PEER LEARNING DOCUMENT: ADVANCE SQL

MY PEERS :

1) **KARAN BAJAJ**
2) **LAV KUMAR**

MY SOLUTION:

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

SOLUTION:

```
SELECT cost_ranges,COUNT(cost_ranges)
FROM(
SELECT *, CASE WHEN replacement_cost BETWEEN 9.99 AND 19.99 THEN 'low'
WHEN replacement_cost BETWEEN 20 AND 24.99 THEN 'medium'
WHEN replacement_cost BETWEEN 25 AND 29.99 THEN 'high'
end as cost_ranges
FROM film
WHERE replacement_cost<=29.99
ORDER BY replacement_cost) T
GROUP BY cost_ranges
```

My Approach:

This SQL query retrieves the count of movies in different replacement cost ranges from the "film" table. The inner query filters the movies that have a replacement cost of $29.99 or less and categorizes them into three cost ranges: low, medium, and high based on their replacement cost. The outer query groups the movies based on their cost ranges and retrieves the count of movies in each group. The final output is a table with two columns - "cost_ranges" and

"COUNT(cost_ranges)" - which shows the count of movies in each replacement cost range (low, medium, high).

Karan's approach:
He has used a window function where he used CASE statements for filtering the data according to the given range of replacement cost and he categorized those filtered data for each case statement as low,medium and high.

Lav's approach:
His approach is similar to that of mine.

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'.
Eg. "STAR OPERATION" "Sports" 181
"JACKET FRISCO" "Drama" 181

SOLUTION:

```
SELECT title AS film_title,fc.category_id category,
length_ film_length
FROM film f LEFT OUTER JOIN film_category fc
ON  f.film_id=fc.film_id
LEFT OUTER JOIN category c
ON c.category_id=fc.category_id
WHERE c.name in ('Drama','Sports')
ORDER BY film_length DESC;
```

My Approach:

This SQL query selects the film title, film category, and film length from the "film" table and joins it with the "film_category" and "category" tables. The query retrieves records where the category name is either "Drama" or "Sports". The "LEFT OUTER JOIN" is used to include all records from the "film" table and matching records from the "film_category" and "category" tables. The query also sorts the records in descending order of the film length. The final output is a table with three columns - "film_title", "category", and "film_length" - which shows the films in the "Drama" or "Sports" category, ordered by film length in descending order.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach

QUESTION 3
Write a query to create a list of the addresses that are not associated to any customer.

SOLUTION:

```
SELECT *
FROM  ADDRESS
WHERE address_id NOT IN (SELECT address_id
FROM CUSTOMER );
```

My Approach:

This SQL query selects all columns from the "address" table and retrieves records where the "address_id" is not present in the "customer" table. The "NOT IN" clause is used to exclude the records from the "address" table that have an "address_id" present in the "customer" table. The final output is a table with all the columns from the "address" table, but only including records where the "address_id" is not present in the "customer" table.

Karan's approach:
Nearly same solution just i have used subquerry and he has used joins

Lav's approach:
Same approach as that of mine just he also used except statement

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

SOLUTION:

```
SELECT CONCAT(co.country,' , ',ci.city) 'country,city',SUM(amount) revenue
FROM payment pay
JOIN customer cus ON cus.customer_id = pay.customer_id
JOIN address adr ON adr.address_id=cus.address_id
JOIN city ci ON ci.city_id=adr.city_id
JOIN country co ON co.country_id=ci.country_id
GROUP BY co.country,ci.city
ORDER BY revenue DESC;
```

My Approach:
This SQL query selects the concatenated country and city name, and the sum of the payment amount made by each customer from the "payment" table. The query joins the "payment", "customer", "address", "city", and "country" tables to retrieve the relevant information about the customers, their addresses, and their payments. The "JOIN" clause is used to combine the records from these tables based on the specified join conditions. The "GROUP BY" clause is used to group the records based on the country and city names. The "ORDER BY" clause is used to sort the records in descending order of the sum of payments made in each city. The final output is a table with two columns - "country,city" and "revenue" - which shows the total revenue generated in each city, ordered by revenue in descending order.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach

QUESTION 5
Write a query to create a list with the average of the sales amount each staff_id has per customer.

