SQL for Data Analysis

Objective: Use SQL queries to extract and analyze data from a database.

Tools: Oracle (Used Oracle as it was available on my laptop).

Deliverables: SQL queries in a SQL file + screenshots of output

Dataset: This SQL task was performed using Oracle Database.

- Database Name: Raj_task_3
- Tables Used: products, orders, order_items, customers, categories, payments

Task Summary:
In this task, an eCommerce database was created using Oracle SQL with tables like `customers`, `products`, `orders`, and `payments`. Realistic data was inserted, and various SQL queries were executed using `SELECT`, `JOIN`, `GROUP BY`, `SUM`, and `AVG` to analyze sales, customer behavior, and product performance.

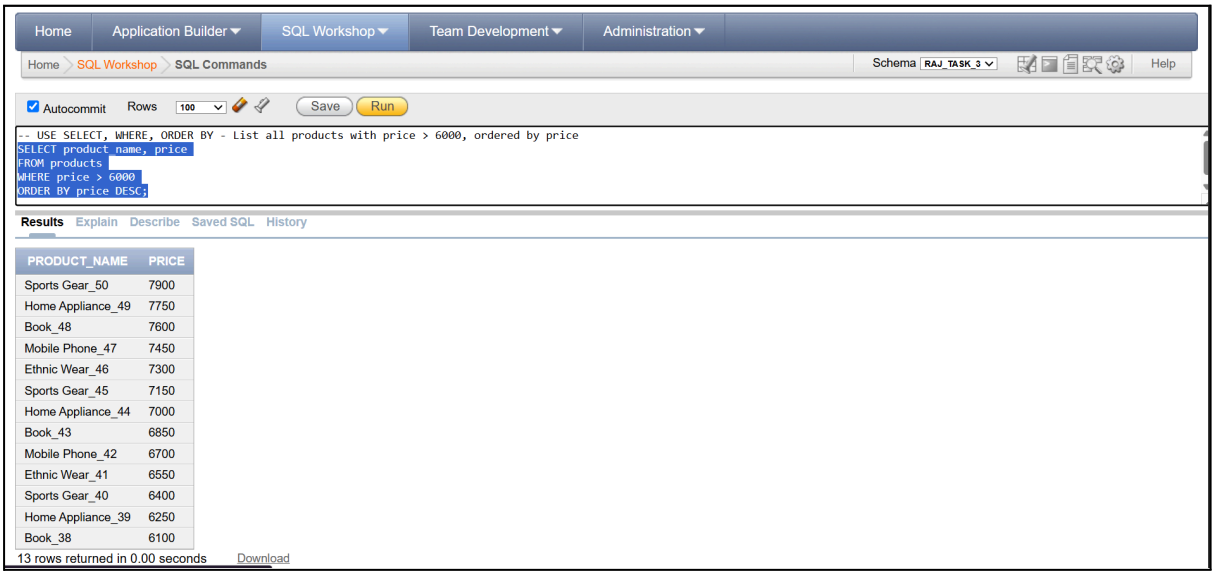
---------- Perform Task and Screenshots ----------

-- 1. USE SELECT, WHERE, ORDER BY List all products with price 6000, ordered by price

```
SELECT
    product_name,
    price
FROM
    products
WHERE
```

```
    price > 6000
ORDER BY
    price DESC;
```

Output:

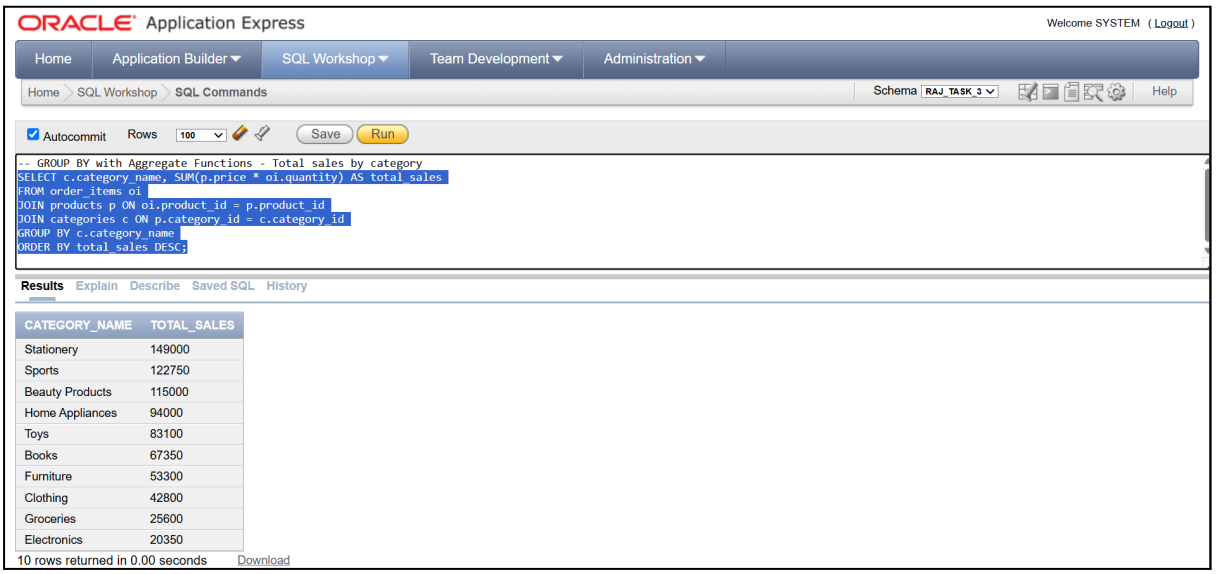


PRODUCT_NAME	PRICE
Sports Gear_50	7900
Home Appliance_49	7750
Book_48	7600
Mobile Phone_47	7450
Ethnic Wear_46	7300
Sports Gear_45	7150
Home Appliance_44	7000
Book_43	6850
Mobile Phone_42	6700
Ethnic Wear_41	6550
Sports Gear_40	6400
Home Appliance_39	6250
Book_38	6100

-- 2. GROUP BY with Aggregate Functions - Total sales by category

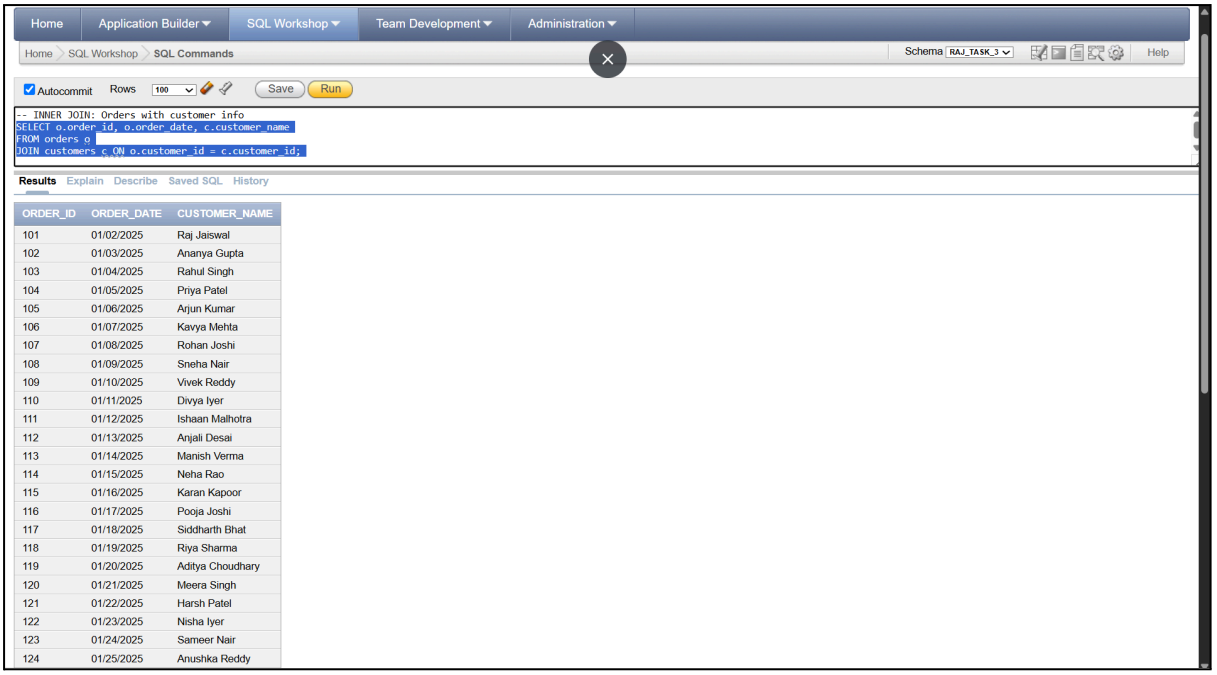
```
SELECT
    c.category_name,
    SUM(p.price * oi.quantity) AS total_sales
FROM
    order_items oi
JOIN
    products p ON oi.product_id = p.product_id
JOIN
    categories c ON p.category_id = c.category_id
GROUP BY
    c.category_name
ORDER BY
    total_sales DESC;
```

