   INTEGER NOT NULL,
  manager_id  INTEGER,
  PRIMARY KEY (emp_id),
  FOREIGN KEY (office_id) REFERENCES Offices (office_id)
    ON UPDATE CASCADE,
  FOREIGN KEY (manager_id) REFERENCES Employees (emp_id)
    ON UPDATE CASCADE
);
```

Suppose that the office with `office_id = 123` needs to be renovated. Write an SQL statement to reassign the employees located in this office to another temporary office located at room number 11 on level 5 at the building named *Tower1*.

Hint: You can use subquery in an update statement.

Suggested Guide:

SQL update statement.

```sql
UPDATE Employees
SET    office_id = (SELECT office_id FROM Offices
                    WHERE  building = 'Tower1'
                      AND  level = 5
                      AND  room_number = 11)
WHERE  office_id = 123;
```

3. **(Backward Reasoning)** You are given the following schema:

   - `Students(matric, sname)`
   - `Workings(pid, matric, since)`
   - `Projects(pid, pname)`
   - `Categories(pid, cname)`

   You are also given the following foreign key:

   - (`Workings.matric`) $\rightsquigarrow$ (`Students.matric`)
   - (`Workings.pid`) $\rightsquigarrow$ (`Projects.pid`)
   - (`Categories.pid`) $\rightsquigarrow$ (`Projects.pid`)

   Consider the following SQL query:

```sql
SELECT *
FROM   Students NATURAL JOIN Workings
                NATURAL JOIN Projects
                NATURAL JOIN Categories;
```

   Say it produces the following result:

matric	sname	pid	since	cname
A0001	AA	P01	2002	CA
A0001	AA	P01	2002	CB
A0001	AA	P02	2004	CB
A0002	BB	P01	2003	CA
A0002	BB	P01	2003	CB
A0003	CC	P03	2004	CA
A0003	CC	P03	2004	CC
A0003	CC	P03	2004	CD
A0004	AA	P03	2004	CA
A0004	AA	P03	2004	CC
A0004	AA	P03	2004	CD

Now consider another the query to find all pair of distinct projects' pid (*i.e.*, `(p1, p2)`) such that the two projects have exactly the same set of categories[1]. It produces the following result:

pid	pid
P01	P04
P04	P01
P03	P05
P05	P03

Simply note that P01 and P04 have exactly the same set of categories. Similarly, P03 and P05 have exactly the same set of categories. What is a possible result of the following SQL query? Show your answer using only P01, P03, P04, and P05?

```sql
SELECT pid FROM Categories
EXCEPT ALL
SELECT DISTINCT pid FROM Categories WHERE cname <> 'CA';
```

Suggested Guide:

A possible solution using only information in both tables above is:

pid
P01
P03
P03
P04
P05
P05

[1]This involves some constructs that you will only learn in Lecture 06.