**Assignments**

**Sub Queries**

1. Get data with all columns of the sales table, and customer name, customer age, product name, and category are in the same result set. (Use join in the subquery, refer to the database and tables from Assignments-05)

**Sol:**

SELECT s.\*, c.customer\_name, c.customer\_age, p.product\_name, p.category

FROM sales s

JOIN customer c ON s.customer\_id = c.customer\_id

JOIN product p ON s.product\_id = p.product\_id;

**Explanation:** In this query, we select all columns from the sales table using s.\*. We also select the customer name and age from the customer table using c.customer\_name and c.customer\_age, respectively. Similarly, we select the product name and category from the product table using p.product\_name and p.category, respectively.

The JOIN keyword is used to join the sales table with the customer and product tables based on their respective foreign keys (s.customer\_id = c.customer\_id and s.product\_id = p.product\_id). This will ensure that we get the customer name, customer age, product name, and category for each sale in the result set.

1. Get data from the sales table, product name, and category in the result set.

**Sol:**

SELECT s.\*, p.product\_name, p.category

FROM sales s

JOIN product p ON s.product\_id = p.product\_id;

**Explanation:** In this query, we select all columns from the sales table using s.\*. We also select the product name and category from the product table using p.product\_name and p.category, respectively.

The JOIN keyword is used to join the sales table with the product table based on their respective foreign keys (s.product\_id = p.product\_id). This will ensure that we get the product name and category for each sale in the result set.

1. Without using the join concept create a sub-query by using the customer, product, sales data.

**Sol:**

SELECT customer\_name, customer\_age, (

SELECT SUM(sales\_amount)

FROM sales

WHERE customer\_id = c.customer\_id) AS total\_sales\_amount

FROM customer c;

**Explanation:** In this query, we select the customer name and age from the customer table using customer\_name and customer\_age, respectively. We also use a subquery to calculate the total sales amount for each customer from the sales table using the SUM function and the WHERE clause to filter the sales by customer\_id.

The subquery is enclosed in parentheses and is used in the main query as an expression in the SELECT clause to retrieve the total sales amount for each customer. We also use an alias total\_sales\_amount to give a name to the calculated column.

**Functions: -**

**string functions: -**

1. Find the maximum length of characters in the Product name string from the Product table

**Sol:**

SELECT MAX(LENGTH(product\_name)) AS max\_product\_name\_length

FROM Product;

**Explanation:** In this query, we select the maximum length of the product\_name column from the Product table using the MAX function and the LENGTH function to calculate the length of each product name.

1. Retrieve product name, sub-category, and category from the Product table and an additional column named “product\_details” which contains a concatenated string of product name, sub-category, and category.

**Sol:**

SELECT product\_name, sub\_category, category, CONCAT(product\_name, ' - ', sub\_category, ' - ', category) AS product\_details

FROM Product;

**Explanation:** In this query, we select the product\_name, sub\_category, and category columns from the Product table. We also use the CONCAT function to concatenate the product\_name, sub\_category, and category columns with a hyphen (-) as a separator to create the "product\_details" column.

1. Analyze the product\_id column and take out the three parts composing the product\_id in three different columns.

**Sol:**

SELECT product\_id, SUBSTRING(product\_id, 1, INSTR(product\_id, '-') - 1) AS part1, SUBSTRING(product\_id, INSTR(product\_id, '-') + 1, INSTR(product\_id, '-', INSTR(product\_id, '-') + 1) - INSTR(product\_id, '-') - 1) AS part2, SUBSTRING(product\_id, INSTR(product\_id, '-', INSTR(product\_id, '-') + 1) + 1) AS part3

FROM Product;

**Explanation:** In this query, we select the product\_id column from the Product table along with three additional columns named part1, part2, and part3. We use the SUBSTRING function to extract the parts of the product\_id column based on the position of the hyphen (-) separator. We use the INSTR function to find the position of the hyphen separator in the product\_id column.

1. List down comma-separated product names where the sub-category is either Chairs or tables.

**Sol:**

SELECT GROUP\_CONCAT(product\_name SEPARATOR ', ') AS products

FROM Product

WHERE sub\_category IN ('Chairs', 'Tables');

**Explanation:** In this query, we select the product\_name column from the Product table and use the GROUP\_CONCAT function to concatenate the product names into a comma-separated list using the SEPARATOR parameter. We also use the WHERE clause to filter the products where the sub\_category is either Chairs or Tables.

**Mathematical functions: -**

1. You are running a lottery for your customers. So, pick a list of 5 lucky customers from the customer table using a random function.

**Sol:**

SELECT \*

FROM Customer

ORDER BY RAND()

LIMIT 5;

**Explanation:** In this query, we select all columns from the Customer table and use the RAND() function to generate a random number for each row. We then order the rows by the random number in ascending order (default) and use the LIMIT clause to select the top 5 rows.

1. Suppose you cannot charge the customer in fraction points. So, for sales value of 1.63, you will get either 1 (or) 2. In such a scenario, find out.
2. Total sales revenue if you are charging the lower integer value of sales always.

**Sol:**

SELECT SUM(FLOOR(sales\_value)) AS total\_revenue

FROM Sales;

**Explanation:** In this query, we select the sum of the FLOOR() function applied to the sales\_value column from the Sales table. The FLOOR() function rounds down the sales value to the nearest integer.

1. Total sales revenue if you are charging the higher integer value of sales always.

**Sol:**

SELECT SUM(CEILING(sales\_value)) AS total\_revenue

FROM Sales;

**Explanation:** In this query, we select the sum of the CEILING() function applied to the sales\_value column from the Sales table. The CEILING() function rounds up the sales value to the nearest integer.

1. Total sales revenue if you are rounding off the sales always.

**Sol:**

SELECT SUM(ROUND(sales\_value)) AS total\_revenue

FROM Sales;

**Explanation:** In this query, we select the sum of the ROUND() function applied to the sales\_value column from the Sales table. The ROUND() function rounds the sales value to the nearest integer.

**Date & Time functions: -**

1. Find out the current age of “batman” who was born on “April 6, 1939” in Years, months, and days

**Sol:**

SELECT

CONCAT(

FLOOR(DATEDIFF(CURDATE(), '1939-04-06') / 365),

' years ',

MONTH(CURDATE()) - MONTH('1939-04-06'),

' months ',

DAY(CURDATE()) - DAY('1939-04-06'),

' days'

) AS current\_age

FROM

Customer

WHERE

customer\_name = 'Batman';

**Explanation:** In this query, we select the current age of Batman by using the DATEDIFF() function to calculate the difference between the current date (CURDATE()) and Batman's birthdate ('1939-04-06') in days. We then divide this difference by 365 to get the age in years and use the FLOOR() function to round down the result. We also use the MONTH() and DAY() functions to calculate the difference in months and days between the current date and Batman's birthdate. Finally, we concatenate these values using the CONCAT() function to get the current age of Batman in years, months, and days.

1. Analyze and find out the monthly sales of the sub-category ‘chair’. Do you Observe any seasonality in sales of this sub-category?

**Sol:**

SELECT

YEAR(order\_date) AS year,

MONTH(order\_date) AS month,

SUM(sales\_value) AS monthly\_sales

FROM

Sales

INNER JOIN

Product ON Sales.product\_id = Product.product\_id

WHERE

Product.sub\_category = 'chair'

GROUP BY

YEAR(order\_date),

MONTH(order\_date);

**Explanation:** In this query, we join the Sales and Product tables on the product\_id column and filter the rows where the sub\_category is "chair". We then group the sales by year and month using the YEAR() and MONTH() functions and calculate the sum of the sales\_value column using the SUM() function. This gives us the monthly sales of the sub-category "chair" for each month and year.

**Joins**

1. Run the below query to create the datasets.
2. /\*Creating sales table of the year 2015\*/

* Create table sales\_2015 as select \* from sales where ship\_date between '2015-01-01' and '2015-12-31';
* select count(\*) from sales\_2015;
* select count(distinct customer\_id) from sales\_2015;

Sol:

/\* Creating sales table of the year 2015 \*/

CREATE TABLE sales\_2015 AS

SELECT \* FROM sales

WHERE ship\_date BETWEEN '2015-01-01' AND '2015-12-31';

/\* Counting the total number of rows in sales\_2015 table \*/

SELECT COUNT(\*) FROM sales\_2015;

/\* Counting the number of distinct customer\_id values in sales\_2015 table \*/

SELECT COUNT(DISTINCT customer\_id) FROM sales\_2015;

1. /\* Customers with ages between 20 and 60 \*/

* create table customer\_20\_60 as select \* from customers where age between 20 and 60;
* select count (\*) from customer\_20\_60;

Sol:

/\* Customers with ages between 20 and 60 \*/

CREATE TABLE customer\_20\_60 AS SELECT \* FROM customer WHERE age BETWEEN 20 AND 60;

SELECT COUNT(\*) FROM customer\_20\_60;

**Explanation:** The given query creates a new table sales\_2015 by selecting all the rows from the existing sales table where the ship\_date column is between January 1, 2015, and December 31, 2015. Then, it performs two select queries to count the total number of rows in the sales\_2015 table and the number of distinct customer\_id values in the sales\_2015 table.

1. Find the total sales that are done in every state for customer\_20\_60 and sales\_2015 table

Hint: Use Joins and Group By command

Sol:

SELECT c.state, SUM(s.sale\_amount)

FROM sales\_2015 s

JOIN customer\_20\_60 c ON s.customer\_id = c.customer\_id

GROUP BY c.state;

**Explanation**: Assuming the customer table has a state column, the query to find the total sales done in every state for customer\_20\_60 and sales\_2015 tables would look like this.

This query joins the sales\_2015 and customer\_20\_60 tables on the customer\_id column, and then groups the results by the state column of the customer\_20\_60 table. The SUM() function is used to calculate the total sales amount for each state.

1. Get data containing Product\_id, Product name, category, total sales value of that product, and total quantity sold. (Use sales and product tables)

Sol:

To get the required data containing Product\_id, Product name, category, total sales value of that product, and total quantity sold, we need to join the sales and products tables and use aggregation functions to calculate the total sales value and total quantity sold for each product. Here's the SQL query to achieve this:

SELECT p.product\_id, p.product\_name, c.category\_name,

SUM(s.sale\_amount) AS total\_sales\_value,

SUM(s.quantity) AS total\_quantity\_sold

FROM sales s

JOIN products p ON s.product\_id = p.product\_id

JOIN categories c ON p.category\_id = c.category\_id

GROUP BY p.product\_id

**Explanation:**

We select the product\_id, product\_name and category\_name columns from the products and categories tables respectively.

We use the SUM function to calculate the total sales value and total quantity sold for each product by grouping the result set by the product\_id column.

Note: The quantity column is assumed to be present in the sales table, which represents the quantity sold for each product in a sale. If the quantity column is not present in the sales table, the query needs to be modified accordingly.