Ratinder and srinivas

```
create database advance_sql_assignment;
use advance_sql_assignment;

CREATE TABLE actor(
actor_id  integer,
first_name VARCHAR(40),
last_name VARCHAR(40),
last_update  timestamp
);
CREATE TABLE address(
address_id  integer,
address VARCHAR(300),
address2 VARCHAR(300),
district varchar(40),
city_id VARCHAR(50),
postal_code VARCHAR(50),
phone VARCHAR(50),
last_update  timestamp
);


CREATE TABLE category(
category_id  INTEGER,
name_  VARCHAR(300),
last_update  TIMESTAMP);

CREATE TABLE city(
city_id  INTEGER,
city  VARCHAR(300),
country_id  INTEGER,
last_update  TIMESTAMP);

CREATE TABLE country(
country_id  INTEGER,
country  VARCHAR(300),
last_update  TIMESTAMP);

CREATE TABLE customer(
customer_id  INTEGER,
store_id  INTEGER,
first_name  VARCHAR(300),
last_name  VARCHAR(300),
email  VARCHAR(300),
address_id  INTEGER,
activebool  BOOLEAN,
create_date  TIMESTAMP,
last_update  TIMESTAMP,
active_  INTEGER);
```

```
CREATE TABLE film_actor(
actor_id  INTEGER,
film_id  INTEGER,
last_update  TIMESTAMP);

CREATE TABLE film_category(
film_id  INTEGER,
category_id INTEGER,
last_update  TIMESTAMP);




CREATE TABLE film(
film_id  INTEGER,
title  VARCHAR(300),
description_  VARCHAR(300),
release_year  INTEGER,
language_id  INTEGER,
original_language_id  INTEGER,
rental_duration  INTEGER,
rental_rate  DECIMAL,
length_  INTEGER,
replacement_cost  DECIMAL,
rating  VARCHAR(300),
last_update  TIMESTAMP,
special_features  VARCHAR(300),
fulltext_  VARCHAR(300));

CREATE TABLE inventory(
inventory_id  INTEGER,
film_id  INTEGER,
store_id  INTEGER,
last_update  TIMESTAMP);

CREATE TABLE language(
language_id  INTEGER,
name_  VARCHAR(300),
last_update  TIMESTAMP);

CREATE TABLE payment(
payment_id  INTEGER,
customer_id  INTEGER,
staff_id  INTEGER,
rental_id  INTEGER,
amount  DECIMAL(20,5),
payment_date  TIMESTAMP);
```

```
SELECT * FROM PAYMENT

CREATE TABLE rental(
rental_id  INTEGER,
rental_date  TIMESTAMP,
inventory_id  INTEGER,
customer_id  INTEGER,
return_date  TIMESTAMP,
staff_id  INTEGER,
last_update  TIMESTAMP);

CREATE TABLE staff(
staff_id  INTEGER,
first_name  VARCHAR(300),
last_name  VARCHAR(300),
address_id  INTEGER,
email  VARCHAR(300),
store_id  INTEGER,
active_  VARCHAR(10),
username  VARCHAR(300),
password_  VARCHAR(300),
last_update  TIMESTAMP,
picture  VARCHAR(300));

CREATE TABLE store(
store_id  INTEGER,
manager_staff_id  INTEGER,
address_id  INTEGER,
last_update  TIMESTAMP);
```

Q1. Write a query that gives an overview of how many films have replacements costs in the following cost ranges

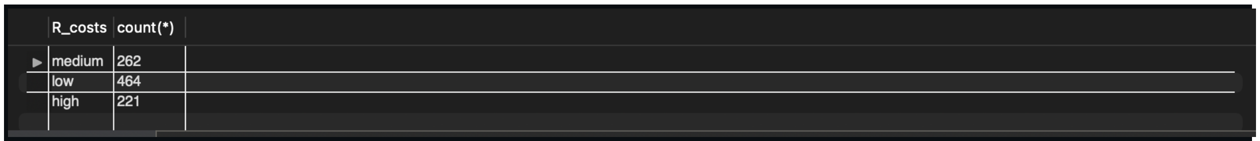
low: 9.99 - 19.99 medium:

20.00 - 24.99 high: 25.00 - 29.99

```
create view sal_garding
as
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 R_costs
from film
where replacement_cost<30;

select R_costs,count(*) from sal_garding
group by R_costs
```

Ratinder and srinivas have same approach

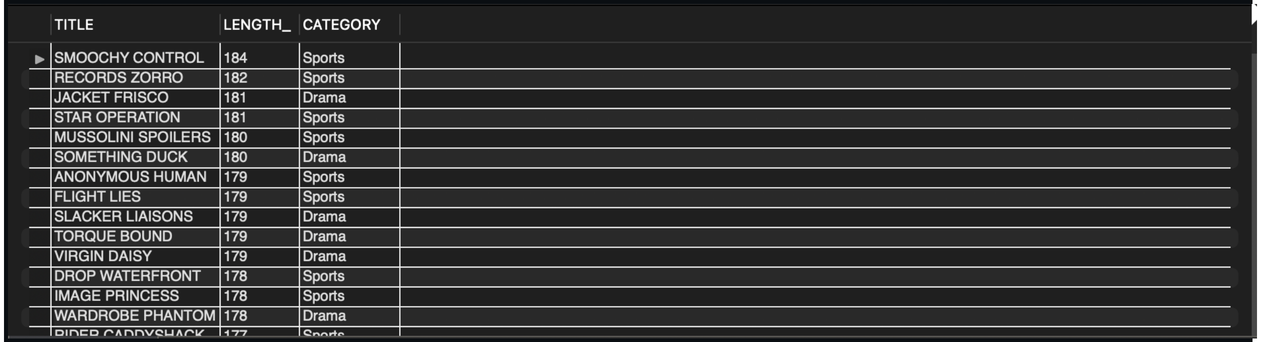


R_costs	count(*)
medium	262
low	464
high	221

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

```
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
```

Srinivas used Correlated query while ratinder used inner join

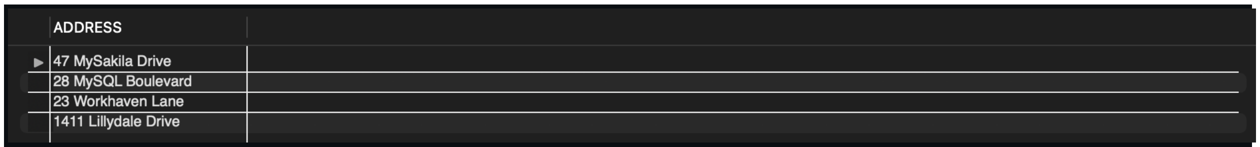


TITLE	LENGTH_	CATEGORY
SMOOCHY CONTROL	184	Sports
RECORDS ZORRO	182	Sports
JACKET FRISCO	181	Drama
STAR OPERATION	181	Sports
MUSSOLINI SPOILERS	180	Sports
SOMETHING DUCK	180	Drama
ANONYMOUS HUMAN	179	Sports
FLIGHT LIES	179	Sports
SLACKER LIAISONS	179	Drama
TORQUE BOUND	179	Drama
VIRGIN DAISY	179	Drama
DROP WATERFRONT	178	Sports
IMAGE PRINCESS	178	Sports
WARDROBE PHANTOM	178	Drama
RIDER CADDYSHACK	177	Sports

