SQL PROJECT:
''''''''''''''''''''''''''''''''

1, Retrieve all products along with the total sales revenue from completed orders.

```
Ans : SELECT
   p.product_id,
   p.product_name,
   SUM(s.quantity * s.price_per_unit) AS total_sales_revenue
FROM products p
JOIN sales s
   ON p.product_id = s.product_id
WHERE s.order_status = 'Completed'
GROUP BY p.product_id, p.product_name
ORDER BY total_sales_revenue DESC;
```

2,List all customers the products they have purchased, showing only those who have ordered more than two products.

Ans: row -wise :
''''''''''''''''''''

```
     SELECT
   c.customer_id,
   c.customer_name,
   p.product_id,
   p.product_name
FROM customers c
JOIN sales s
   ON c.customer_id = s.customer_id
JOIN products p
   ON s.product_id = p.product_id
WHERE c.customer_id IN (
   SELECT customer_id
   FROM sales
   GROUP BY customer_id
   HAVING COUNT(DISTINCT product_id) > 2
)
ORDER BY c.customer_id, p.product_name;
```

Aggregated :
''''''''''''''''''''''''

```
SELECT
   c.customer_id,
   c.customer_name,
   STRING_AGG(DISTINCT p.product_name, ', ') AS products_purchased,
    COUNT(DISTINCT p.product_id) AS total_products
FROM customers c
JOIN sales s
    ON c.customer_id = s.customer_id
JOIN products p
    ON s.product_id = p.product_id
GROUP BY c.customer_id, c.customer_name
HAVING COUNT(DISTINCT p.product_id) > 2
ORDER BY total_products DESC;
```

3.Find the total amount spent by customers in Gujarat who have ordered product price greater than 10,000.

Ans: aggregated:

''''''''''''''''''''''''''''''''

```
SELECT
    c.customer_id,
    c.customer_name,
    SUM(s.quantity * s.price_per_unit) AS total_amount_spent
FROM customers c
JOIN sales s
    ON c.customer_id = s.customer_id
WHERE c.customer_state = 'Gujarat'
  AND s.price_per_unit > 10000
GROUP BY c.customer_id, c.customer_name
ORDER BY total_amount_spent DESC;
```

ROW_WISE:

“”””””””””””’

```
SELECT
    c.customer_id,
    c.customer_name,
    s.order_id,
    s.product_id,
    s.quantity,
    s.price_per_unit,
    (s.quantity * s.price_per_unit) AS order_amount
FROM customers c
JOIN sales s
   ON c.customer_id = s.customer_id
WHERE c.customer_state = 'Gujarat'
  AND s.price_per_unit > 10000
ORDER BY c.customer_id, s.order_id;
```

4.Retrieve the list of all orders that have not yet been shipped.

Ans:SELECT

```
    s.order_id,
    s.product_id,
    s.quantity,
    s.price_per_unit,
    sh.shipping_status
FROM sales s
LEFT JOIN shippings sh
    ON s.order_id = sh.order_id
WHERE sh.shipping_status IS NULL
   OR sh.shipping_status <> 'Shipped'
ORDER BY s.order_id;
```

5.Find the average order value per customer for orders with a quantity of more than 5.

```
Ans:option1:SELECT
    c.customer_id,
    c.customer_name,
    AVG(s.quantity * s.price_per_unit) AS avg_order_value
FROM
    customers c
JOIN
    sales s ON c.customer_id = s.customer_id
WHERE
    s.quantity > 5
GROUP BY
    c.customer_id, c.customer_name
ORDER BY
    avg_order_value DESC;
```

Option 2:

```
'''''''''''''''
SELECT
    customer_id,
    AVG(order_total) AS avg_order_value
FROM (
    SELECT
        s.customer_id,
        s.order_id,
        SUM(s.quantity * s.price_per_unit) AS order_total
    FROM
        sales s
    WHERE
        s.quantity > 5
    GROUP BY
        s.customer_id, s.order_id
) AS order_sums
```

```
GROUP BY customer_id;
```

6.Get the top 5 customers by total spending on accessories.
Ans: using window function:
’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’

```
       SELECT customer_id, customer_name, total_spent
FROM (
   SELECT
      c.customer_id,
      c.customer_name,
      SUM(s.quantity * s.price_per_unit) AS total_spent,
      ROW_NUMBER() OVER (ORDER BY SUM(s.quantity * s.price_per_unit)
DESC) AS rn
   FROM
      customers c
   JOIN
      sales s ON c.customer_id = s.customer_id
   JOIN
      products p ON s.product_id = p.product_id
   WHERE
      p.category = 'Accessories'
   GROUP BY
      c.customer_id, c.customer_name
) numbered
WHERE rn <= 5
ORDER BY rn;
```

