Homework Assignment 7

- This is a self-study exercise, so feel free to work through it at your own pace.
- Carefully read the questions and try to write and execute SQL queries to solve them.
- You are **allowed to re-use any of the queries from the lecture slides** while developing solutions to the problems.

1. Consider the following tables and answer the questions below.

* Table: Orders

order_id	customer_id	item	order_date	quantity	price
1	101	Burger	5/1/24	2	5
2	102	Pizza	5/1/24	1	12
3	101	Salad	5/2/24	1	6
4	103	Burger	5/2/24	3	5
5	101	Pizza	5/3/24	2	12
6	102	Salad	5/3/24	1	6
7	103	Pizza	5/3/24	1	12
8	101	Burger	5/4/24	1	5

* Table: Customers

customer_id	name
101	Alice Kim
102	Brian Lee
103	Chloe Park
104	David Cho

* Table: Orders
- order_id: A unique identifier for each order.
- customer_id: A reference to the unique ID of the customer who placed the order.
- item: The name of the food item ordered.
- order_date: The date when the order was placed.
- quantity: The number of items ordered.
- price: The price of a single unit of the item at the time of order.

* Table: Customers
- customer_id: A unique identifier for each customer.
- name: The full name of the customer.

Evaluate the provided SQL queries. Write down the expected results. **Remember, all results should be presented in table format.**

(a)
```
SELECT customer_id, order_date, item,
    ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date) AS order_seq,
    LAG(item) OVER (PARTITION BY customer_id ORDER BY order_date) AS prev_item
FROM Orders;
```

customer_id INT	order_date DATE	item VARCHAR	order_seq BIGINT	prev_item VARCHAR
101	2024-05-01	Burger	1	NULL
101	2024-05-02	Salad	2	Burger
101	2024-05-03	Pizza	3	Salad
101	2024-05-04	Burger	4	Pizza
102	2024-05-01	Pizza	1	NULL
102	2024-05-03	Salad	2	Pizza
103	2024-05-02	Burger	1	NULL
103	2024-05-03	Pizza	2	Burger

(b)
```
SELECT order_id, order_date, item, quantity * price AS revenue,
    SUM(quantity * price) OVER (
        ORDER BY order_date
        RANGE BETWEEN INTERVAL 2 DAY PRECEDING AND CURRENT ROW
    ) AS 2day_revenue_sum
FROM Orders;
```

order_id INT	order_date DATE	item VARCHAR	revenue DECIMAL	2day_revenue_sum DECIMAL
1	2024-05-01	Burger	10.00	22.00
2	2024-05-01	Pizza	12.00	22.00
3	2024-05-02	Salad	6.00	43.00
4	2024-05-02	Burger	15.00	43.00
5	2024-05-03	Pizza	24.00	85.00
6	2024-05-03	Salad	6.00	85.00
7	2024-05-03	Pizza	12.00	85.00
8	2024-05-04	Burger	5.00	68.00

(c)
```
SELECT c.name, o.order_date, o.item, o.quantity * o.price AS order_total,
    SUM(o.quantity * o.price) OVER (
        PARTITION BY c.customer_id ORDER BY o.order_date
    ) AS running_total
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id;
```

name VARCHAR	order_date DATE	item VARCHAR	order_total DECIMAL	running_total DECIMAL
Alice Kim	2024-05-01	Burger	10.00	10.00
Alice Kim	2024-05-02	Salad	6.00	16.00
Alice Kim	2024-05-03	Pizza	24.00	40.00
Alice Kim	2024-05-04	Burger	5.00	45.00
Brian Lee	2024-05-01	Pizza	12.00	12.00
Brian Lee	2024-05-03	Salad	6.00	18.00
Chloe Park	2024-05-02	Burger	15.00	15.00
Chloe Park	2024-05-03	Pizza	12.00	27.00

(d) Write an SQL query that returns the following result.

customer_id	item	revenue	revenue_rank
101	Pizza	24	1
101	Burger	15	2
101	Salad	6	3
102	Pizza	12	1
102	Salad	6	2
103	Burger	15	1
103	Pizza	12	2

Each row shows the total revenue per item for each customer, where revenue is defined as the sum of (quantity × price) for each order. Use a window function to assign a rank of revenue within each customer group. You may use `RANK()` and `PARTITION BY` appropriately.

