Name

UIN

# Midterm Exam

## March 7th, 2024
## 5:00pm - 6:15pm

## CS480 - Database Systems

## Results

---

*Please leave this empty!*

1.1 1.2 1.3 1.4 Sum

# Instructions

- Try to answer all the questions using what you have learned in class. Keep hard questions until the end.
- **When writing a query, write the query in a way that it would work over all possible database instances and not just for the given example instance!**
- The exam is closed book and closed notes!
- **For relational algebra questions assume set semantics!**

Consider the following database schema and example instance for a restaurant database:

## restaurant

| name | city | cuisine |
|---|---|---|
| Dehli Palace | Chicago | indian |
| Goat-a-bunch | Chicago | american |
| Himalayan Experience | Chicago | nepalese |
| Lao Szechuan | Chicago | szechuan |

## menu

| restaurant | item | price | calories |
|---|---|---|---|
| Dehli Palace | Naan | 3.5 | 80 |
| Dehli Palace | Tandoori Chicken | 15.3 | 560 |
| Himalayan Experience | Goat Curry | 28 | 430 |
| Lao Szechuan | Fuqi feipian | 13 | 340 |
| Goat-a-bunch | Cheeseburger | 16 | 740 |

## customer

| cname | city |
|---|---|
| Peter | Chicago |
| Bob | Schaumburg |
| Alice | Madison |
| Sunap | Chicago |

## orders

| oid | restaurant | item | quantity | cname |
|---|---|---|---|---|
| 1 | Dehli Palace | Naan | 3 | Peter |
| 1 | Dehli Palace | Tandoori Chicken | 2 | Peter |
| 2 | Lao Szechuan | Fuqi feipian | 10 | Alice |
| 3 | Lao Szechuan | Fuqi feipian | 1 | Alice |
| 4 | Goat-a-bunch | Cheeseburger | 2 | Sunap |

**Hints:**

- Attributes with black background form the primary key of a relation (.e.g, `name` for relation `restaurant`)
- The attribute `restaurant` of relation `menu` is a foreign key to relation `restaurant`
- The attributes `restaurant` and `item` of relation `orders` is a foreign key to relation `menu`
- The attribute `cname` of relation `orders` is a foreign key to relation `customer`
- All foreign keys have been created with the **CASCADE** option.

## Part 1.1 Relational Algebra (Total: 28 Points)

### Question 1.1.1 (8 Points)

Write a **relational algebra** expression that returns the `price` and `calories` of items on the menu for restaurant *Dehli Palace* that cost more than $10.

### Solution

$$\pi_{price,calories}(\sigma_{restaurant=DehliPalace \wedge price>10}(\text{ menu }))$$

### Question 1.1.2 (10 Points)

Write a **relational algebra** expression that returns the `name` of restaurants that serve *indian* or *nepalese* cuisine and have at least one item from the following list on their menu: *Tandoori Chicken* or *Naan*.

### Solution

$\pi_{name}(\sigma_{cuisine=indian \vee cuisine=nepalese}(\text{ restaurant }) \bowtie_{name=restaurant} \sigma_{item=TandooriChicken \vee item=Naan}(\text{ menu }))$

### Question 1.1.3 (10 Points)

Write a **relational algebra** expression that returns pairs of a customer name (`cname`) with one of the customer's orders (`oid`) for each order that does **not** have any item with a quantity larger than 1.

### Solution

$$\pi_{cname,oid}(\text{ orders }) - \pi_{cname,oid}(\sigma_{quantity>1}(\text{ orders }))$$

## Part 1.2 SQL - DDL (Total: 12 Points)

### Question 1.2.1 (12 Points)

Write an **SQL DDL statement** that creates a new relation `paymentmethod`. A payment method belongs to an existing customer in the database. For each payment method we want to record its `type` (the only valid two values are *credit card* or *debit card*), the `cardnumber` (an 8 digit number sequence), the `expirationdate`, a `pin` (a 4 digit number), and the `issuer` (e.g., *discover* or *master card*). The card number of a payment method is unique. When a customer gets deleted, then we should also delete all of the payment methods for this customer.

### Solution

```
CREATE TABLE paymentmethod (
    cname TEXT REFERENCES customer ON DELETE CASCADE ON UPDATE CASCADE,
    cardnumber CHAR(12) PRIMARY KEY,
    type TEXT CHECK (type = 'credit card' OR type = 'debit card'),
    pin CHAR(4),
    expirationdate DATE,
    issuer TEXT
);
```

## Part 1.3 SQL - Queries (Total: 40 Points)

### Question 1.3.1 (11 Points)

Write an **SQL** query that returns information about ordered items for restaurants that serve *american* cuisine. For each item of an order for such a restaurant, return the `oid`, `restaurant name`, `city`, `item`, and `quantity`.

### Solution

```
SELECT oid, o.restaurant, city, o.item, quantity
FROM restaurant r, orders o, menu m
WHERE r.name = o.restaurant
      AND o.restaurant = m.restaurant
      AND o.item = m.item
      AND r.cuisine = 'american';
```

### Question 1.3.2 (14 Points)

Write an **SQL query** that returns for each restaurant the number of items with more than 300 calories ordered by customers that are not from the city where the restaurant is located. Your query should take the `quantity` of items in an order into account. The result schema should be the name of the restaurant and the number of items fulfilling the condition specified above.

Returning restaurants that have no such orders is optional!

### Solution

```
SELECT name, sum(COALESCE(quantity,0)) AS ttlitems
FROM restaurant r
     LEFT OUTER JOIN
     menu m ON (r.name = m.restaurant AND m.calories > 300)
     LEFT OUTER JOIN
     orders o ON (m.restaurant = o.restaurant AND m.item = o.item)
     LEFT OUTER JOIN
     customer c ON (o.cname = c.cname AND r.city != c.city)
GROUP BY name;
```

### Question 1.3.3 (15 Points)

Write an **SQL query** that calculates for each order the total calories for the order. Do not forget that each item in an order has a quantity. Then return the `oid` and `total calories` for order(s) that have the maximum total calories.

### Solution

```
WITH order_cal AS (SELECT oid, sum(calories * quantity) AS ttlcal
                     FROM menu m, orders o
                    WHERE m.restaurant = o.restaurant
                    GROUP BY oid)
SELECT oid, ttlcal
FROM order_cal
WHERE ttlcal = (SELECT max(ttlcal) FROM order_cal);
```

## Part 1.4 SQL - Updates (Total: 20 Points)

### Question 1.4.1 (7 Points)

Write an **SQL statement** that updates the orders table to reduce the quantity of all items by 50% for orders from the restaurant *"Lao Szechuan"*. Note that quantity has to be an integer value. You can use SQL's round function to round a floating point value.

### Solution

```
UPDATE orders
   SET quantity = round(0.5 * quantity)
 WHERE restaurant = 'Lao Szechuan';
```

### Question 1.4.2 (13 Points)

Write an **SQL statement** that deletes all orders from customers that live in Chicago (the customer's `city` is Chicago).

### Solution

```
DELETE FROM orders
 WHERE cname IN (SELECT cname
                 FROM customer
                 WHERE city = 'Chicago');
```