Option 2:

```
SELECT
   c.customer_id,
   c.customer_name,
   SUM(s.quantity * s.price_per_unit) AS total_spent
FROM
   customers c
JOIN
   sales s ON c.customer_id = s.customer_id
JOIN
   products p ON s.product_id = p.product_id
WHERE
   p.category = 'Accessories'
GROUP BY
   c.customer_id, c.customer_name
ORDER BY
   total_spent DESC
FETCH FIRST 5 ROWS ONLY;  -- PostgreSQL & Oracle 12c+ syntax
```

7.Retrieve a list of customers who have not made any payment for their orders.
Ans:

```
SELECT DISTINCT
   c.customer_id,
   c.customer_name
FROM
   customers c
JOIN
   sales s ON c.customer_id = s.customer_id
LEFT JOIN
   payments p ON s.order_id = p.order_id
WHERE
   p.payment_id IS NULL
ORDER BY
   c.customer_id;
```

8.Find the most popular product based on total quantity sold in 2023.

```
Ans:SELECT
   p.product_id,
   p.product_name,
   SUM(s.quantity) AS total_quantity_sold
FROM
   sales s
JOIN
   products p ON s.product_id = p.product_id
WHERE
   EXTRACT(YEAR FROM s.order_date) = 2023
GROUP BY
   p.product_id, p.product_name
ORDER BY
   total_quantity_sold DESC
FETCH FIRST 1 ROW ONLY;
```

9.List all orders that were cancelled and the reason for cancellation, if available.

```
Ans:SELECT
   s.order_id,
   s.customer_id,
   s.order_date,
   COALESCE(sh.return_reason, 'Not provided') AS cancellation_reason
```

```
FROM
   sales s
LEFT JOIN
   shipping sh ON s.order_id = sh.order_id
WHERE
   s.order_status = 'Cancelled'
ORDER BY
   s.order_date;
```

Note:coalesce is used for providing default msg as not provided ...!

10.Retrieve the total quantity of products sold by category in 2023.

```
Ans:SELECT
   p.category,
   SUM(s.quantity) AS total_quantity_sold
FROM
   sales s
JOIN
   products p ON s.product_id = p.product_id
WHERE
   EXTRACT(YEAR FROM s.order_date) = 2023
GROUP BY
   p.category
ORDER BY
   total_quantity_sold DESC;
```

11.Get the count of returned orders by shipping provider in 2023.

```
Ans:SELECT
   sh.shipping_providers,
   COUNT(sh.shipping_id) AS returned_orders_count
FROM
   shipping sh
JOIN
   sales s ON sh.order_id = s.order_id
WHERE
   sh.return_date IS NOT NULL
   AND EXTRACT(YEAR FROM sh.return_date) = 2023
GROUP BY
   sh.shipping_providers
ORDER BY
   returned_orders_count DESC;
```

12.Show the total revenge generated per month for the year 2023.

```
Ans:SELECT
   TO_CHAR(s.order_date, 'YYYY-MM') AS month,
   SUM(s.quantity * s.price_per_unit) AS total_revenue
FROM
   sales s
WHERE
   EXTRACT(YEAR FROM s.order_date) = 2023
GROUP BY
   TO_CHAR(s.order_date, 'YYYY-MM')
ORDER BY
   month;
```

13.Find the customer who have made the most purchase in a single month.

```
Ans:SELECT customer_id, customer_name, month, total_purchase
FROM (
   SELECT
      s.customer_id,
      c.customer_name,
      TO_CHAR(s.order_date, 'YYYY-MM') AS month,
      SUM(s.quantity * s.price_per_unit) AS total_purchase,
      RANK() OVER (ORDER BY SUM(s.quantity * s.price_per_unit) DESC) AS
rnk
   FROM
      sales s
   JOIN
      customers c ON s.customer_id = c.customer_id
   GROUP BY
      s.customer_id, c.customer_name, TO_CHAR(s.order_date, 'YYYY-MM')
) ranked
WHERE rnk = 1;
```