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

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

Srinivas have same approach as mine while ratinder used left join

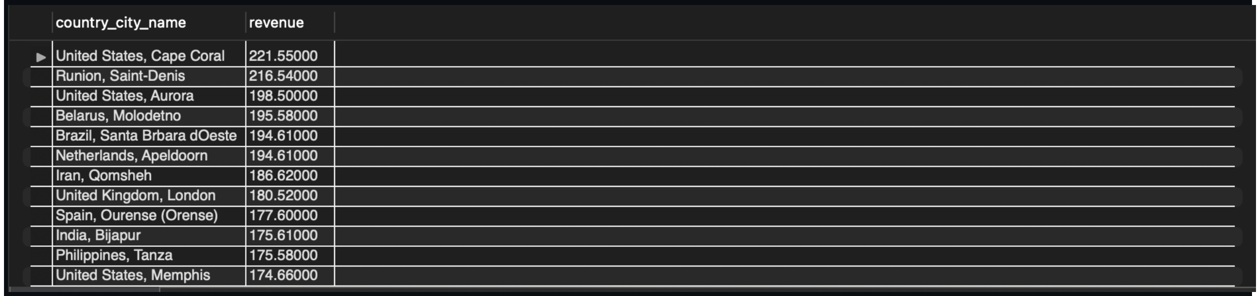


ADDRESS
47 MySakila Drive
28 MySQL Boulevard
23 Workhaven Lane
1411 Lillydale Drive

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

```
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;
```

Srinivas and ratinder both have same project

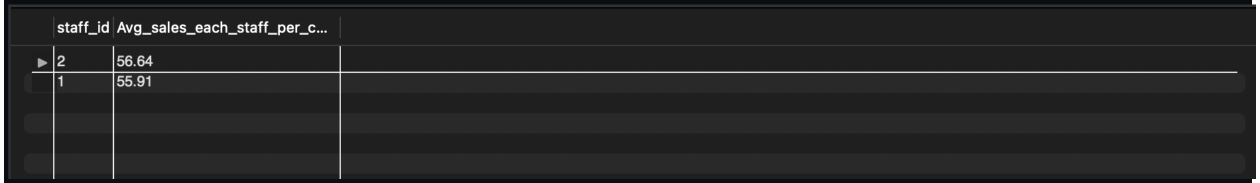


country_city_name	revenue
United States, Cape Coral	221.55000
Runion, Saint-Denis	216.54000
United States, Aurora	198.50000
Belarus, Molodetno	195.58000
Brazil, Santa Brbara dOeste	194.61000
Netherlands, Apeldoorn	194.61000
Iran, Qomsheh	186.62000
United Kingdom, London	180.52000
Spain, Ourense (Orense)	177.60000
India, Bijapur	175.61000
Philippines, Tanza	175.58000
United States, Memphis	174.66000

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

```
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;
```

Both have almost same approach

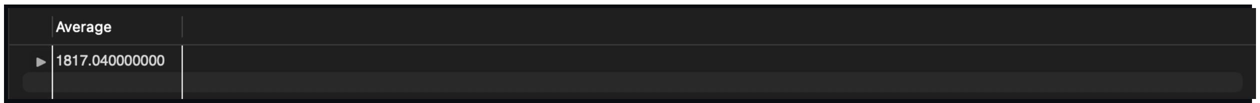


	staff_id	Avg_sales_each_staff_per_c...
▶	2	56.64
	1	55.91

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

```
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
```

Ratinder and srinivas used subquery

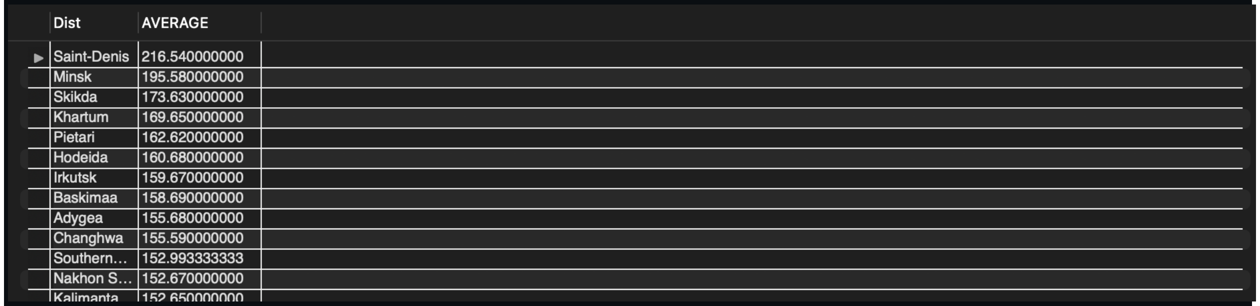


	Average
▶	1817.040000000

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.

```
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
```