Possible Sol 1:

```
SELECT customer_id, item, SUM(quantity * price) AS revenue,
  RANK() OVER (
    PARTITION BY customer_id ORDER BY SUM(quantity * price) DESC
  ) AS revenue_rank
FROM Orders
GROUP BY customer_id, item;
```

Possible Sol 2:

```
WITH item_revenue AS (
  SELECT customer_id, item, SUM(quantity * price) AS revenue
  FROM Orders
  GROUP BY customer_id, item
)
SELECT *,
  RANK() OVER (PARTITION BY customer_id ORDER BY revenue DESC) AS revenue_rank
FROM item_revenue;
```

2. Structured Qeury Language. Consider the following database tables.

* Table: *artist*

id	name	networth	location
101	Marcus Miller	5,000,000	NY
102	Pat Metheny	10,000,000	MA
103	John Scofield	2,000,000	NY
104	Abraham Laboriel	1,500,000	CA
105	Kenny Garrett	1,500,000	NJ
106	Cory Wong	750,000	MN
107	Jacob Collier	20,000,000	England
108	Victor Wooten	5,000,000	TN
109	Earl Klugh	16,000,000	MI

* Table: *album*

id	a_id	title	year	label
10001	101	Afrodeezia	2015	Blue Note
10002	101	Renaissance	2012	Dreyfus Jazz
10003	102	The Unity Sessions	2016	Nonesuch
10004	102	Speaking of Now	2002	Warner Bros
10005	102	Letter from Home	1989	Geffen
10006	102	First Circle	1984	ECM
10007	103	Combo 66	2018	Verve
10008	103	Uberjam	2002	Verve
10009	103	Groove Elation	1995	Blue Note
10010	104	Guidum	1995	Wigwam
10011	104	Dear Friends	1993	101 South
10012	105	Sounds from the Ancestors	2021	Mack Avenue
10013	105	Beyond the Wall	2006	Nonesuch
10014	105	Songbook	1997	Warner Bros
10015	106	Motivational Music for the Syncopated Soul	2019	
10016	106	The Optimist	2018	
10017	107	In My Room	2016	Membran

(3 pt. each) Given the above tables, evaluate the following queries and write down the expected results.

(a)
```
SELECT location FROM artist
GROUP BY location
HAVING COUNT(*) = 1;
```

location
MA
CA
MN
England
TN
MI

(b)
```
SELECT label, COUNT(*) CNT FROM album
WHERE year < 2000
GROUP BY label
HAVING label LIKE 'W%';
```

label	CNT
Wigwam	1
Warner Bros	1

(c)
```
SELECT name FROM artist
WHERE id IN (
      SELECT a_id FROM album
      WHERE year BETWEEN 2000 AND 2010
)
ORDER BY name;
```

name
John Scofield
Kenny Garrett
Pat Metheny

(d)
```
SELECT name, location FROM artist a1
WHERE EXISTS(
   SELECT * FROM artist a2
   WHERE a1.name <> a2.name AND a1.location = a2.location
);
```

name	location
Marcus Miller	NY
John Scofield	NY
Kenny Garrett	NY

(e)
```
SELECT location, COUNT(DISTINCT album.title) AS CNT FROM artist, album
WHERE artist.id=album.a_id
GROUP BY location
ORDER BY CNT;
```

location	CNT
England	1
CA	2
MN	2
MA	4
NY	8

3. More qurey exercises. Launch and access the MySQL databases distributed with the class virtual machine. Below uses the "*sakila*" database (DVD rental database), which consists of 16 tables regarding movie inventory, actors, customers, rental history, payment information, *etc*. For each of the following questions, **find the answer based on the information recorded in the database** and **write a query that shows how you obtained the answer**.