Output:



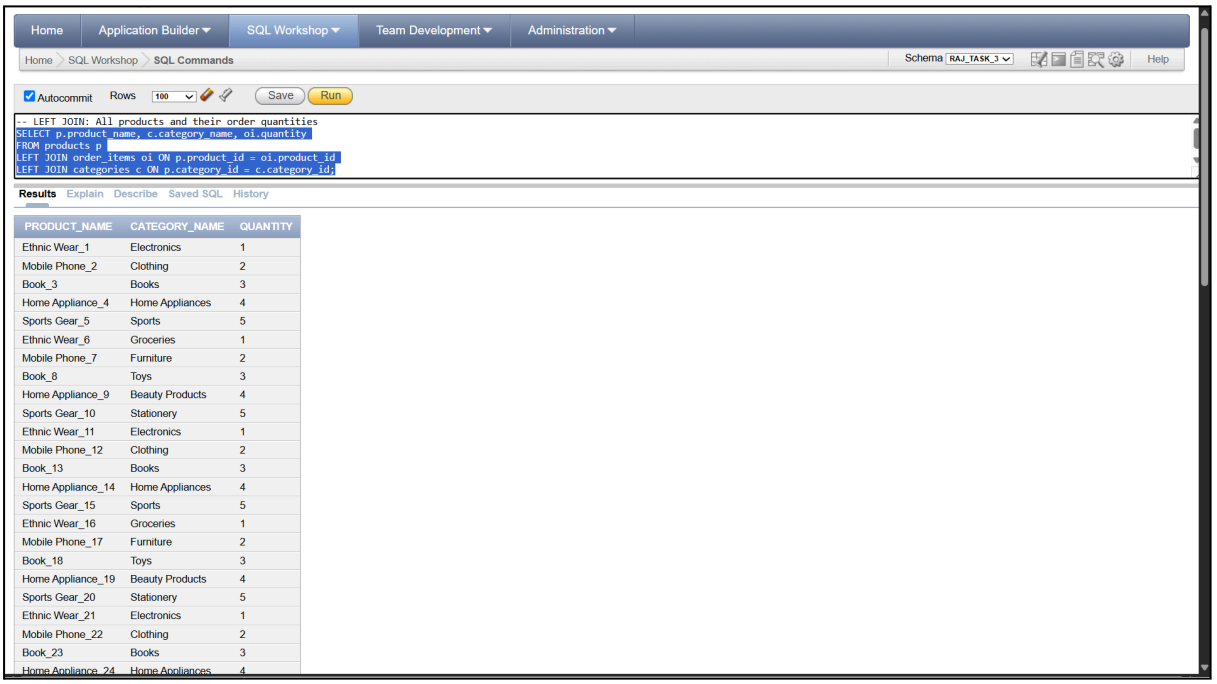

```
/*-------- Use JOINS (INNER, LEFT, RIGHT) --------*/
-- 3.1 INNER JOIN: Orders with customer info
SELECT o.order_id, o.order_date, c.customer_name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id;
```

Output:



```
-- 3.2 LEFT JOIN: All products and their order quantities
SELECT p.product_name, c.category_name, oi.quantity
FROM products p
LEFT JOIN order_items oi ON p.product_id = oi.product_id
LEFT JOIN categories c ON p.category_id = c.category_id;
```

