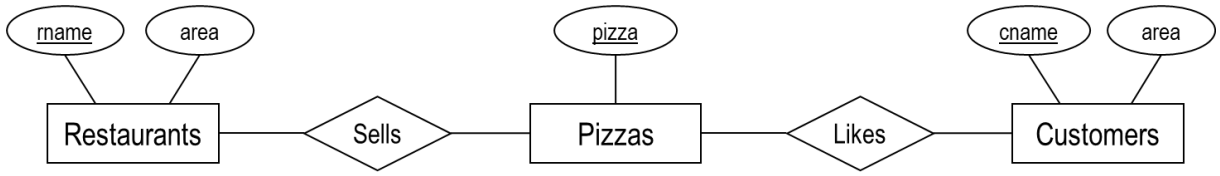
Discussions

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(pizza)`	All the pizzas of interest.
`Customers(cname, area)`	The name and location of each customer.
`Restaurants(rname, area)`	The name and location of each restaurant.
`Recipes(pizza, ingredients)`	The ingredients used in each pizza.
`Sells(rname, pizza, price)`	Pizzas sold by restaurants and the prices.
`Likes(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. **(Basic Query)**
   (a) Find all restaurant names located in the `'East'`.
   (b) Find all areas with at least one customer.
   (c) Find all the pizzas sold by `'Corleone Corner'`.

2. **(Equivalent Query)** For each of the following queries, write an *equivalent* SQL query that does **not** use any subquery. You may use set operation.

   (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
);
```

3. **(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.

   (d) Find the most expensive pizzas and the restaurants that sell them (*at the most expensive price*).

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.

2. **(Update)** Consider the following relational schema.

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

```sql
CREATE TABLE Employees (
  emp_id       INT,
  name         TEXT NOT NULL,
  office_id    INT  NOT NULL,
  manager_id   INT,
  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.

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:

```
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

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

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?

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