Advanced SQL Peer Learning Document

Question-1:

Write a query that gives an overview of how many films have replacements costs in the following cost ranges

- low: 9.99 - 19.99
- medium: 20.00 - 24.99
- high: 25.00 - 29.99

Common Solution:

```
SELECT
SUM(CASE WHEN replacement_cost BETWEEN 9.99 AND 19.99 THEN 1 ELSE 0 END) AS
low,
SUM(CASE WHEN replacement_cost BETWEEN 20.00 AND 24.99 THEN 1 ELSE 0 END)
AS medium,
SUM(CASE WHEN replacement_cost BETWEEN 25.00 AND 29.99 THEN 1 ELSE 0 END)
AS high
FROM film;
```

Conclusion: All three of us used the same approach, where we used a case statement to get the result.

Question-2:

Write a query to create a list of the film titles including their film title, film length and film category name ordered descendingly by the film length. Filter the results to only the movies in the category 'Drama' or 'Sports'.

Ratinder's Solution:

```
SELECT f.title, f.length, c.name
FROM film AS f INNER JOIN film_category AS fc
ON f.film_id = fc.film_id
INNER JOIN category AS c
ON fc.category_id = c.category_id
WHERE c.name IN ('Sports', 'Drama')
ORDER BY f.length DESC;
```

Purushottam's Solution:

```
SELECT TITLE,LENGTH_,NAME_ AS CATEGORY
FROM FILM F,FILM_CATEGORY FC,CATEGORY C
WHERE F.FILM_ID=FC.FILM_ID AND FC.CATEGORY_ID=C.CATEGORY_ID AND C.NAME_ IN
('SPORTS','DRAMA')
ORDER BY F.LENGTH_ DESC
```

Conclusion: The approach of all three of us is the same, but the implementation is different. Ratinder solved using ANSI way of joining using inner join, whereas me and purushottam used THETA way of joining using **where** condition.

Question-3:

Write a query to create a list of the addresses that are not associated to any customer.

Ratinder's Solution:

```
SELECT a.address_id, a.address, a.district, a.city_id
FROM address AS a LEFT JOIN customer AS c
ON a.address_id = c.address_id
WHERE c.customer_id IS NULL;
```

Purushottam's Solution:

```
SELECT ADDRESS
FROM ADDRESS A
WHERE A.ADDRESS_ID NOT IN (SELECT ADDRESS_ID FROM CUSTOMER)
```

Conclusion: Ratinder used joining of both address and customer tables to get the output whereas me and purushottam used subquery to solve the problem.

Question-4:

Write a query to create a list of the revenue (sum of amount) grouped by a column in the format "country, city" ordered in decreasing amount of revenue. eg. "Poland, Bydgoszcz" 52.88.

Ratinder and purushottam's Solution:

```
SELECT CONCAT(co.country, ', ', ci.city) AS Country_City,
round(sum(p.amount), 2) AS revenue
FROM payment AS p INNER JOIN customer AS cu
ON p.customer_id = cu.customer_id
INNER JOIN address AS a
ON cu.address_id = a.address_id
INNER JOIN city AS ci
ON ci.city_id = a.city_id
INNER JOIN country AS co
ON co.country_id = ci.country_id
GROUP BY co.country, ci.city;
```

Conclusion: Both ratinder and purushottam solved using the joining of 5 tables to get the output. But I used a derived table to join three tables and joined the other two tables to solve the solution.

Question-5:

Write a query to create a list with the average of the sales amount each staff_id has per customer.

Ratinder's Solution:

```
SELECT t1.staff_id, round(AVG(t1.total_sum), 2)
FROM
(SELECT p.staff_id, p.customer_id, SUM(p.amount) AS total_sum
FROM payment AS p
GROUP BY p.staff_id, p.customer_id) AS t1
GROUP BY t1.staff_id;
```

Purushottam's Solution:

```
SELECT DISTINCT d.staff_id,
ROUND(AVG(d.SUM_by_staff_customer) OVER(PARTITION BY d.staff_id),2) AS
Avg_sales_each_staff_per_cust
FROM
(
SELECT DISTINCT p.staff_id,p.customer_id,
SUM(p.amount) OVER(PARTITION BY p.staff_id,p.customer_id) AS
SUM_by_staff_customer
FROM payment p
) AS d                                          -- d is the alias for this
derived table
ORDER BY Avg_sales_each_staff_per_cust DESC;
```

Conclusion: The approach of all three of us is the same, but the implementation is different. Ratinder used group by function whereas me and purushottam used window function is solve the question.

