ADVANCE SQL ASSIGNMENT

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
```

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;
```

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 );
```

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;
```

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
```

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
```

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 DISTRICT,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
```

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;
```

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)
);
```

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"),
);
```

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}$'
```

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 ;
```

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;
```

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;
```