SOLUTION:

```
SELECT STAFF_ID,(AMT/CO) AS AVERAGE_SALES_PER_CUSTOMER
FROM (
SELECT staff_id,COUNT(DISTINCT customer_id) CO,SUM(amount) AMT
FROM payment
GROUP BY staff_id) T
```

My Approach:
This SQL query calculates the average sales per customer for each staff member in the "payment" table by dividing the total payment amount by the number of distinct customers for each staff member. The final output is a table with two columns - "staff_id" and "average_sales_per_customer" - which shows the average sales per customer for each staff member.

Karan's approach:
I have used a derived table and karan used window functions to get the sum of amount spend by each customer per sales id.

Lav's approach:
Same approach

QUESTION 6
Write a query that shows average daily revenue of all Sundays.

SOLUTION:

```
SELECT AVG(ADR) AVERAGE_DAILY_REVENUE
FROM (
SELECT DATE(PAYMENT_DATE ) SUNDAYS,COUNT(DISTINCT CUSTOMER_ID)
CO,sum(AMOUNT) ADR
FROM PAYMENT
WHERE WEEKDAY(PAYMENT_DATE)=6
GROUP BY DATE(PAYMENT_DATE)
ORDER BY DATE(PAYMENT_DATE )) S
```

My Approach:
This SQL query calculates the average daily revenue for Sundays in the "payment" table. The subquery groups the payment records by date and counts the number of distinct customers for each Sunday, and sums the payment amounts made on each Sunday. The outer query then calculates the average daily revenue for Sundays by averaging the sum of payments made on each Sunday. The final output is a table with one column - "AVERAGE_DAILY_REVENUE" - which shows the average daily revenue for Sundays.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach

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.

SOLUTION:

```
SELECT DISTINCT,SUM(AMOUNT)/COUNT(DISTINCT C.CUSTOMER_ID)
AVEREAGE_CUSTOMER_SPENDING
FROM CUSTOMER C JOIN ADDRESS A
ON C.ADDRESS_ID=A.ADDRESS_ID
JOIN PAYMENT P
ON C.CUSTOMER_ID=P.CUSTOMER_ID
GROUP BY DISTRICT
```

Approach:
This SQL query calculates the average customer spending for each district by dividing the total payment amount made by all customers in the district by the number of distinct customers in the district. The query joins the "customer", "address", and "payment" tables to obtain the district and payment amount for each customer, and groups the records by district. The final output is a table with two columns - "district" and "average_customer_spending" - which shows the average customer spending for each district.

Karan's approach:
Karan has used window function for his solution and i have just used joins

Lav's approach:
Same approach as that of mine

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

SOLUTION:

```
WITH T AS
(SELECT name,SUM(AMOUNT) MS,TITLE,DENSE_RANK() OVER(PARTITION BY
NAME    ORDER BY SUM(AMOUNT) DESC) RN
FROM PAYMENT PY JOIN RENTAL RE
ON PY.RENTAL_ID=RE.RENTAL_ID
JOIN INVENTORY IV
ON IV.INVENTORY_ID=RE.INVENTORY_ID
JOIN FILM FM
ON FM.FILM_ID=IV.FILM_ID
JOIN FILM_CATEGORY FC
ON FC.FILM_ID=FM.FILM_ID
JOIN CATEGORY CT
ON CT.CATEGORY_ID=FC.CATEGORY_ID
GROUP BY NAME,FM.FILM_ID,TITLE)

SELECT NAME CATEGORY_NAME,MS AS TOTAL_REVENUE,TITLE
FROM T
WHERE RN=1;
```

My Approach:

This SQL query uses a Common Table Expression (CTE) called "T" to find the top-rented film category for each customer, along with the total revenue earned from each category. The CTE joins multiple tables including payment, rental, inventory, film, film_category, and category, and aggregates the results by customer name, film title, and sum of payment amount. The outer SELECT statement retrieves the category name, total revenue, and film title for each customer's top-rented film category based on the dense rank of the total revenue in descending order.