Ratinder used inner join while srinivas used correlated subquery



	Dist	AVERAGE
▶	Saint-Denis	216.540000000
	Minsk	195.580000000
	Skikda	173.630000000
	Khartum	169.650000000
	Pietari	162.620000000
	Hodeida	160.680000000
	Irkutsk	159.670000000
	Baskimaa	158.690000000
	Adygea	155.680000000
	Changhwa	155.590000000
	Southern...	152.993333333
	Nakhon S...	152.670000000
	Kalimanta	152.650000000

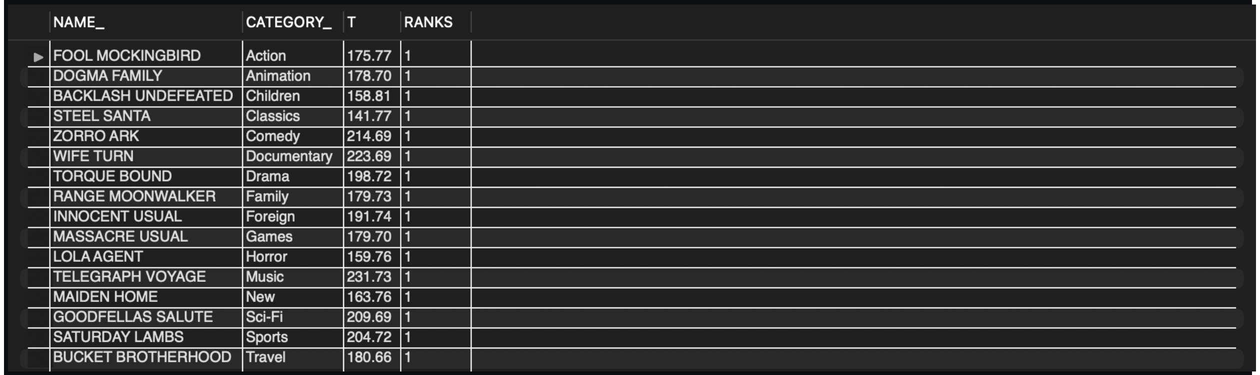
Q8. Write a query to list down the highest overall revenue collected (sum of amount per title) by a film in each category. Result should display the film title, category name and total revenue. eg. "FOOL MOCKINGBIRD" "Action" 175.77 "DOGMA FAMILY" "Animation" 178.7 "BACKLASH UNDEFEATED" "Children" 158.81

```
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;
```

Ratinder used correlated and srinivas used similar approach as mine except i created view for better clearance



NAME_	CATEGORY_	T	RANKS
FOOL MOCKINGBIRD	Action	175.77	1
DOGMA FAMILY	Animation	178.70	1
BACKLASH UNDEFEATED	Children	158.81	1
STEEL SANTA	Classics	141.77	1
ZORRO ARK	Comedy	214.69	1
WIFE TURN	Documentary	223.69	1
TORQUE BOUND	Drama	198.72	1
RANGE MOONWALKER	Family	179.73	1
INNOCENT USUAL	Foreign	191.74	1
MASSACRE USUAL	Games	179.70	1
LOLA AGENT	Horror	159.76	1
TELEGRAPH VOYAGE	Music	231.73	1
MAIDEN HOME	New	163.76	1
GOODFELLAS SALUTE	Sci-Fi	209.69	1
SATURDAY LAMBS	Sports	204.72	1
BUCKET BROTHERHOOD	Travel	180.66	1

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

```
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
);
```

Ratinder and srinivas used almost same approach.

