CS2102 Tutorial 4: SQL 2

Wk 6, Sem 2, 2021/22

1. Consider a database consisting of the following two tables shown below:

bar

a	b
1	10
2	20
3	30
4	40

foo

f	a
100	2
200	7
300	3
400	2

For each of the following queries on the database, either state that the query is an *invalid* SQL query or show the query's output if the query is a *valid* SQL query.

(a) Query A

```sql
SELECT *
FROM   bar b
WHERE  EXISTS (
  SELECT 1
  FROM   foo f
  WHERE  f.f > 100
    AND  f.a = b.a
);
```

(b) Query B

```sql
SELECT *
FROM   bar b
WHERE  EXISTS (
  SELECT 1
  FROM   foo f
  WHERE  f.f > 100
)
  AND  f.a = b.a;
```

(c) Query C

```sql
SELECT *
FROM   bar b
WHERE  EXISTS (
  SELECT 1
  FROM   foo f
  WHERE  f.f > 100
    AND  a = b.a
);
```

(d) Query D

```sql
SELECT *
FROM   bar b
WHERE  EXISTS (
  SELECT 1
  FROM   foo f
  WHERE  f.f > 100
    AND  a = a
);
```

(e) Query E

```sql
SELECT *
FROM   bar b
WHERE  EXISTS (
  SELECT 1
  FROM   foo f
  WHERE  f.f > 100
    AND  f.a = b.a
    AND  b > 20
);
```

Solution: We begin the solution with a warning:

⚠ ***It is a good defensive programming practice to use distinct aliases and use explicitly qualified column names.***

(a) **Valid:** This follows the good practice.

a	b
2	20
3	30

(b) **Invalid:** The alias `f` for table `foor` declared in the *inner query* is not visible to the *outer query* due to scoping.

(c) **Valid:** The expression "`a = b.a`" is *equivalent* to "`f.a = b.a`".

a	b
2	20
3	30

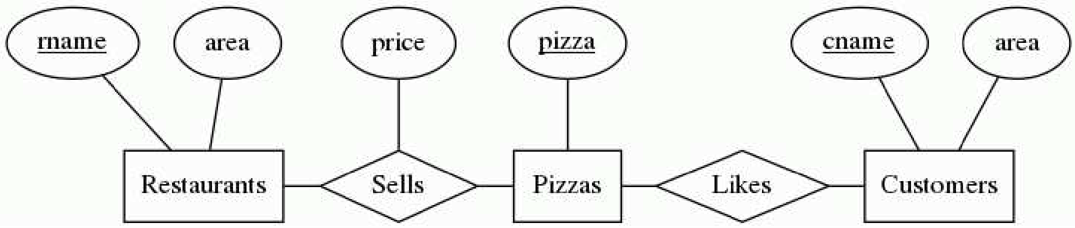
(d) **Valid:** The expression "`f > 100`" is *equivalent* to "`f.f > 100`" (*the* `f` *refers to the attribute not the table alias even though both have the same name*) and "`a = a`" is *equivalent* to "`f.a = f.a`" (*which is always true, hence the result*).

a	b
1	10
2	20
3	30
4	40

(e) **Valid:** The expression "`b > 20`" is *equivalent* to "`b.b > 20`" (*the* `b` *refers to the attribute not the table alias even though both have the same name*). How does it know that it is "`b.b`" instead of "`f.b`"? SQL is smart that way, it knows that the table `foo` or `f` does not have an attribute `b`.

a	b
3	30

2. Questions 2 to 5 are based on the pizza database schema used in the lectures; we show its ER diagram below.



For each of the following queries, write an equivalent SQL query that does not use any subquery.