Karan's approach:
Karan has used multiple derived tables and also used window function in the outer derived table Whereas I have used cte(common table expression) .

Lav's approach:
His approach is similar to that of karan's.

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

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)
);
```

Approach:
This SQL statement alters the table "rental" by partitioning it based on the range of the rental date year. The table is partitioned into five parts: "p1_less_than_2005", "p2_between_2005_2010", "p3_between_2011_2015", "p4_between_2016_2020", and "p5_greater_than_2020". Each partition has a specific range of years, and the data in the "rental" table is distributed among the partitions based on the rental date year. This can help improve query performance and optimize data storage.

Karan’s approach:
Nearly same solution

Lav’s approach:
Same approach

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"

SOLUTION:

```
ALTER TABLE film
PARTITION BY LIST (rating)

SUBPARTITION BY HASH (film_id) SUBPARTITIONS 4 (
PARTITION partition_1 VALUES ("R"),
PARTITION partition_2 VALUES ("PG-13", "PG"),
PARTITION partition_3 VALUES ("G", "NC-17"),
);
```

My Approach:
This SQL statement is used to alter the table "film" by partitioning it based on the "rating" column using the LIST method. Additionally, it specifies that each partition will be further subpartitioned based on the "film_id" column using the HASH method and creating 4 subpartitions. Further there are 3 partitions named "partition_1", "partition_2", and "partition_3" based on the different possible values of the "rating" column.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach

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.
9*1**, 9*2**, 9*3**, 9*4**, 9*5**
eg. postal codes - 91522, 80100, 92712, 60423, 91111, 9211
result - 2

SOLUTION:

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

My Approach:

This SQL query selects the count of postal codes from the ADDRESS table that match the regular expression pattern ^9[0-9][1-5][0-9]{2}$, which matches any five-digit postal code

starting with a 9, where the second digit is between 0 and 9, and the third digit is between 1 and 5.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach

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.

SOLUTION:

```
CREATE VIEW payment_between_5_8 AS

SELECT *
FROM payment
WHERE amount BETWEEN 5 AND 8;
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 ;
```

My Approach:
Here I have created a view named "payment_between_5_8" that selects all rows from the "payment" table where the "amount" column value is between 5 and 8. Then created an event named "refresh_payment_between_5_8" that will run every day and replace the view with the latest data from the "payment" table that matches the condition. The use of the "delimiter" command is to change the default delimiter from ";" to "$$" to allow the creation of the event containing ";" inside its body.

Karan's approach:
Nearly same solution.

Lav's approach:
Same approach

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.

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;
```

My Approach:
This SQL statement uses the GROUP BY clause with ROLLUP to group the payment table by staff_id and customer_id, while also calculating the total amount of payments made by each staff member and customer. The GROUPING function is used to create two flag columns indicating whether the staff_id or customer_id has been aggregated by the ROLLUP operation.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach

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'.

Solution:

```
SELECT customer_id, staff_id, payment_id, amount,
LAG(amount, 1) OVER(ORDER BY payment_id) AS previous_payment_id_amt,
LEAD(amount, 1) OVER(ORDER BY payment_id) AS next_payment_id_amt,
LAG(amount, 1) OVER(PARTITION BY customer_id, staff_id ORDER BY
payment_id) AS py_sales,
LEAD(amount, 1) OVER(PARTITION BY customer_id, staff_id ORDER BY
payment_id) AS ny_sales
FROM payment
WHERE customer_id=269 and staff_id=1;
```

My Approach:
This SQL query selects the payment information for a specific customer and staff member with ID 269 and 1, respectively. It also includes the amount of the previous and next payment, as well as the previous and next payment amount for the same customer and staff member. The LAG and LEAD functions are used to retrieve the values of previous and next rows in the result set, respectively, based on the order of the payment_id column. The PARTITION BY clause is used to group the results by customer and staff, so that the LAG and LEAD functions return the values of the previous and next payments for the same customer and staff member.

Karan's approach:
Nearly same solution

Lav's approach:
Same approach