Advanced SQL Peer Learning Document

Q1. 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:

Query:

```
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;
```

Explanation:

- We have been asked to give the overview of the films where the replacement_cost is in the given range categorizing it into low, medium and high.
- I have used window function where I have used CASE statement for filtering the data according to the given range of replacement cost and I categorize those filtered data for each case statement as low, medium and high.
- If the particular replacement_cost is within the specified range for the particular category then returning 1 else 0 which we are summing up using SUM() function which results in total number of films in that particular range.

Srinivas' 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;
```

Purushottam' 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;
```

Difference:

- They have also used case to get the result.

Q2. 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:

Query:

```
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;
```

Explanation:

- Here we have to create a list of the film titles including their film title, film length and film category name ordered descendingly by the film length for the category “sports” and “drama”
- So for that I have joined three tables i.e., film, film_category, category and extracted the film_title, film_length and film_category by applying the filter for category name to be only “Sports” and “Drama” ordering by the length of the film in descending order.

Srinivas’ solution:

```
SELECT f.film_id, title, length

FROM film_category fc, category c, film f
```

where fc.category_id = c.category_id

AND (c.name = 'Drama' or c.name = 'Sports')

AND fc.film_id = f.film_id

ORDER BY length DESC

Purushottam' 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

Difference:

The approach of all three of us is the same, but the implementation is different. I have solved the ANSI way of joining using inner join, whereas Srinivas and Purushottam used THETA way of joining using **where** condition.

Q3. Write a query to create a list of the addresses that are not associated to any customer.

Solution:

Query:

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;
```

Explanation:

- I have used two tables i.e., address and customer to extract address_id, address, district, city_id and customer_id
- I have used left join above to retrieve all the data from the address table irrespective of whether it is present in the customer table. So whichever customer_id is not associated with any address will result in null values which I am using in the where clause to filter out the data where customer_id is null which will give me all the addresses which are not related to any customer.

Srinivas' solution:

```
SELECT address_id, address,district

FROM address

WHERE address_id NOT IN

(SELECT address_id

FROM customer)
```

Purushottam' solution:

```
SELECT ADDRESS

FROM ADDRESS A

WHERE A.ADDRESS_ID NOT IN

(SELECT ADDRESS_ID FROM CUSTOMER)
```

Difference:

I have used joining of both address and customer tables to get the output whereas Srinivas and Purushottam used subquery to solve the problem.

Q4. 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:

Query:

```
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;
```

Explanation:

- So for getting the desired result I have joined 4 tables i.e., country, city, address and payment to fetch country name, city name which I am concatenating using CONCAT() function.
- Also calculated the revenue by using group by clause and then sum function which is calculating the sum of amount by Country,City.

Srinivas' solution:

```
SELECT concat(co.country,', ', c.city) as "Country, city",
sum(amount) as "revenue"
FROM (
        SELECT a.address_id, city_id, amount
    FROM payment p, customer c, address a
    WHERE p.customer_id = c.customer_id
    AND c.address_id = a.address_id
) a, city c, country co
WHERE a.city_id = c.city_id AND
c.country_id = co.country_id
GROUP BY co.country, c.city
ORDER BY revenue DESC
```

Purushottam' solution:

```
SELECT DISTINCT CONCAT(c.country,', ',ct.city) AS country_city_name,
SUM(p.amount) OVER(PARTITION BY c.country, ct.city) AS revenue
FROM country c,city ct,address ad,customer cu,payment p
WHERE c.country_id=ct.country_id
AND ad.city_id=ct.city_id
AND cu.address_id=ad.address_id
AND p.customer_id=cu.customer_id
```

ORDER BY revenue DESC;

Difference:

Both I and Purushottam solved using the joining of 5 tables to get the output. But Srinivas has used a derived table to join three tables and joined the other two tables to solve the solution.

Q5. Write a query to create a list with the average of the sales amount each staff_id has per customer.

result: 2 56.64 1 55.91

Solution:

Query:

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;

Explanation:

- Here each customer has multiple transaction with particular staff id, so first we need to find total amount of transaction each customer had with particular staff id, so I am calculating that In the derived table (t1) above using group by which is calculating the sum of amount partitioning by staff_id, customer_id.

- Now using derived table I am extracting staff_id and calculating average of the total_sum per customer calculated in derived table to find out the average amount of sales each staff had.

Srinivas' solution:

```
SELECT staff_id, ROUND(AVG(sum_amount), 2) as sales_amount

FROM (

SELECT DISTINCT staff_id, customer_id,

SUM(amount) OVER(PARTITION BY staff_id,customer_id) AS sum_amount

FROM payment

) a

GROUP BY staff_id
```

Purushottam' 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;
```

Difference:

The approach of all three of us is the same, but the implementation is different. I have used group by function whereas Srinivas and Purushottam have used window function to solve the question.

Q6. Write a query that shows average daily revenue of all Sundays.

Solution:

Query:

```
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;
```

Explanation:

- I have used a derived table here named t1 which is calculating total sum of all payments made on each sunday by using dayname function and group by on payment_date.
- Then in the outer query I am calculating average of total payments for every sunday.

Srinivas' solution:

```
SELECT ROUND(AVG(sum_sunday_revenue),2) AS avg_sunday_revenue

FROM(

SELECT SUM(amount) AS sum_sunday_revenue

FROM payment

WHERE DAYOFWEEK(payment_date) = 1

GROUP BY DATE(payment_date)) a
```

Purushottam' 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
```

Difference:

The function we used to find the day of payment date is different. I have used DAYNAME(), Purushottam has used WEEKDAY() whereas Srinivas has used DAYOFWEEK() function to solve the problem.

Q7. 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:

Query:

```
SELECT t1.district, AVG(t1.avg_amt_by_district)
```

```
FROM

(SELECT a.district, c.customer_id, SUM(p.amount) AS avg_amt_by_district

FROM payment AS p

INNER JOIN customer AS c

ON p.customer_id = c.customer_id

INNER JOIN address AS a

ON a.address_id = c.address_id

GROUP BY a.district, c.customer_id) AS t1

GROUP BY t1.district;
```

Explanation:

- In the inner query I calculated the total of all payments made by a customer in a district and named the derived table as t1.
- Then in the outer query I have calculated how much the average customer spent in total (customer life-time value) grouped by the different districts.

Srinivas' solution:

```
SELECT district, SUM(amount)/count(distinct c.customer_id) as "avg_per_district"

FROM customer c, payment p, address a

WHERE c.customer_id = p.customer_id AND

c.address_id = a.address_id

GROUP BY district
```

Purushottam' solution:

```
SELECT DISTRICT Dist,

SUM(AMOUNT)/(SELECT COUNT(CUSTOMER_ID)
```

FROM CUSTOMER JOIN ADDRESS ON CUSTOMER.ADDRESS_ID=ADDRESS.ADDRESS_ID WHERE DISTRICT=Dist) AVERAGE

FROM PAYMENT

JOIN CUSTOMER ON PAYMENT.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID

JOIN ADDRESS ON CUSTOMER.ADDRESS_ID=ADDRESS.ADDRESS_ID

GROUP BY DISTRICT

ORDER BY AVERAGE DESC

Difference:

All three of us used the same approach where we joined the customers table and payments table and address table to get the output. But I have used subquery and both of them have divided total sum by the count of customer_id.

Q8. 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:

Query:

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;
```

Explanation:

- In the innermost query I have calculated the entire earnings of the movie using group by on film title and category name.

- After that I have taken distinct t1.title, t1.name, t1.total_revenue_by_film and maximum revenue of a film in the same category.
- In the outermost query I have selected only those records whose revenue equals maximum revenue in that same category.

Srinivas' 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
```

Purushottam' 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;
```

Difference:

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

Q9. 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:

Query:

```
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)

);
```

Explanation:

- I partitioned the rental table by rental_date with the specified range of years.
- First Partition “p1_less_than_2005” will contain those values where year is less than 2005
- Second Partition “p2_between_2005_2010” will contain those values where year is between 2005 and 2010
- Third Partition “p3_between_2011_2015” will contain those values where year is between 2011 and 2015
- Fourth Partition “p4_between_2016_2020” will contain those values where year is between 2016 and 2020
- And everything after 2020 will be stored in the partition “p5_greater_than_2020”.

Srinivas’ solution:

```
ALTER TABLE rental PARTITION BY RANGE(YEAR(rental_date))

(

  PARTITION p0 VALUES LESS THAN (2005),

  PARTITION p1 VALUES LESS THAN (2011),

  PARTITION p2 VALUES LESS THAN (2016),

  PARTITION p3 VALUES LESS THAN (2021),

  PARTITION p4 VALUES LESS THAN MAXVALUE

);
```

Purushottam’ solution:

```
ALTER TABLE rental

PARTITION BY RANGE(YEAR(rental_date))
```

(

PARTITION rental_less_than_2005 VALUES LESS THAN (2005),

PARTITION rental_between_2005_2010 VALUES LESS THAN (2011),

PARTITION rental_between_2011_2015 VALUES LESS THAN (2016),

PARTITION rental_between_2016_2020 VALUES LESS THAN (2021),

PARTITION rental_greater_than_2020 VALUES LESS THAN MAXVALUE

);

Difference:

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

Q10. 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:

Query:

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'),

);
```

Explanation:

- Here first I have used PARTITION BY LIST based on rating.
- Then I have used SUBPARTITION BY HASH based on film_id.

Srinivas' solution:

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

(

   PARTITION p_r VALUES ('R'),

   PARTITION p_pg13pg VALUES ('PG-13', 'PG'),

   PARTITION p_gnc17 VALUES ('G', 'NC-17')

)

PARTITIONS 4

SUBPARTITION BY HASH (film_id)

(

   SUBPARTITION p1,

   SUBPARTITION p2,

   SUBPARTITION p3,

   SUBPARTITION p4
```

);

Purushottam’ 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')

);

Difference:

All three of us have used the same approach but different names for the partitions to solve the problem where the table is first partitioned using list partitioning and then sub-partitioned using Hash partitioning.

Q11. 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:

Query:

```
SELECT COUNT(address_id)

FROM address

WHERE postal_code REGEXP '^9[0-9][1-5][0-9]{2}$';
```

Explanation:

- Here I have used a regular expression which ensure there is a 9 as first character, then a digit, then a digit between 1-5, then two digits at the end.
- And I have used a count function to get the count of all address_id where postal code matches the regular expression.

Srinivas' solution:

```
SELECT COUNT(*) AS "NumPostalCodes"

FROM address

WHERE postal_code REGEXP '^9[0-9][1-5][0-9]{2}$';
```

Purushottam' solution:

```
SELECT count(postal_code)

FROM address

WHERE postal_code REGEXP '^9[0-9][1-5][0-9]{2}';
```

Difference:

Srinivas and I have used the same regular expression but Purushottam has not used '$' at the end of his regular expression.

Q12. 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:

Query:

```
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 ;
```

Explanation:

- I have created the event named “refresh_payment_between_5_8” which is scheduled to refresh every one day and inside that I have created the normal view which stores all the data of the payment table where the amount is between $5 and $ 8.

Srinivas’ solution:

```
CREATE VIEW payment_between_5_8 AS       -- Creating a view which gives the details of payments between 5 and 8

SELECT *

FROM payment

WHERE amount BETWEEN 5 AND 8;


DELIMITER $$                             -- creating an event which will refresh this particular view after particular time..

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 ;
```

Purushottam' 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;
```

Difference:

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

Q13. 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:

Query:

```
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;
```

Explanation:

- I have used the payment table to extract staff_id,customer_id and applied grouping on both customer_id and staff_id to handle null values and clearly indicate the ROLLUP sum for the particular column.

Srinivas' solution:

```
SELECT staff_id, customer_id,

SUM(amount) AS sales_amount,

GROUPING(staff_id) AS "staff_id_is_null",

GROUPING(customer_id) AS "Customer_id_is_null"

FROM payment

GROUP BY staff_id, customer_id WITH ROLLUP
```

Purushottam' solution:

```
SELECT p.staff_id,p.customer_id,
```

```
GROUPING(p.staff_id) as staff,

GROUPING(p.customer_id) as customer,sum(p.amount) as sum_of_sales

FROM payment p

GROUP BY p.staff_id,p.customer_id

WITH ROLLUP;
```

Difference:

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

Q.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:

Query:

```
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;
```

Explanation:

- I have used the payment table and used the window function LEAD to extract the next payment and LAG to extract the previous payment order by payment_id.
- Then I have found the ny_sales - the amount of the next payment made by the same customer to the same staff member and py sales - the amount of the previous payment made by the same customer to the same staff member.For this as well I have used lead and lag function partitioned by customer_id and staff_id
- Used where clause to filter out the data where customer_id=269 and staff_id=1.

Srinivas' solution:

```
SELECT p.payment_id,p.customer_id, p.staff_id, p.amount,

   LAG(p.amount) OVER (ORDER BY p.payment_id) AS prev_payment_amount,
        -- gives prev payment_id amount

   LEAD(p.amount) OVER ( ORDER BY p.payment_id) AS next_payment_amount,
        -- gives next payment_id amount

   LAG(p.amount) OVER(PARTITION BY customer_id, staff_id order by payment_id)

                        as py_sales, -- Prev payment done by same customer to staff

   LEAD(p.amount) OVER(PARTITION BY customer_id, staff_id order by payment_id)

                        as ny_sales  -- Next payment done by same customer to staff

FROM payment p

WHERE p.customer_id = '269' AND p.staff_id = '1';
```

Purushottam' 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;
```

Difference:

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