(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
);
```

Solution:

(a) Query A

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

(b) Query B

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

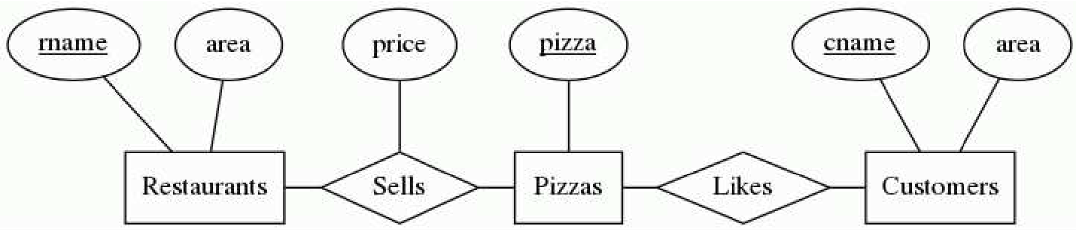
(c) Query C

```
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) Query D

```
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;
```

3. 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 Alice likes but Bob does not like.

(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.

(d) 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.

Solution:

(a) **Solution 1**:

```
SELECT pizza FROM Likes
WHERE  cname = 'Alice'
  AND  pizza NOT IN (
    SELECT pizza FROM Likes
    WHERE cname = 'Bob'
);
```

Solution 2:

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

Solution 3:

```
SELECT pizza FROM Likes
WHERE  cname = 'Alice'
  AND  NOT pizza = ANY (
    SELECT pizza FROM Likes
    WHERE cname = 'Bob'
);
```

Note that if Bob 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 Alice likes.

Solution 4: *Probably simplest*

```
SELECT pizza FROM Likes WHERE cname = 'Alice'
EXCEPT
SELECT pizza FROM Likes WHERE cname = 'Bob';
```

WRONG ANSWER: *The following answer is incorrect*

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

This answer looks similar to **Solution 3** BUT it is *incorrect*. If Bob 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 Alice 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 <> R.2rname   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 <> R.2rname
);
```

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
);
```

(d) You should recognize that this query is simply an extension of the previous query requireing 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.

```
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 S2.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:

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

4. 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.

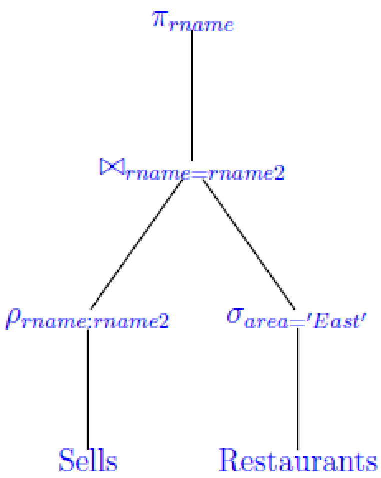
Q_1 Query 1

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

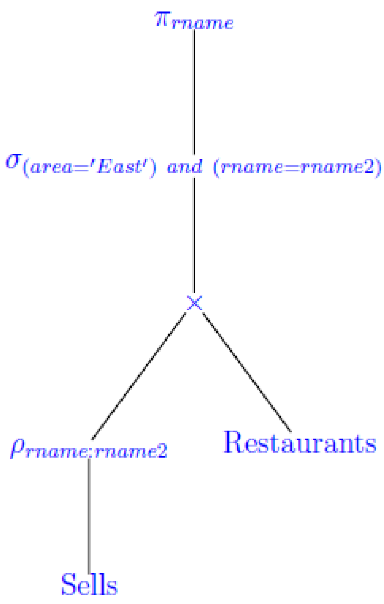
Q_2 Query 2

```
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

Solution: 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.

5. 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.

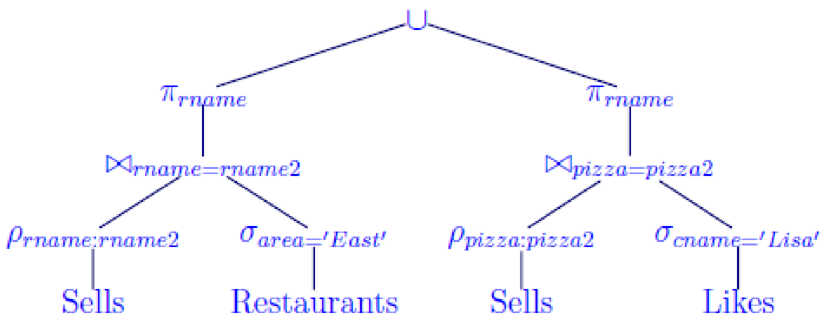
Q_1 Query 1

```
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';
```

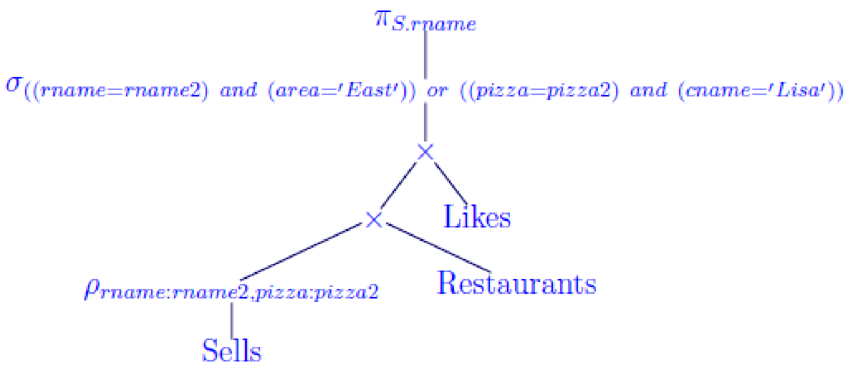
Q_2 Query 2

```
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

Solution: 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

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

6. Consider *again* the following relational schema discussed in Tutorial 2.

```
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*.

Solution: SQL update statement.

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

7. Given the tables R and S shown below, compute the output of each of the following queries.

   (a) `SELECT * FROM R NATURAL JOIN S;`

   (b) `SELECT * FROM R INNER JOIN S ON R.A = S.A;`

   (c) `SELECT * FROM R LEFT OUTER JOIN S ON R.A = S.A;`

   (d) `SELECT * FROM R RIGHT OUTER JOIN S ON R.A = S.A;`

   (e) `SELECT * FROM R FULL OUTER JOIN S ON R.A = S.A;`

R

X	A	Y	B	Z
0	10	0	9	2
30	8	0	5	1
60	4	1	3	3
0	0	0	4	5

S

A	B	C	D
17	1	20	100
4	2	40	200
4	3	30	100
8	5	60	500

Solution:

(a) `SELECT * FROM R NATURAL JOIN S;`

A	B	X	Y	Z	C	D
8	5	30	0	1	60	100
4	3	60	1	3	30	100

(b) `SELECT * FROM R INNER JOIN S ON R.A = S.A;`

X	A	Y	B	Z	A	B	C	D
30	8	0	5	1	8	5	60	500
60	4	1	3	3	4	2	40	200
60	4	1	3	3	4	3	30	100

(c) `SELECT * FROM R LEFT OUTER JOIN S ON R.A = S.A;`

X	A	Y	B	Z	A	B	C	D
0	10	0	9	2	NULL	NULL	NULL	NULL
30	8	0	5	1	8	5	60	500
60	4	1	3	3	4	2	40	200
60	4	1	3	3	4	3	30	100
90	0	0	4	5	NULL	NULL	NULL	NULL

(d) `SELECT * FROM R RIGHT OUTER JOIN S ON R.A = S.A;`

X	A	Y	B	Z	A	B	C	D
30	8	0	5	1	8	5	60	500
60	4	1	3	3	4	2	40	200
60	4	1	3	3	4	3	30	100
NULL	NULL	NULL	NULL	NULL	17	1	20	100

(e) `SELECT * FROM R FULL OUTER JOIN S ON R.A = S.A;`

X	**A**	**Y**	**B**	**Z**	**A**	**B**	**C**	**D**
0	10	0	9	2	NULL	NULL	NULL	NULL
30	8	0	5	1	8	5	60	500
60	4	1	3	3	4	2	40	200
60	4	1	3	3	4	3	30	100
90	0	0	4	5	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	17	1	20	100