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"

```
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')
);
```

Both used same approach

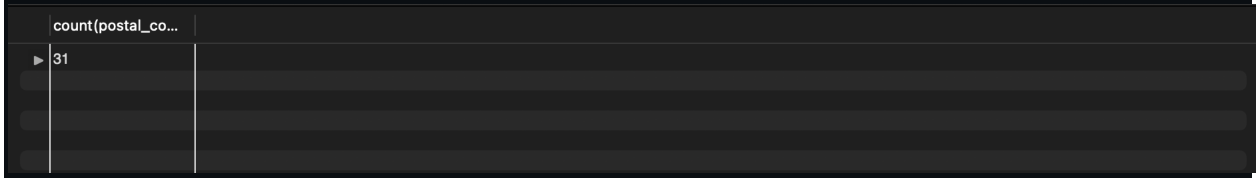
Q11. Write a query to count the total number of addresses from the “address” table where the ‘postal_code’ is of the below formats. Use regular expression.

9*1***, *92***, 9*3***, *94***, 9*5**

eg. postal codes - 91522, 80100, 92712, 60423, 91111, 9211 result - 2

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

Both used regex expression as mine approach



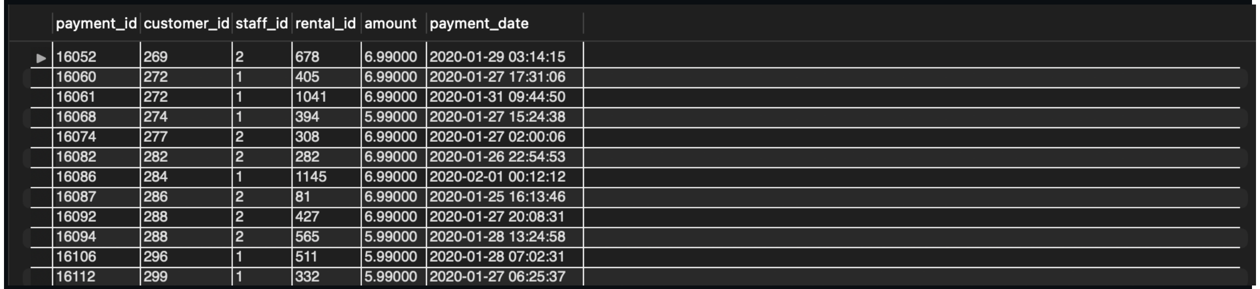
```
count(postal_co...
31
```

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.

```
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;
```