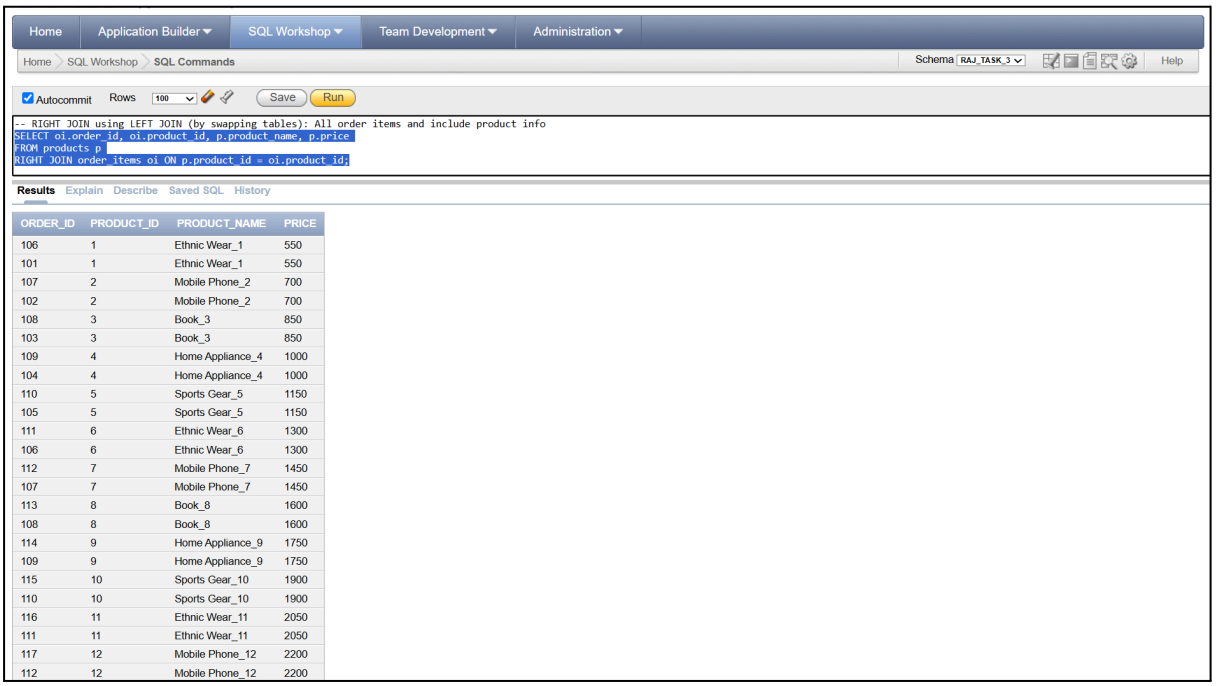
Output:



PRODUCT_NAME	CATEGORY_NAME	QUANTITY
Ethnic Wear_1	Electronics	1
Mobile Phone_2	Clothing	2
Book_3	Books	3
Home Appliance_4	Home Appliances	4
Sports Gear_5	Sports	5
Ethnic Wear_6	Groceries	1
Mobile Phone_7	Furniture	2
Book_8	Toys	3
Home Appliance_9	Beauty Products	4
Sports Gear_10	Stationery	5
Ethnic Wear_11	Electronics	1
Mobile Phone_12	Clothing	2
Book_13	Books	3
Home Appliance_14	Home Appliances	4
Sports Gear_15	Sports	5
Ethnic Wear_16	Groceries	1
Mobile Phone_17	Furniture	2
Book_18	Toys	3
Home Appliance_19	Beauty Products	4
Sports Gear_20	Stationery	5
Ethnic Wear_21	Electronics	1
Mobile Phone_22	Clothing	2
Book_23	Books	3
Home Appliance_24	Home Appliances	4

```
-- 3.3 RIGHT JOIN using LEFT JOIN (by swapping tables):
All order items and include product info
SELECT oi.order_id, oi.product_id, p.product_name,
p.price
FROM products p
RIGHT JOIN order_items oi ON p.product_id =
oi.product_id;
```

Output:

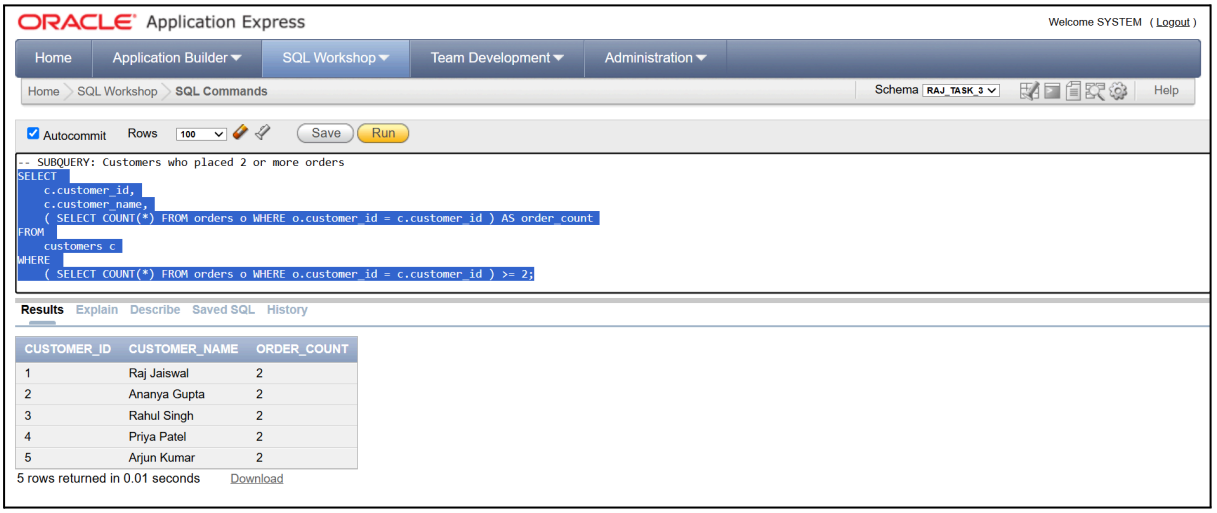


ORDER_ID	PRODUCT_ID	PRODUCT_NAME	PRICE
106	1	Ethnic Wear_1	550
101	1	Ethnic Wear_1	550
107	2	Mobile Phone_2	700
102	2	Mobile Phone_2	700
108	3	Book_3	850
103	3	Book_3	850
109	4	Home Appliance_4	1000
104	4	Home Appliance_4	1000
110	5	Sports Gear_5	1150
105	5	Sports Gear_5	1150
111	6	Ethnic Wear_6	1300
106	6	Ethnic Wear_6	1300
112	7	Mobile Phone_7	1450
107	7	Mobile Phone_7	1450
113	8	Book_8	1600
108	8	Book_8	1600
114	9	Home Appliance_9	1750
109	9	Home Appliance_9	1750
115	10	Sports Gear_10	1900
110	10	Sports Gear_10	1900
116	11	Ethnic Wear_11	2050
111	11	Ethnic Wear_11	2050
117	12	Mobile Phone_12	2200
112	12	Mobile Phone_12	2200

-- 4. SUBQUERY: Customers who placed 2 or more orders

```
SELECT
    c.customer_id,
    c.customer_name,
    ( SELECT COUNT(*) FROM orders o WHERE o.customer_id = c.customer_id ) AS order_count
FROM
    customers c
WHERE
    ( SELECT COUNT(*) FROM orders o WHERE o.customer_id = c.customer_id ) >= 2;
```

Output:



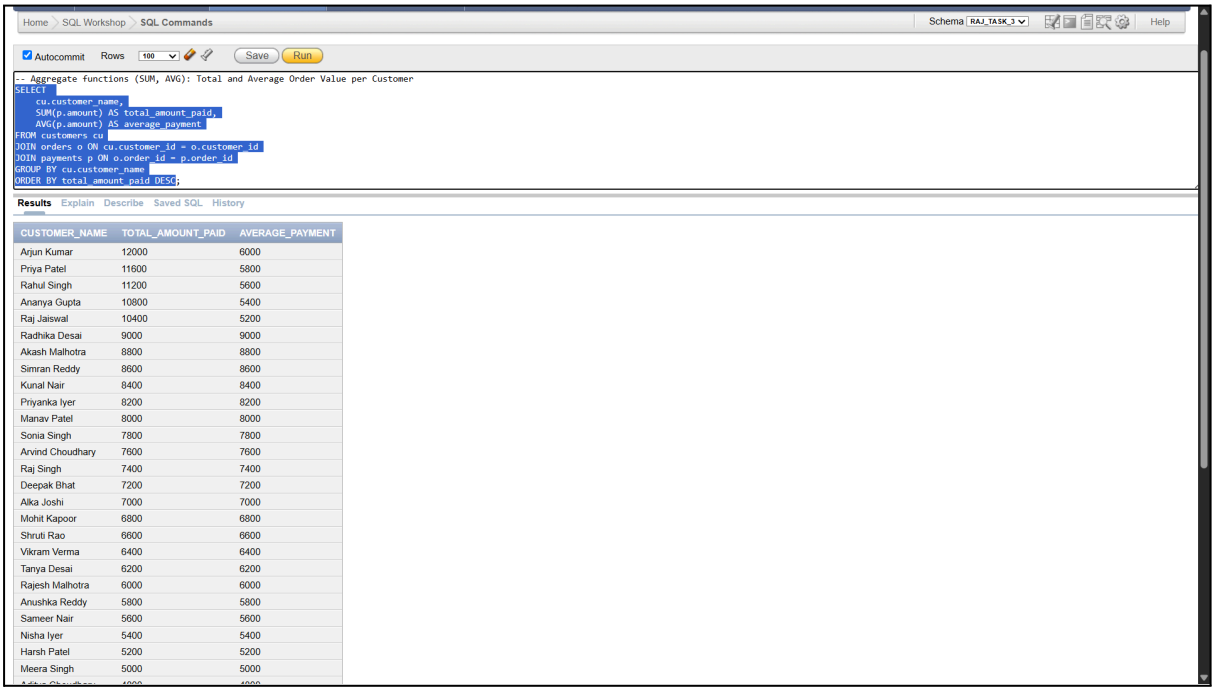
CUSTOMER_ID	CUSTOMER_NAME	ORDER_COUNT
1	Raj Jaiswal	2
2	Ananya Gupta	2
3	Rahul Singh	2
4	Priya Patel	2
5	Arjun Kumar	2

5 rows returned in 0.01 seconds Download

-- 5. Aggregate functions (SUM, AVG): Total and Average Order Value per Customer

```
SELECT
    cu.customer_name,
    SUM(p.amount) AS total_amount_paid,
    AVG(p.amount) AS average_payment
FROM
    customers cu
JOIN
    orders o ON cu.customer_id = o.customer_id
JOIN
    payments p ON o.order_id = p.order_id
GROUP BY
    cu.customer_name
ORDER BY
    total_amount_paid DESC;
```

Output:



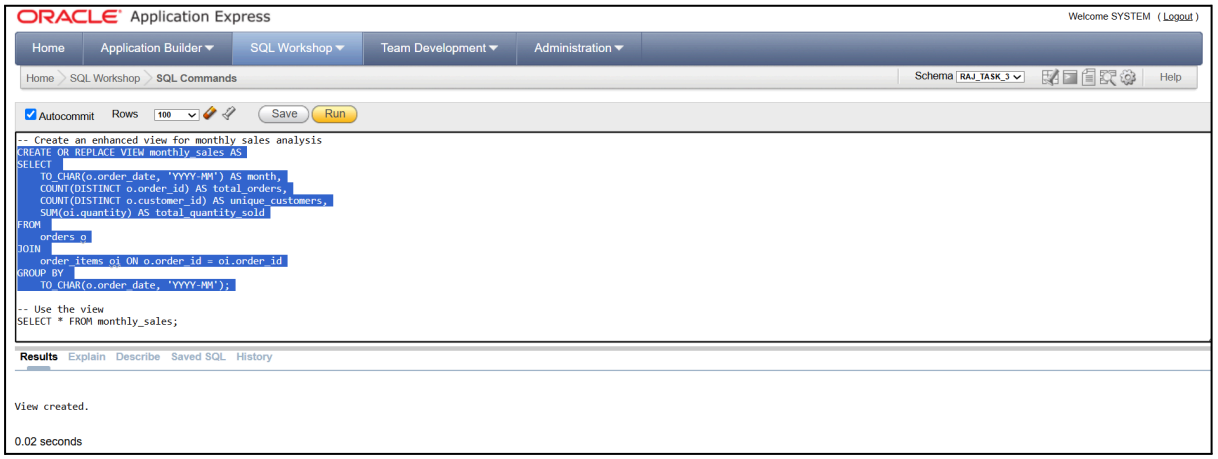
CUSTOMER_NAME	TOTAL_AMOUNT_PAID	AVERAGE_PAYMENT
Arjun Kumar	12000	6000
Priya Patel	11600	5800
Rahul Singh	11200	5600
Ananya Gupta	10800	5400
Raj Jaiswal	10400	5200
Radhika Desai	9000	9000
Akash Malhotra	8800	8800
Simran Reddy	8600	8600
Kunal Nair	8400	8400
Priyanka Iyer	8200	8200
Manav Patel	8000	8000
Sonia Singh	7800	7800
Arvind Choudhary	7600	7600
Raj Singh	7400	7400
Deepak Bhat	7200	7200
Alka Joshi	7000	7000
Mohit Kapoor	6800	6800
Shruti Rao	6600	6600
Vikram Verma	6400	6400
Tanya Desai	6200	6200
Rajesh Malhotra	6000	6000
Anushka Reddy	5800	5800
Sameer Nair	5600	5600
Nisha Iyer	5400	5400
Harsh Patel	5200	5200
Meera Singh	5000	5000