Question-6:

Write a query that shows average daily revenue of all Sundays.

Ratinder's Solution:

```
SELECT ROUND(AVG(t1.sum_by_each_sunday), 2)
FROM
(SELECT DATE(payment_date), SUM(amount) AS sum_by_each_sunday
FROM payment
WHERE DAYNAME(payment_date)='Sunday'
GROUP BY DATE(payment_date)) AS t1;
```

Purushottam's Solution:

```
SELECT SUM(AMOUNT) /(SELECT  COUNT(DISTINCT DATE(PAYMENT_DATE))  FROM
PAYMENT WHERE WEEKDAY(DATE(PAYMENT_DATE))=6) as Average
FROM PAYMENT
WHERE WEEKDAY((PAYMENT_DATE))=6
```

Conclusion: The function we used to find the payment day is on sunday is different. Ratinder used DAYNAME(), purushottam used WEEKDAY() whereas I used DAYOFWEEK() function to solve the problem.

Question-7:

Write a query to create a list that shows how much the average customer spent in total (customer life-time value) grouped by the different districts.

Conclusion: All three of us used the same approach where we joined the customers table and payments table and address table to get the output.

Question-8:

Write a query to list down the highest overall revenue collected (sum of amount per title) by a film in each category. Result should display the film title, category name and total revenue.
eg. "FOOL MOCKINGBIRD" "Action" 175.77
    "DOGMA FAMILY" "Animation" 178.7
    "BACKLASH UNDEFEATED" "Children" 158.81

My solution:

```
SELECT film_name,
category_name,
collection AS "Highest_Overall_revenue"
FROM
(     SELECT  DISTINCT f.title AS "film_name",
    ct.name AS "Category_name",
      collection,
      RANK() OVER( PARTITION BY ct.name order by collection DESC) AS
"Rank_by_collection"
      FROM film f , film_category fc,
      (     SELECT DISTINCT film_id,
            sum(amount) AS collection
            FROM payment p, rental r, inventory i
            WHERE p.rental_id = r.rental_id AND
            i.inventory_id = r.inventory_id
            GROUP BY film_id
      ) c, category ct
      WHERE f.film_id = c.film_id AND
      fc.film_id = f.film_id AND
      fc.category_id = ct.category_id
) x
WHERE Rank_by_collection =1
```

Ratinder's Solution:

```
SELECT t2.title, t2.name, t2.max_revenue_by_category
FROM
(SELECT distinct t1.title, t1.name, t1.total_revenue_by_film,
MAX(t1.total_revenue_by_film) OVER(PARTITION BY t1.name) AS
max_revenue_by_category
FROM
(SELECT f.title, c.name,
SUM(p.amount) AS total_revenue_by_film
FROM film AS f
INNER JOIN film_category AS fc
ON f.film_id = fc.film_id
INNER JOIN category AS c
ON fc.category_id = c.category_id
INNER JOIN inventory AS i
ON i.film_id = f.film_id
INNER JOIN rental AS r
ON r.inventory_id = i.inventory_id
INNER JOIN payment AS p
ON p.rental_id = r.rental_id
GROUP BY f.title, c.name) AS t1
) AS t2
WHERE max_revenue_by_category=t2.total_revenue_by_film;
```

Purushottam's Solution:

```
CREATE VIEW TOTAL_REV AS(
SELECT F.TITLE X, C.NAME_ Y, ROUND(SUM(P.AMOUNT),2) T
FROM FILM F,INVENTORY I, RENTAL R, FILM_CATEGORY FC,PAYMENT P,
CATEGORY C
WHERE F.FILM_ID=I.FILM_ID AND I.INVENTORY_ID=R.INVENTORY_ID
AND R.RENTAL_ID=P.RENTAL_ID
AND F.FILM_ID=FC.FILM_ID
AND FC.CATEGORY_ID=C.CATEGORY_ID
GROUP BY F.TITLE, C.NAME_ );

CREATE VIEW RANKING AS
(SELECT  TOTAL_REV.X NAME_,TOTAL_REV.Y CATEGORY_, TOTAL_REV.T, RANK()
OVER(PARTITION BY TOTAL_REV.Y ORDER BY TOTAL_REV.T DESC) RANKS
FROM TOTAL_REV);

SELECT * FROM RANKING
WHERE RANKS=1;
```

Conclusion: All three of us have our different ways of solving the problem. Me and Ratinder used derived tables whereas purushottam tried to solve using views concept.

Question-9:

Modify the table "rental" to be partitioned using PARTITION command based on ‘rental_date’ in below intervals:

- <2005
- between 2005–2010
- between 2011–2015
- between 2016–2020
- >2020 - Partitions are created yearly

Common Solution:

```
ALTER TABLE rental
PARTITION BY RANGE (YEAR(rental_date))
(
PARTITION p1_less_than_2005 VALUES LESS THAN (2005),
PARTITION p2_between_2005_2010 VALUES LESS THAN (2011),
PARTITION p3_between_2011_2015 VALUES LESS THAN (2016),
PARTITION p4_between_2016_2020 VALUES LESS THAN (2021),
PARTITION p5_greater_than_2020 VALUES LESS THAN (MAXVALUE)
);
```

Conclusion: All three of us have used the same approach to solve the problem where the table is partitioned using range partitioning.

Question-10:

Modify the table "film" to be partitioned using PARTITION command based on ‘rating’ from below list. Further apply hash sub-partitioning based on ‘film_id’ into 4 sub-partitions.

- partition_1 - "R"
- partition_2 - "PG-13", "PG"
- partition_3 - "G", "NC-17"

Common Solution:

```
ALTER TABLE film
PARTITION BY LIST(rating)
SUBPARTITION BY HASH(film_id) SUBPARTITIONS 4
(
PARTITION PR values('R'),
PARTITION Pgs values('PG-13', 'PG'),
PARTITION GNC values('G', 'NC-17')
);
```

Conclusion: All three of us have used the same approach to solve the problem where the table is sub-partitioned using Hash partitioning.

Question-11:

Write a query to count the total number of addresses from the “address” table where the ‘postal_code’ is of the below formats. Use regular expression. 91**, 92**, 93**, 94**, 9*5**

Ratinder’s Solution:

```
SELECT COUNT(address_id)
FROM address
WHERE postal_code REGEXP '9.[1-5]..';
```

Purushottam’s Solution:

```
SELECT count(postal_code)
FROM address
WHERE postal_code REGEXP '^9[0-9][1-5][0-9]{2}';
```

Conclusion: The regular expressions are different from each other.

Question-12:

Write a query to create a materialized view from the “payment” table where ‘amount’ is between(inclusive) $5 to $8. The view should manually refresh on demand. Also write a query to manually refresh the created materialized view.

Common Solution:

```
DELIMITER $$
CREATE EVENT refresh_payment_between_5_8
ON SCHEDULE EVERY 1 DAY
DO
BEGIN
  CREATE OR REPLACE VIEW payment_between_5_8 AS
  SELECT *
  FROM payment
  WHERE amount BETWEEN 5 AND 8;
```

```
END$$
DELIMITER ;

SELECT * FROM payment_between_5_8;
```

Conclusion: All three of us have used the same approach where we used event which refreshes the view after specific time period.

Question-13:

Write a query to list down the total sales of each staff with each customer from the 'payment' table. In the same result, list down the total sales of each staff i.e. sum of sales from all customers for a particular staff. Use the ROLLUP command. Also use GROUPING command to indicate null values.

Common Solution:

```
SELECT staff_id, customer_id, GROUPING(staff_id) AS flag1,
GROUPING(customer_id) AS flag2, SUM(amount) AS grouped_amt
FROM payment
GROUP BY staff_id, customer_id
WITH ROLLUP;
```

Conclusion: All three of us have the same approach where it is mentioned in the question to use GROUPING command.

Question-14:

Write a single query to display the customer_id, staff_id, payment_id, amount, amount on immediately previous payment_id, amount on immediately next payment_id ny_sales for the payments from customer_id '269' to staff_id '1'.

Common Solution:

```
select customer_id,payment_id,staff_id,
lead(amount) over(order by payment_id) next_payment,
lag(amount) over(order by payment_id) previous_amount,
lead(amount) over(Partition by customer_id,staff_id ORDER BY payment_id) as
ny_sales,
lag(amount) over(Partition by customer_id,staff_id ORDER BY payment_id) as
py_sales
from payment
where customer_id=269 and staff_id=1;
```

Conclusion: All three of us used the same approach where we used the lead(), lag() window functions to solve the problem.