(a) (3 pt.) How many distinct films rated 'PG' are available?

Query: SELECT COUNT(DISTINCT inventory.film_id, title, rating) FROM inventory
JOIN film ON inventory.film_id = film.film_id
WHERE rating = 'PG';

Answer: 183

(b) (3 pt.) How many active customers are living in the district of England?

Query:
SELECT COUNT(*) FROM customer c
JOIN address a on c.address_id = a.address_id
WHERE district = 'England' AND active=1;

Answer: 6

(c) (4 pt.) Considering the rental history (rental) and payment history (payment), who has paid the largest amount of money for renting movies? List the first and last name of the customer, the total number of movie rentals, and total amount of money s/he has paid.

SELECT first_name, last_name, COUNT(r.rental_id) AS CNT, SUM(amount) AS AMT FROM rental r
JOIN customer c on r.customer_id = c.customer_id
JOIN payment p on r.rental_id = p.rental_id
GROUP BY first_name, last_name
ORDER BY AMT DESC
LIMIT 1;

Answer: KARL SEAL, 45, $221.55

(d) (4 pt.) List three most frequent categories of film available at *store_id*=2 (if a store has multiple copies of the same film, consider each copy as an individual inventory).
Tip: Use **LIMIT** 3 *at the end of your query to limit the number of output tuples.*

Query: SELECT c.name, COUNT(*) AS CNT FROM film f
JOIN film_category fc on f.film_id = fc.film_id
JOIN category c on fc.category_id = c.category_id
JOIN inventory i on f.film_id = i.film_id
WHERE i.store_id=2
GROUP BY c.name
ORDER BY CNT DESC LIMIT 3;

Answer: Sports, Animation, Documentary
15, 2, 6

(e) (3 pt.) What is the title of the movie that has the longest description (film_text.description) among the rental store with store_id=2 has?

Query:

```
SELECT *, LENGTH(description) AS DESC_LENGTH FROM film_text
RIGHT JOIN inventory ON film_text.film_id = inventory.film_id
WHERE store_id = 2
ORDER BY DESC_LENGTH DESC
LIMIT 1
# LIMIT 1 is optional
# One may select different set of queries
# The join type could be one of {JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN} – does not affect the result
```

Answer:
CANDIDATE PERDITION

(j) (4 pt.) Which of the films starred by "FRED COSTNER" rented the most? Write the title of the film.
Query:

```
SELECT i.film_id, f.title, COUNT(rental_id) AS RENTAL_CNT FROM rental
JOIN inventory i on rental.inventory_id = i.inventory_id
JOIN film_actor fa on i.film_id = fa.film_id
JOIN film f on i.film_id = f.film_id
JOIN actor a on fa.actor_id = a.actor_id
WHERE first_name='FRED' AND last_name='COSTNER'
GROUP BY i.film_id, f.title
ORDER BY RENTAL_CNT DESC
LIMIT 1;
```

Answer:
BROTHERHOOD BLANKET

(k) (2 pt.) Using the *'customer_list'* view, list all names of people whose address is in the city of 'London'.
Query:

```
SELECT name FROM customer_list
WHERE city = 'LONDON';
```

Answer:
MATTIE HOFFMAN, CECIL VINES

(l) (3 pt.) Write a query that uses only tables (does not use any views) and returns the same information as in the previous problem (Problem (k)).

Query:

```
SELECT first_name, last_name FROM customer c
JOIN address a on c.address_id = a.address_id
JOIN city c2 on a.city_id = c2.city_id
WHERE c2.city='London';
```

Or

```
SELECT CONCAT(first_name, ' ', last_name) FROM customer c
JOIN address a on c.address_id = a.address_id
JOIN city c2 on a.city_id = c2.city_id
WHERE c2.city='London';
```

Write queries for the following questions and show the results of your queries.

(c) Who are the top five customers spending the most amount of money on movie rental in the database?
Hint: Use the `payment` table.

```
SELECT customer.customer_id, first_name, last_name, SUM(amount) AS SUM_AMT
FROM payment
JOIN customer ON payment.customer_id = customer.customer_id
GROUP BY customer.customer_id, first_name, last_name
ORDER BY SUM_AMT DESC
LIMIT 5;
```

customer_id	first_name	last_name	SUM_AMT
526	KARL	SEAL	221.55
148	ELEANOR	HUNT	216.54
144	CLARA	SHAW	195.58
178	MARION	SNYDER	194.61
137	RHONDA	KENNEDY	194.61

(d) Who are the top five customers renting the most movies in the database?
Hint: Use the `rental` table.

```
SELECT customer.customer_id, first_name, last_name, COUNT(rental_id) AS CNT_RENTAL
FROM rental
JOIN customer ON rental.customer_id = customer.customer_id
GROUP BY customer.customer_id, first_name, last_name
ORDER BY CNT_RENTAL DESC
LIMIT 5;
```

customer_id	first_name	last_name	CNT_RENTAL
148	ELEANOR	HUNT	46
526	KARL	SEAL	45
236	MARCIA	DEAN	42
144	CLARA	SHAW	42
75	TAMMY	SANDERS	41

(e) What of the top seven movies (films) rented the most by the customers?
Hint: Use the `rental`, `inventory`, and `film` tables.

```
SELECT inventory.film_id, film.title, COUNT(*) CNT FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN film ON inventory.film_id = film.film_id
GROUP BY inventory.film_id, film.title
ORDER BY CNT DESC
LIMIT 7;
```

film_id	title	CNT
103	BUCKET BROTHERHOOD	34
738	ROCKETEER MOTHER	33
489	JUGGLER HARDLY	32
767	SCALAWAG DUCK	32
730	RIDGEMONT SUBMARINE	32
331	FORWARD TEMPLE	32
382	GRIT CLOCKWORK	32

4. Answer the following questions that are from the textbook exercise problem sets. You may refer to the Internet as well as the textbook for assistance; however, your solution should contain your own ideas in your own language.

(a) (4 pt.; Exercise 5.6) Consider the bank database of Figure 5.21. Let us define a view *branch_cust* as follows:

CREATE VIEW branch_cust **AS**
SELECT branch_name, customer_name
FROM depositor, account
WHERE depositor.account_number = account.account_number

Suppose that the view is materialized; that is, the view is computed and stored. Write triggers to maintain the view, that is, to keep it up-to-date on insertions to depositor or account. It is not necessary to handle deletions or updates. Note that, for simplicity, we have not required the elimination of duplicates.

```
create trigger insert_into_branch_cust_via_depositor
after insert on depositor
referencing new row as inserted
for each row
insert into branch_cust
      select branch_name, inserted.customer_name
      from account
      where inserted.account_number = account.account_number

create trigger insert_into_branch_cust_via_account
after insert on account
referencing new row as inserted
for each statement
insert into branch_cust
      select inserted.branch_name, customer_name
      from depositor
      where depositor.account_number = inserted.account_number
```

Figure 5.22 Trigger code for Exercise 5.6.

(b) (3 pt.; Exercise 5.7) Consider the bank database of Figure 5.21. Write an SQL trigger to carry out the following action: On **DELETE** of an account, for each customer-owner of the account, check if the owner has any remaining accounts, and if she does not, delete her from the depositor relation.

```
create trigger check-delete-trigger after delete on account
referencing old row as orow
for each row
delete from depositor
where depositor.customer_name not in
      ( select customer_name from depositor
      where account_number <> orow.account_number )
end
```