14.Retrieve the number of orders made per product category in 2023 and order by total quantity sold.

```
Ans:SELECT
   p.category,
   COUNT(DISTINCT s.order_id) AS total_orders,
   SUM(s.quantity) AS total_quantity_sold
FROM
   sales s
JOIN
   products p ON s.product_id = p.product_id
WHERE
   EXTRACT(YEAR FROM s.order_date) = 2023
GROUP BY
   p.category
```

```
ORDER BY
    total_quantity_sold DESC;
```

15.List the products that have never been ordered.

```
Ans:SELECT
    p.product_id,
    p.product_name,
    p.category,
    p.brand
FROM
    products p
LEFT JOIN
    sales s ON p.product_id = s.product_id
WHERE
    s.order_id IS NULL
ORDER BY
    p.product_id;
```

JOINS:

1.Retrieve a list of all customers with the corresponding product names they ordered. (Use an inner join between customers and sales table.)

```
SELECT
    c.customer_id,
    c.customer_name,
    p.product_name
FROM
    customers c
INNER JOIN sales s
    ON c.customer_id = s.customer_id
INNER JOIN products p
    ON s.product_id = p.product_id
ORDER BY
    c.customer_name, p.product_name;
```

2.List all the products and show the details of customer who have placed order for them. Include products that have no orders. (Use a left join between products and sales table.)

```
SELECT
    p.product_id,
    p.product_name,
    p.category,
    p.brand,
    c.customer_id,
    c.customer_name,
    c.state
FROM
    products p
LEFT JOIN sales s
    ON p.product_id = s.product_id
LEFT JOIN customers c
    ON s.customer_id = c.customer_id
ORDER BY
    p.product_name, c.customer_name;
```

3.List all orders and their shipping status. Include orders that do not have any shipping records.(Use a left join between sales and shipping tables.)

```
SELECT
    s.order_id,
    s.order_date,
    s.customer_id,
    s.order_status,
    sh.shipping_id,
    sh.shipping_date,
    sh.delivery_status
FROM
    sales s
LEFT JOIN shipping sh
    ON s.order_id = sh.order_id
ORDER BY
    s.order_id;
```

4.Retrieve all products including those with no orders along with the price. (Use a right join between the products and sales table).

```
SELECT
    p.product_id,
    p.product_name,
    p.price,
    s.order_id,
    s.quantity
FROM
    sales s
RIGHT JOIN products p
    ON s.product_id = p.product_id
```

```
ORDER BY
   p.product_name;
```

5.Get a list of all customers who have placed orders, including those with no payment records. (Use a full order join between the customers and payments table.)

```
SELECT
   c.customer_id,
   c.customer_name,
   c.state,
   s.order_id,
   p.payment_id,
   p.payment_date,
   p.payment_status
FROM
   customers c
JOIN sales s
   ON c.customer_id = s.customer_id
LEFT JOIN payments p
   ON s.order_id = p.order_id
ORDER BY
   c.customer_name, s.order_id;
```

JOINS+WHERE CLAUSE:

1.Find the total number of completed orders made by customers from the state Delhi, (use inner join between customers and sales, and apply a where condition.)

```
SELECT
   COUNT(*) AS total_completed_orders
FROM
   customers c
INNER JOIN sales s
   ON c.customer_id = s.customer_id
WHERE
   c.state = 'Delhi'
   AND s.order_status = 'Completed';
```

2.Retrieve a list of products ordered by customer from the state Karnataka with price greater than 10,000. Use inner join between sales, customers, and products.

```
SELECT
    c.customer_id,
    c.customer_name,
    c.state,
    p.product_id,
    p.product_name,
    p.price,
    s.order_id,
    s.quantity,
    s.price_per_unit
FROM
    sales s
INNER JOIN customers c
    ON s.customer_id = c.customer_id
INNER JOIN products p
    ON s.product_id = p.product_id
WHERE
    c.state = 'Karnataka'
    AND p.price > 10000
ORDER BY
    c.customer_name, p.product_name;
```

3.List all customers who have placed order where the product category is Accessories and the order status is Completed.(Use in inner join with Sales, Customers, and Products.)

```
SELECT DISTINCT
    c.customer_id,
    c.customer_name,
    c.state
FROM
    sales s
INNER JOIN customers c
    ON s.customer_id = c.customer_id
INNER JOIN products p
    ON s.product_id = p.product_id
WHERE
    p.category = 'Accessories'
    AND s.order_status = 'Completed'
ORDER BY
    c.customer_name;
```

4.Show the order details of customers who have paid for their orders, excluding those who have cancelled their orders. (Use inner join between sales and payment and apply where for order status.)

```
SELECT
   s.order_id,
   s.order_date,
   s.customer_id,
    s.quantity,
   s.price_per_unit,
   s.order_status,
   p.payment_id,
   p.payment_status
FROM
   sales s
INNER JOIN payments p
   ON s.order_id = p.order_id
WHERE
   p.payment_status = 'Paid'
   AND s.order_status <> 'Cancelled'
ORDER BY
   s.order_date;
```

5.Retrieve products ordered by customers who are in the Gujarat state and whose total order price is greater than Rs.15,000. (Use inner join between sales, customers, and products.)

```
SELECT
   c.customer_id,
   c.customer_name,
   c.state,
   p.product_id,
   p.product_name,
   s.order_id,
   s.quantity,
   s.price_per_unit,
   (s.quantity * s.price_per_unit) AS total_order_price
FROM
   sales s
INNER JOIN customers c
   ON s.customer_id = c.customer_id
INNER JOIN products p
   ON s.product_id = p.product_id
WHERE
```

```
    c.state = 'Gujarat'
    AND (s.quantity * s.price_per_unit) > 15000
ORDER BY
    c.customer_name, p.product_name;
```

JOINS+GROUP BY +HAVING

1.Find the total quantity of each product ordered by customers from Delhi and only include products with a quantity greater than 5, (use inner join with sales, customers, and products, and group by product.)

```
SELECT
    p.product_id,
    p.product_name,
    SUM(s.quantity) AS total_quantity
FROM
    sales s
INNER JOIN customers c ON s.customer_id = c.customer_id
INNER JOIN products p ON s.product_id = p.product_id
WHERE
    c.state = 'Delhi'
GROUP BY
    p.product_id, p.product_name
HAVING
    SUM(s.quantity) > 5;
```

2.Get the average payment amount per customer who has placed more than three orders. (Use inner join between payments and sales grouped by customer and apply a halving clause.)

```
SELECT
    c.customer_id,
    c.customer_name,
    AVG(s.quantity * s.price_per_unit)::NUMERIC(12,2) AS avg_payment_amount
FROM
    sales s
INNER JOIN payments p
    ON s.order_id = p.order_id
INNER JOIN customers c
    ON s.customer_id = c.customer_id
WHERE
    p.payment_status = 'Paid'
GROUP BY
    c.customer_id, c.customer_name
HAVING
    COUNT(DISTINCT s.order_id) > 3;
```

3.Retrieve the total sales for each product category and only include categories where the total sales exceed 1,00,000 (use inner join between sales and products group by category.)

```
SELECT
   p.category,
   SUM(s.quantity * s.price_per_unit) AS total_sales
FROM
   sales s
INNER JOIN products p ON s.product_id = p.product_id
GROUP BY
   p.category
HAVING
   SUM(s.quantity * s.price_per_unit) > 100000;
```

4.Show the number of customers in each state who have made purchases with a total spend greater than 50,000 used in a joint between sales and customers.

```
SELECT
   c.state,
   COUNT(c.customer_id) AS customer_count
FROM
   sales s
INNER JOIN customers c
   ON s.customer_id = c.customer_id
GROUP BY
   c.state, c.customer_id
HAVING
   SUM(s.quantity * s.price_per_unit) > 50000;
```

5.List the total sales by brand for products that have been ordered more than 10 times. Use inner join between sales product group by brand.

```
SELECT
   p.brand,
   SUM(s.quantity * s.price_per_unit) AS total_sales
FROM
   sales s
INNER JOIN products p
   ON s.product_id = p.product_id
GROUP BY
   p.brand
HAVING
   SUM(s.quantity) > 10;
```

JOINS+WHERE+GROUP BY +HAVING +ORDER BY

1.Retrieve the total sales per customer in DELHI where the order status is completed. Only include those with total sales greater than 50,000 and order the results by total sales. Use inner join between sales and customer.

```
SELECT
   c.customer_id,
   c.customer_name,
   SUM(s.quantity * s.price_per_unit) AS total_sales
FROM
   sales s
INNER JOIN customers c
   ON s.customer_id = c.customer_id
WHERE
   c.state = 'Delhi'
   AND s.order_status = 'Completed'
GROUP BY
   c.customer_id, c.customer_name
HAVING
   SUM(s.quantity * s.price_per_unit) > 50000
ORDER BY
   total_sales DESC;
```

2.Show the total quantity sold per product in the Accessories category where the total quantity sold is greater than 50 and order the results by product name. Use inner join between sales and product.

```
SELECT
   p.product_id,
   p.product_name,
   SUM(s.quantity) AS total_quantity
FROM
   sales s
INNER JOIN products p
   ON s.product_id = p.product_id
WHERE
   p.category = 'Accessories'
GROUP BY
   p.product_id, p.product_name
HAVING
   SUM(s.quantity) > 50
ORDER BY
   p.product_name;
```

3.Find the total number of orders for customers from Maharashtra who have spent more than 1 lakh and order the results by the total amount spent. Use Inner Join between sales and customer.

```
SELECT
   c.customer_id,
   c.customer_name,
   COUNT(s.order_id) AS total_orders,
   SUM(s.quantity * s.price_per_unit) AS total_spent
FROM
   sales s
INNER JOIN customers c
   ON s.customer_id = c.customer_id
WHERE
   c.state = 'Maharashtra'
GROUP BY
   c.customer_id, c.customer_name
HAVING
   SUM(s.quantity * s.price_per_unit) > 100000
ORDER BY
   total_spent DESC;
```

4.Get the number of orders per product and filter to include only products that have been ordered more than 10 times, then order the results by the highest number of orders. Use inner join between sales and product.

```
SELECT
   p.product_id,
   p.product_name,
   COUNT(s.order_id) AS total_orders
FROM
   sales s
INNER JOIN products p
   ON s.product_id = p.product_id
GROUP BY
   p.product_id, p.product_name
HAVING
   COUNT(s.order_id) > 10
ORDER BY
   total_orders DESC;
```

5.Retrieve the number of payments made per customer where the payment status is Payment Successed and group by customer, order by payment count, use inner join between payments and customer.

```
SELECT
   c.customer_id,
   c.customer_name,
   COUNT(p.payment_id) AS payment_count
FROM
   payments p
INNER JOIN customers c
   ON p.order_id IN (
      SELECT order_id
      FROM sales
      WHERE customer_id = c.customer_id
   )
WHERE
   p.payment_status = 'Payment Successed'
GROUP BY
   c.customer_id, c.customer_name
ORDER BY
   payment_count DESC;
```

DATE FUNCTIONS

1.List all orders that were placed within the year 2023 use order date with the extract function.

```
SELECT
   order_id,
   customer_id,
   product_id,
   quantity,
   price_per_unit,
   order_date,
   order_status
FROM
   sales
WHERE
   EXTRACT(YEAR FROM order_date) = 2023;
```

2.Retrieve customers who have made purchase in the month of January use OrderDate and ToChar to extract the month.

```
SELECT DISTINCT
   c.customer_id,
   c.customer_name,
   c.state
FROM
   sales s
INNER JOIN customers c
   ON s.customer_id = c.customer_id
WHERE
   TO_CHAR(s.order_date, 'MM') = '01';
```

3.Calculate the number of days between the payment date and the order date for each order. Use the Age function.

```
SELECT
    s.order_id,
    s.customer_id,
    p.payment_id,
    s.order_date,
    p.payment_date,
    EXTRACT(DAY FROM AGE(p.payment_date, s.order_date)) AS days_between
FROM
    sales s
INNER JOIN payments p
    ON s.order_id = p.order_id;
```

4.Find the total sales for each year use extract with order date to group by year.

```
SELECT
    EXTRACT(YEAR FROM order_date) AS order_year,
    SUM(quantity * price_per_unit) AS total_sales
FROM
    sales
GROUP BY
    EXTRACT(YEAR FROM order_date);
```

5.Show all orders where the shipping date is after the payment date. Use date comparison.

```
SELECT
    s.order_id,
    s.customer_id,
    TO_CHAR(s.order_date, 'DD-MM-YYYY') AS order_date,
    TO_CHAR(p.payment_date, 'DD-MM-YYYY') AS payment_date,
    TO_CHAR(sh.shipping_date, 'DD-MM-YYYY') AS shipping_date,
    s.order_status
FROM
    sales s
INNER JOIN payments p
    ON s.order_id = p.order_id
```

```
INNER JOIN shipping sh
    ON s.order_id = sh.order_id
WHERE
    sh.shipping_date > p.payment_date;
```