Both used same approach as mine

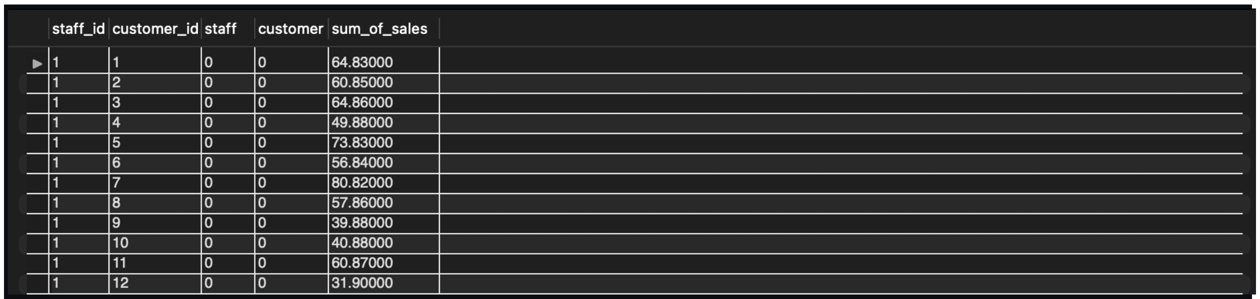


payment_id	customer_id	staff_id	rental_id	amount	payment_date
16052	269	2	678	6.99000	2020-01-29 03:14:15
16060	272	1	405	6.99000	2020-01-27 17:31:06
16061	272	1	1041	6.99000	2020-01-31 09:44:50
16068	274	1	394	5.99000	2020-01-27 15:24:38
16074	277	2	308	6.99000	2020-01-27 02:00:06
16082	282	2	282	6.99000	2020-01-26 22:54:53
16086	284	1	1145	6.99000	2020-02-01 00:12:12
16087	286	2	81	6.99000	2020-01-25 16:13:46
16092	288	2	427	6.99000	2020-01-27 20:08:31
16094	288	2	565	5.99000	2020-01-28 13:24:58
16106	296	1	511	5.99000	2020-01-28 07:02:31
16112	299	1	332	5.99000	2020-01-27 06:25:37

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.

```
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;
```

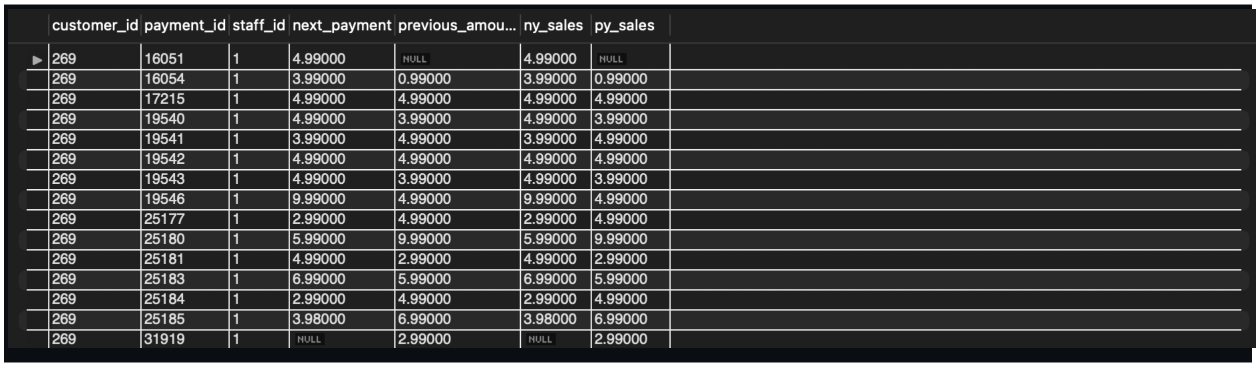
Both used same approach



staff_id	customer_id	staff	customer	sum_of_sales
1	1	0	0	64.83000
1	2	0	0	60.85000
1	3	0	0	64.86000
1	4	0	0	49.88000
1	5	0	0	73.83000
1	6	0	0	56.84000
1	7	0	0	80.82000
1	8	0	0	57.86000
1	9	0	0	39.88000
1	10	0	0	40.88000
1	11	0	0	60.87000
1	12	0	0	31.90000

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’.

```
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;
```



customer_id	payment_id	staff_id	next_payment	previous_amou...	ny_sales	py_sales
269	16051	1	4.99000	NULL	4.99000	NULL
269	16054	1	3.99000	0.99000	3.99000	0.99000
269	17215	1	4.99000	4.99000	4.99000	4.99000
269	19540	1	4.99000	3.99000	4.99000	3.99000
269	19541	1	3.99000	4.99000	3.99000	4.99000
269	19542	1	4.99000	4.99000	4.99000	4.99000
269	19543	1	4.99000	3.99000	4.99000	3.99000
269	19546	1	9.99000	4.99000	9.99000	4.99000
269	25177	1	2.99000	4.99000	2.99000	4.99000
269	25180	1	5.99000	9.99000	5.99000	9.99000
269	25181	1	4.99000	2.99000	4.99000	2.99000
269	25183	1	6.99000	5.99000	6.99000	5.99000
269	25184	1	2.99000	4.99000	2.99000	4.99000
269	25185	1	3.98000	6.99000	3.98000	6.99000
269	31919	1	NULL	2.99000	NULL	2.99000

Both used same approach as mine