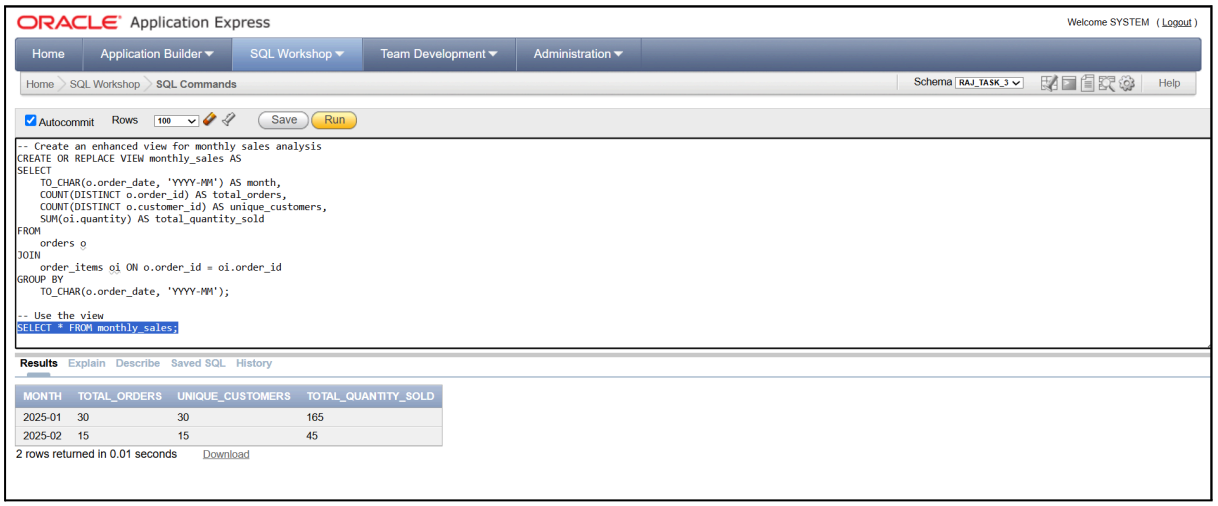
-- 6. Create an enhanced view for monthly sales analysis

```
CREATE OR REPLACE VIEW monthly_sales AS
SELECT
    TO_CHAR(o.order_date, 'YYYY-MM') AS month,
    COUNT(DISTINCT o.order_id) AS total_orders,
    COUNT(DISTINCT o.customer_id) AS unique_customers,
    SUM(oi.quantity) AS total_quantity_sold
FROM
    orders o
JOIN
    order_items oi ON o.order_id = oi.order_id
GROUP BY
    TO_CHAR(o.order_date, 'YYYY-MM');

-- Use the view
SELECT * FROM monthly_sales;
```

Output:





-- 7. Optimize queries with indexes: Create index on frequently searched columns

```
CREATE INDEX idx_products_price ON products(price);
CREATE INDEX idx_orders_customer ON orders(customer_id);

-- View created indexes
SELECT
    index_name,
    table_name,
    column_name
FROM
    user_ind_columns
WHERE
    table_name IN ('PRODUCTS', 'ORDERS');
```

Output: