PROJECT OF SQL SOLVED BY ANNU

```sql
use mini_project;

select * from products;

-- level-1:-
-- question1
select name , email from customers;
-- question2
select * from products;
-- question 3
-- List all unique product categories;
-- Useful for analyzing the range of departments or for creating filters on the website
Select  DISTINCT category
FROM products;
-- 4. Show all products priced above ₹1,000
-- this helps identify high-value items for premium promotions or pricing strategy reviews.
select *
from products
where price > 1000;
SELECT *
FROM products
WHERE price > 1000;

-- 5. Display products within a mid-range price bracket (₹2,000 to ₹5,000)
-- A merchandising team might need this to create a mid-tier pricing campaign.

select * from products where price >= 2000 and price <= 5000;

-- 6. Fetch data for specific customer IDs (e.g., from loyalty program list)
-- This is used when customer IDs are pre-selected from another system.

select * from customers;

-- 7. Identify customers whose names start with the letter ‘A’
-- Used for alphabetical segmentation in outreach or app display.

select * from customers where name like 'a%';

-- 8. List electronics products priced under ₹3,000
-- Used by merchandising or frontend teams to showcase budget electronics.
select * from products where price <= 3000;
select * from products where category = 'electronics' and price < 3000;

-- 9. Display product names and prices in descending order of price
-- This helps teams easily view and compare top-priced items.
select name , price from products  order by price desc;

-- 10. Display product names and prices, sorted by price and then by name
-- The merchandising or catalog team may want to list products from most expensive to cheapest. If
-- multiple products have the same price, they should be sorted alphabetically for clarity on
storefronts or printed catalogs.

select name , price from products order by price desc , name asc;


-- Level 2: Filtering and Formatting
-- 1. Retrieve orders where customer information is missing (possibly due to data migration
ordeletion)
-- Used to identify orphaned orders or test data where customer_id is not linked.
select * from orders where customer_id is null;
```

```
-- 2. Display customer names and emails using column aliases for frontend readability
-- Useful for feeding into frontend displays or report headings that require user-friendly labels.
SELECT
    name AS Customer_Name,
    email AS Customer_Email
FROM customers;

-- 3.  * Calculate total value per item ordered by multiplying quantity and item price
-- This can help generate per-line item bill details or invoice breakdowns.
select name, price*stock_quantity as total_value from products;

-- 4. Combine customer name and phone number in a single column
-- Used to show brief customer summaries or contact lists.
select concat(name, "-->", phone) as contact_directory from customers;

-- 5. Extract only the date part from order timestamps for date-wise reporting
-- Helps group or filter orders by date without considering time.
select date(order_date) from orders;

-- 6. List products that do not have any stock left
-- This helps the inventory team identify out-of-stock items.
select * from products where stock_quantity = 0;

-- level3
-- Level 3: Aggregations
-- 1. Count the total number of orders placed
-- Used by business managers to track order volume over time.
select count(*) as count_orders from orders;

-- 2. Calculate the total revenue collected from all orders
-- This gives the overall sales value.
```

```sql
select sum(total_amount) as total_revenue from orders;

-- 3. Calculate the average order value
-- Used for understanding customer spending patterns.
select avg(total_amount) as avg_ordervalue from orders;

-- 4. Count the number of customers who have placed at least one order
-- This identifies active customers.
select count(distinct customer_id) as customer_count from orders;

-- 5. Find the number of orders placed by each customer
-- Helpful for identifying top or repeat customers.
SELECT c.customer_id, c.name, COUNT(o.order_id) AS total_orders
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name;

-- 6. Find total sales amount made by each customer
select customer_id, sum(total_amount) from orders group by customer_id;

-- 7. List the number of products sold per category
-- This helps category managers assess performance by department.
select category , count(product_id) as product_sold from products group by category;

-- 8. Find the average item price per category
-- Useful to compare pricing across departments.
select category, avg(price) as avg_price from products group by category;

-- 9. Show number of orders placed per day
-- Used to track daily business activity and demand trends.
select order_date, count(order_id) as total_orders from orders group by order_date;
```

```
-- 10. List total payments received per payment method

-- Helps the finance team understand preferred transaction modes.

select method , sum(amount_paid) as total_payment from payments group by method;



-- Level 4: Multi-Table Queries (JOINS)

-- 1. Retrieve order details along with the customer name (INNER JOIN)

-- Used for displaying which customer placed each order.

SELECT o.order_id, o.order_date, o.total_amount, c.name

FROM orders o

INNER JOIN customers c

ON o.customer_id = c.customer_id;


-- 2. Get list of products that have been sold (INNER JOIN with order_items)

-- Used to find which products were actually included in orders.

SELECT DISTINCT p.product_id, p.name

FROM products p

INNER JOIN order_items oi

ON p.product_id = oi.product_id;



-- 3. List all orders with their payment method (INNER JOIN)

-- Used by finance or audit teams to see how each order was paid for.

select * from orders;

select * from payments;

SELECT o.order_id, o.order_date, p.method

FROM orders o

INNER JOIN payments p

ON o.order_id = p.order_id;
```

```
-- 4. Get list of customers and their orders (LEFT JOIN)

-- Used to find all customers and see who has or hasn’t placed orders.

SELECT c.customer_id, c.name, o.order_id

FROM customers c

LEFT JOIN orders o

ON c.customer_id = o.customer_id;



-- 5. List all products along with order item quantity (LEFT JOIN)

-- Useful for inventory teams to track what sold and what hasn’t.

SELECT

   p.product_id,

   p.name,

   p.stock_quantity,

   oi.quantity AS sold_quantity

FROM products p

LEFT JOIN order_items oi

ON p.product_id = oi.product_id;



-- 6. List all payments including those with no matching orders (RIGHT JOIN)

-- Rare but used when ensuring all payments are mapped correctly.

SELECT p.payment_id, p.amount_paid, o.order_id

FROM payments p

LEFT JOIN orders o

ON p.order_id = o.order_id;



-- 7. Combine data from three tables: customer, order, and payment
```

```sql
-- Used for detailed transaction reports.
select c.*, o.*, p.*
from orders o
left join customers c
on o.customer_id =c.customer_id
left join payments p
on o.order_id = p.order_id;




-- Level 5: Subqueries (Inner Queries)
-- 1. List all products priced above the average product price
-- Used by pricing analysts to identify premium-priced products.
select * from products where price > (select avg(price) from products);

-- 2. Find customers who have placed at least one order
-- Used to identify active customers for loyalty campaigns.
SELECT customer_id, name
FROM customers
WHERE customer_id IN (
   SELECT customer_id FROM orders
);


-- 3. Show orders whose total amount is above the average for that customer
-- Used to detect unusually high purchases per customer.
SELECT o.order_id, o.customer_id, o.total_amount
FROM orders o
JOIN (
   SELECT customer_id, AVG(total_amount) AS avg_amount
```

```
    FROM orders
    GROUP BY customer_id
) a
ON o.customer_id = a.customer_id
WHERE o.total_amount > a.avg_amount;


-- 4. Display customers who haven’t placed any orders
-- Used for re-engagement campaigns targeting inactive users.
select name from customers where customer_id not in
(select customer_id from orders);

SELECT customer_id, name
FROM customers
WHERE customer_id NOT IN (
    SELECT customer_id FROM orders
);


-- 5. Show products that were never ordered
-- Helps with inventory clearance decisions or product deactivation.
select name from products where product_id not in
(select product_id from orders);


-- 6. Show highest value order per customer
-- Used to identify the largest transaction made by each customer.
select customer_id, max(total_amount) from orders
group by customer_id;

-- 7. Highest Order Per Customer (Including Names)
-- Used to identify the largest transaction made by each customer. Outputs name as well.

SELECT c.customer_id, c.name, MAX(o.total_amount) AS highest_order

FROM customers c

JOIN orders o

ON c.customer_id = o.customer_id

GROUP BY c.customer_id, c.name;
```

```
-- Level 6: Set Operations

-- 1. List all customers who have either placed an order or written a product review

-- Used to identify engaged customers from different activity areas.

select customer_id from orders

union

select customer_id from product_reviews;
```

```
-- 2. List all customers who have placed an order as well as reviewed a product [intersect notsupported]

-- Used to identify highly engaged customers for rewards.

select distinct customer_id from orders where customer_id in

(select customer_id from product_reviews);
```