SQL PROJECT

CASE STUDY: RETAIL STORE SALES ANALYSIS

- P.D.S.SRIYA

Case Study: Retail Store Sales Data Analysis

Background:
A retail chain called "RetailMart" has multiple stores across different regions. The company wants to analyze the sales performance of various products in different stores to make informed business decisions. The database holds the following relevant tables:

Stores:

- Store_ID: Unique identifier for the store
- Store_Name: Name of the store
- Region: Region where the store is located

Products:

- Product_ID: Unique identifier for the product
- Product_Name: Name of the product
- Category: Category to which the product belongs

Sales:

- Sale_ID: Unique identifier for each sale
- Store_ID: Foreign key linking to the Stores table
- Product_ID: Foreign key linking to the Products table
- Sale_Date: Date of the sale
- Quantity: Number of units sold
- Total_Sale_Amount: Total amount for the sale

Creating Tables

Stores Table :

```
CREATE TABLE Stores (
        Store_ID NUMBER PRIMARY KEY,
        Store_Name VARCHAR2(50),
        Region VARCHAR2(50)
);
```

Products Table:

```
CREATE TABLE Products (
        Product_ID NUMBER PRIMARY KEY,
        Product_Name VARCHAR2(100),
        Category VARCHAR2(50)
);
```

Sales Table:

```
CREATE TABLE Sales (
        Sale_ID NUMBER PRIMARY KEY,
        Store_ID NUMBER,
        Product_ID NUMBER,
        Sale_Date DATE,
        Quantity NUMBER,
        Total_Sale_Amount NUMBER(10, 2),
         CONSTRAINT fk_store FOREIGN KEY (Store_ID) REFERENCES
Stores(Store_ID),
         CONSTRAINT fk_product FOREIGN KEY (Product_ID) REFERENCES
Products(Product_ID)
);
```

Inserting Sample Data

Insert Data into the Stores Table:

```
INSERT INTO Stores (Store_ID, Store_Name, Region) VALUES (1, 'RetailMart Downtown', 'North');
INSERT INTO Stores (Store_ID, Store_Name, Region) VALUES (2, 'RetailMart Suburb', 'West');
INSERT INTO Stores (Store_ID, Store_Name, Region) VALUES (3, 'RetailMart Eastside', 'East');
INSERT INTO Stores (Store_ID, Store_Name, Region) VALUES (4, 'RetailMart Uptown', 'South');
```

Insert Data into the Products Table:

```
INSERT INTO Products (Product_ID, Product_Name, Category) VALUES (1, 'Laptop', 'Electronics');
INSERT INTO Products (Product_ID, Product_Name, Category) VALUES (2, 'Smartphone', 'Electronics');
INSERT INTO Products (Product_ID, Product_Name, Category) VALUES (3, 'Jeans', 'Clothing');
INSERT INTO Products (Product_ID, Product_Name, Category) VALUES (4, 'Shirt', 'Clothing');
INSERT INTO Products (Product_ID, Product_Name, Category) VALUES (5, 'Blender', 'Appliances');
```

Insert Data into the Sales Table:

```
INSERT INTO Sales (Sale_ID, Store_ID, Product_ID, Sale_Date, Quantity, Total_Sale_Amount)
VALUES (1, 1, 1, TO_DATE('2024-06-01', 'YYYY-MM-DD'), 5, 2500.00);
INSERT INTO Sales (Sale_ID, Store_ID, Product_ID, Sale_Date, Quantity, Total_Sale_Amount)
VALUES (2, 2, 2, TO_DATE('2024-07-02', 'YYYY-MM-DD'), 10, 5000.00);
INSERT INTO Sales (Sale_ID, Store_ID, Product_ID, Sale_Date, Quantity, Total_Sale_Amount)
VALUES (3, 3, 3, TO_DATE('2024-08-03', 'YYYY-MM-DD'), 20, 800.00);
INSERT INTO Sales (Sale_ID, Store_ID, Product_ID, Sale_Date, Quantity, Total_Sale_Amount)
VALUES (4, 1, 4, TO_DATE('2024-09-04', 'YYYY-MM-DD'), 15, 450.00);
INSERT INTO Sales (Sale_ID, Store_ID, Product_ID, Sale_Date, Quantity, Total_Sale_Amount)
VALUES (5, 4, 5, TO_DATE('2024-10-05', 'YYYY-MM-DD'), 7, 1400.00);
```

```
ALTER TABLE Products ADD Price NUMBER(10, 2);
UPDATE Products SET Price = 10000 WHERE Product_ID = 1;
UPDATE Products SET Price = 5000 WHERE Product_ID = 2;
UPDATE Products SET Price = 1300 WHERE Product_ID = 3;
UPDATE Products SET Price = 1900 WHERE Product_ID = 4;
UPDATE Products SET Price = 2000 WHERE Product_ID = 5;
```

Products Table

PRODUCT ID	PRODUCT NAME	CATEGORY	PRICE
1	Laptop	Electronics	10000
2	Smartphone	Electronics	5000
3	Jeans	Clothing	1300
4	Shirt	Clothing	1900
5	Blender	Appliances	2000

Stores Table

STORE ID	STORE NAME	REGION
4	Retail Mart Uptown	South
1	Retail Mart Downtown	North
2	Retail Mart Subrub	West
3	Retail Mart Eastside	East

Sales Table

SALE ID	STORE ID	PRODUCT ID	SALE DATE	QUANTITY	TOTAL SALE AMOUNT
1	1	1	01- 06 -24	5	2500
2	2	2	02- 07- 24	10	5000
3	3	3	03- 08 -24	20	800
4	1	4	04- 09 -24	15	450
5	4	5	05- 10 -24	7	1400

1Q.The management Wants to Know the total sales in each region.

STORES TABLE

STORE_ID	STORE_NAME	REGION
1	RetailMart Downtown	North
2	RetailMart Suburb	West
3	RetailMart Eastside	East
4	RetailMart Uptown	South

SALES TABLE

SALE_ID	STORE_ID	PRODUCT_ID	SALE_DATE	QUANTITY	TOTAL_SALE_AMOUNT
1	1	1	01-09-23	5	2500
2	2	2	02-09-23	10	5000
3	3	3	03-09-23	20	800
4	1	4	04-09-23	15	450
5	4	5	05-09-23	7	1400

EXPLANATION:-

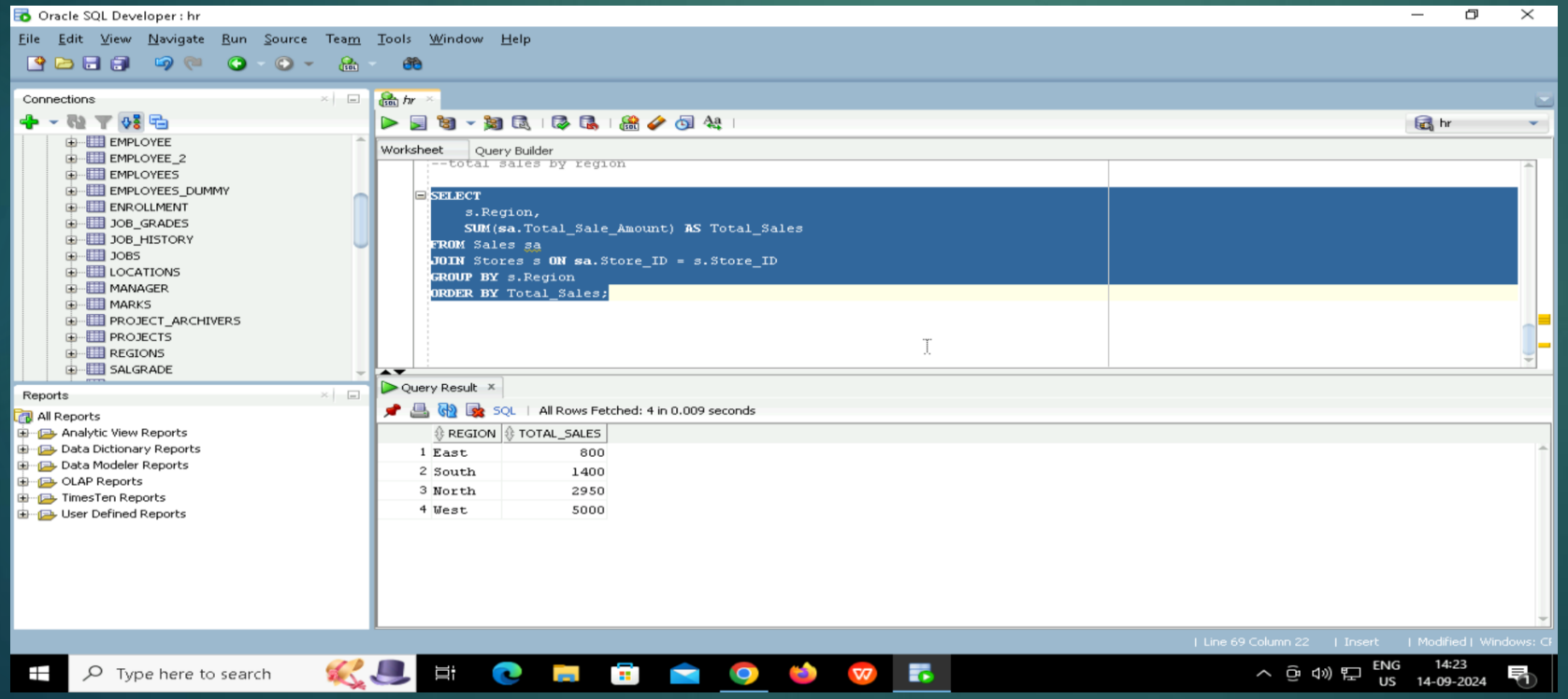
- JOIN is used to combine the Sales table with the Stores table using the common Store_ID.
- SUM(sa.Total_Sale_Amount) calculates the total sales amount for each region.
- GROUP BY s.Region groups the results by region to get the total sales per region.
- ORDER BY Total_Sales DESC sorts the regions by total sales in descending order.

QUERY :-

```
SELECT
        s.Region,
        SUM(sa.Total_sale_Amount) AS Total_sales
        FROM Sales sa
        JOIN Stores s ON sa.Store_ID =  s.Store_ID
        GROUP BY  s.Region
        ORDER BY Total_Sales;
```

OUTPUT : -

REGION	TOTAL_SALES
East	800
South	1400
North	2950
West	5000



2Q. Identify the top 5 best-selling products in terms of sales volume across all stores.

Sales Table

SALE ID	STORE ID	PRODUCT ID	SALE DATE	QUANTITY	TOTAL SALE AMOUNT
1	1	1	01- 09 -23	5	2500
2	2	2	02- 09 - 23	10	5000
3	3	3	03- 09 -23	20	800
4	1	4	04- 09 -23	15	450
5	4	5	05- 09 -23	7	1400

Products Table

PRODUCT ID	PRODUCT NAME	CATEGORY
1	Laptop	Electronics
2	Smartphone	Electronics
3	Jeans	Clothing
4	Shirt	Clothing
5	Blender	Appliances

Explanation:

- **JOIN** COMBINES THE SALES TABLE WITH THE PRODUCTS TABLE USING THE PRODUCT_ID.
- **SUM**(SA.TOTAL_SALE_AMOUNT) CALCULATES THE TOTAL SALES AMOUNT FOR EACH PRODUCT CATEGORY.
- **GROUP BY** P.CATEGORY GROUPS THE RESULTS BY PRODUCT CATEGORY TO AGGREGATE SALES.
- **ORDER BY** TOTAL_SALES DESC SORTS THE CATEGORIES BY TOTAL SALES IN DESCENDING ORDER.

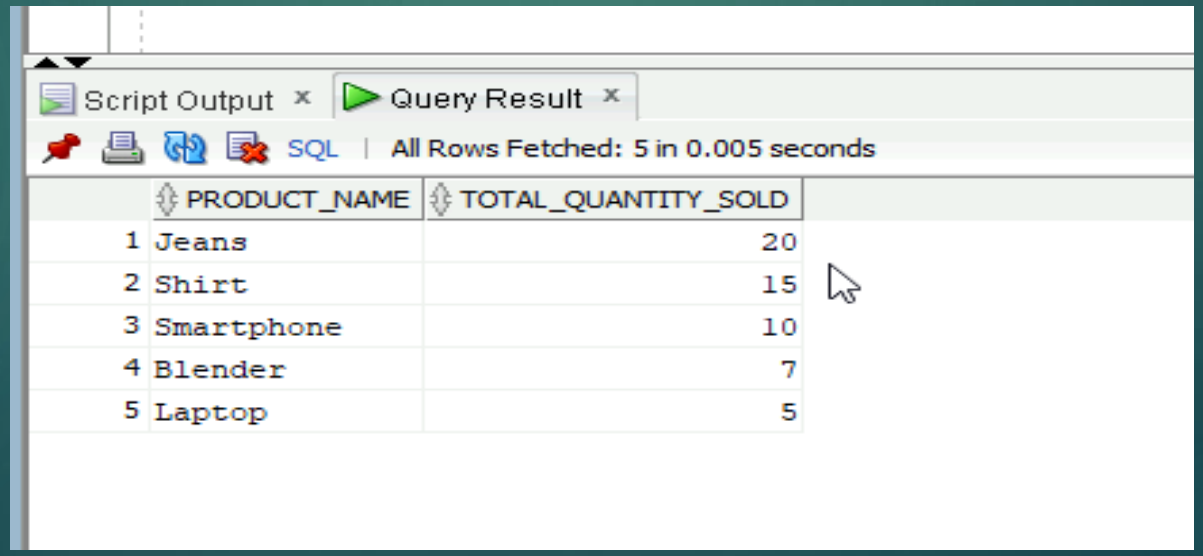
Query:

```
SELECT
  p.Category,
  SUM(sa.Total_Sale_Amount) AS Total_Sales,
  AVG(sa.Total_Sale_Amount) AS Average_Sales_Per_Transaction
FROM
  Sales sa
JOIN
  Products p ON sa.Product_ID = p.Product_ID
GROUP BY
  p.Category
ORDER BY
  Total_Sales DESC;
```

Output

Product Name	Total Quantity Sold
Jeans	20
Shirt	15
Smartphone	10
Blender	7
Laptop	5

Screen shot of Executed Query:



3Q.Category-Wise Sales: Get a breakdown of sales by product category.

SALES TABLE

SALE ID	STORE ID	PRODUCT ID	SALE DATE	QUANTITY	TOTAL SALE AMOUNT
1	1	1	01-09-23	5	2500
2	2	2	02-09-23	10	5000
3	3	3	03-09-23	20	800
4	1	4	04-09-23	15	450
5	4	5	05-09-23	7	1400

PRODUCTS TABLE

PRODUCT ID	PRODUCT NAME	CATEGORY
1	LAPTOP	ELECTRONICS
2	SMARTPHONE	ELECTRONICS
3	JEANS	CLOTHING
4	SHIRT	CLOTHING
5	BLENDER	APPLIANCES

EXPLANATION:-

> JOIN is used to combine the Sales table with the products table using the common PRODUCT_ID.

> SUM(S.Total_Sale_Amount) calculates the total sales amount for each region.

> GROUP BY p.category groups the results by region to get the total sales per region.

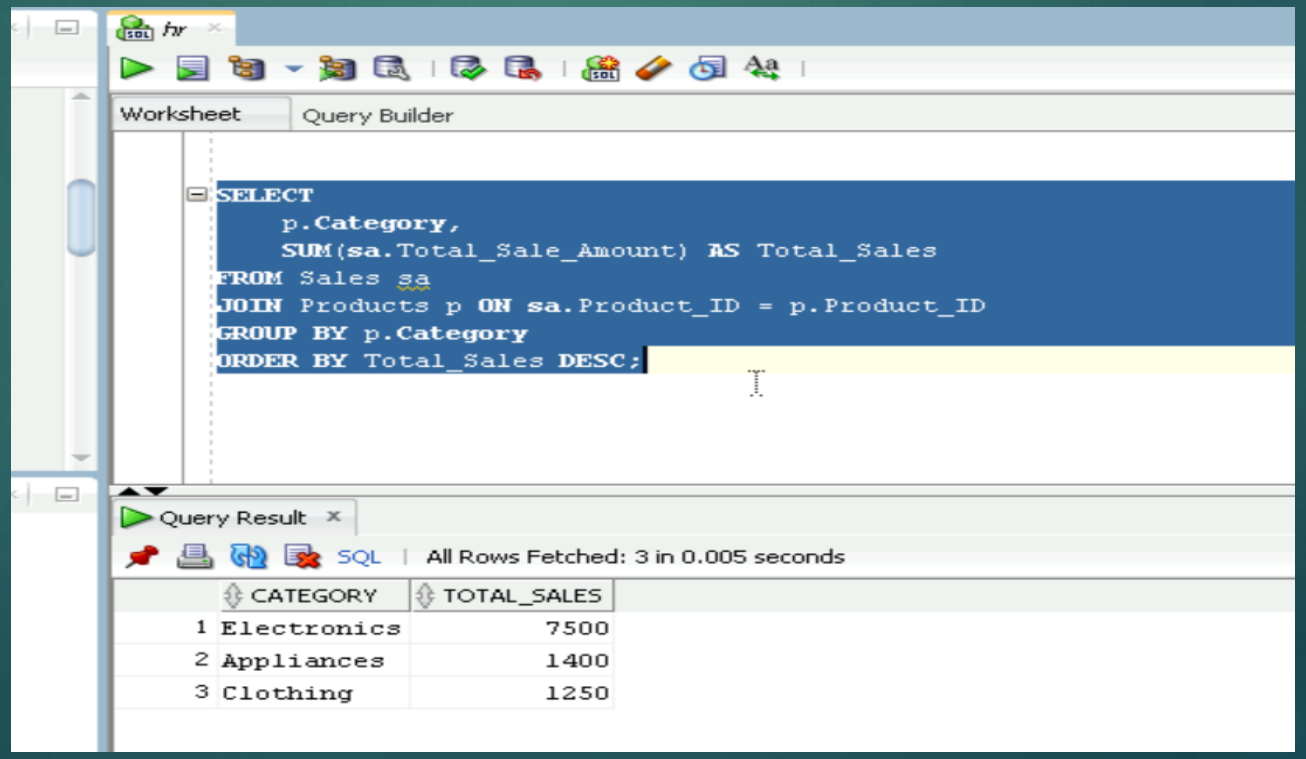
> ORDER BY Total_Sales DESC sorts the regions by total sales in descending order.

QUERY:-

```
SELECT
        P.CATEGORY,
        SUM(S.TOTAL_SALE_AMOUNT) TOTAL_SALES
FROM SALES S
JOIN PRODUCTS P ON S.PRODUCT_ID=P.PRODUCT_ID
GROUP BY P.CATEGORY
ORDER BY TOTAL_SALES DESC;
```

Output:-

Category	Total_sales
ELECTRONICS	7500
APPLIANCES	1400
CLOTHING	1250



4. Monthly Sales Growth: Calculate the monthly sales growth for the current year.

SALES_DATE	TOTAL_SALES _AMOUNT
2024-06-01	2500
2024-07-02	5000
2024-08-03	800
2024-09-04	450
2024-10-05	1400

Month	Total_Month_Sale	Prev_Month_Sal	Monthly_growth_Sale
06-01	2500	NULL	0
07-02	5000	2500	100
08-03	800	5000	-525
09-04	450	800	-77
10-05	1400	450	67

EXPLANATION:

current Subquery:

- This subquery calculates the total sales for each month in the current year.
- TO_CHAR(Sale_Date, 'YYYY-MM') AS Month: Formats the sale date to 'YYYY-MM'.
- SUM(Total_Sale_Amount) AS Total_Monthly_Sales: Sums up the sales amount for each month.
- GROUP BY TO_CHAR(Sale_Date, 'YYYY-MM'): Groups the results by month.

previous Subquery:

- This subquery also calculates the total sales for each month in the current year.
- It is similar to the current subquery but is used to join with the current data to find the previous month's sales.

LEFT JOIN Operation:

- Joins the current month with the previous month.
- ON current.Month = TO_CHAR(ADD_MONTHS(TO_DATE(previous.Month, 'YYYY-MM'), 1), 'YYYY-MM'): Joins each month with the previous month by adding one month to the previous month's date.

- CASE Statement:
- Calculates the monthly growth percentage:
- If there is no data for the previous month (previous.Total_Monthly_Sales IS NULL), it sets the growth percentage to 0.
- Otherwise, it calculates the percentage change using: ROUND(((current.Total_Monthly_Sales - previous.Total_Monthly_Sales) / previous.Total_Monthly_Sales) * 100, 2).

QUERY:

```
SELECT
   current.Month,
   current.Total_Monthly_Sales,
   previous.Total_Monthly_Sales AS Previous_Month_Sales,
   CASE
      WHEN previous.Total_Monthly_Sales IS NULL THEN 0
      ELSE ROUND(((current.Total_Monthly_Sales - previous.Total_Monthly_Sales) / previous.Total_Monthly_Sales) * 100, 2)
   END AS Monthly_Growth_Percentage
FROM (
   SELECT
      TO_CHAR(Sale_Date, 'YYYY-MM') AS Month,
      SUM(Total_Sale_Amount) AS Total_Monthly_Sales
   FROM Sales
   GROUP BY
      TO_CHAR(Sale_Date, 'YYYY-MM')
   ORDER BY Month
   ) current
LEFT JOIN
```

```
(
  SELECT
    TO_CHAR(Sale_Date, 'YYYY-MM') AS Month,
    SUM(Total_Sale_Amount) AS Total_Monthly_Sales
  FROM Sales
  GROUP BY TO_CHAR(Sale_Date, 'YYYY-MM')
  ORDER BY Month
  ) previous ON
  current.Month = TO_CHAR(ADD_MONTHS(TO_DATE(previous.Month, 'YYYY-MM'), 1), 'YYYY-MM');
```

5.Determine the performance of each store by comparing sales volumes.

Store_ID	Store_Name	Region
1	Retail Mart Downtown	North
2	Retail Mart Suburb	West
3	Retail Mart Eastside	East
4	Retail mart Uptown	South

Sale_ID	Store_ID	Product_ID	Sale_Date	Quantity	Total_Sale_Amount
1	1	1	2023-09-01	5	2500.00
2	2	2	2023-09-02	10	5000.00
3	3	3	2023-09-03	20	800.00
4	1	4	2023-09-04	15	450.00
5	4	5	2023-09-05	7	1400.00

Joins, Aggregate Functions, Group By and Order By Clauses

- Joins in SQL are used to combine rows from two or more tables based on a related column between them.
- Aggregate functions in SQL are used to perform calculations on a set of values and return a single value.
- Group By Clause groups the rows sharing the same values in specified columns.
- Order By Clause sorts the result set by one or more columns in ascending or descending order to present data in a specified sequence.

QUERY:

```
SELECT
    s.Store_Name,
    SUM(sa.Total_Sale_Amount) AS Total_Sales
FROM
    Sales sa
JOIN
    Stores s ON sa.Store_ID = s.Store_ID
GROUP BY
    s.Store_Name
ORDER BY
    Total_Sales DESC;
```

OUTPUT:

Store_Name	Total_Sales
Retail Mart Suburb	5000
Retail Mart Downtown	2950
Retail Mart Uptown	1400
Retail Mart Eastside	800

```sql
SELECT
    s.store_name,
    SUM(sa.TOTAL_SALE_AMOUNT) AS TOTAL_SALES
FROM
    SALES sa
JOIN STORES s ON s.store_id=sa.store_id
GROUP BY
    s.store_name
ORDER BY
    TOTAL_SALES DESC
```

Query Result

All Rows Fetched: 4 in 0.1 seconds

	STORE_NAME	TOTAL_SALES
1	RetailMart Suburb	5000
2	RetailMart Downtown	2950
3	RetailMart Uptown	1400
4	RetailMart Eastside	800

6.Product Category Insights: Offer discounts or promotions based on category-wise sales trends.

PRODUCT_ID	PRODUCT_NAME	CATEGORY	PRICE
1	Laptop	Electronics	10000
2	SmartPhone	Electronics	5000
3	Jeans	Clothings	1300
4	Shirt	Clothings	1900
5	Blender	Appliaces	2000

QUERY:

```
SELECT
    Product_ID,
    Product_Name,
    Category,
    Price,
    CASE
        WHEN Price > 3000 THEN Price * 0.2
        ELSE Price * 1
    END AS DISCOUNT
FROM Products;
```

OUTPUT:

Oracle SQL Developer : hr

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections
- hr
  - Tables (Filtered)
  - Views
  - Indexes
  - Packages
  - Procedures
  - Functions
  - Operators
  - Queues
  - Queues Tables
  - Triggers
  - Types
  - Sequences

Reports
- All Reports
  - Analytic View Reports
  - Data Dictionary Reports
  - Data Modeler Reports
  - OLAP Reports
  - TimesTen Reports
  - User Defined Reports

hr

Worksheet | Query Builder

```
SELECT
        Product_ID,
        Product_Name,
        Category,
        Price,
        CASE
            WHEN Price > 3000 THEN Price * 0.2
            ELSE Price * 1
        END AS DISCOUNT
FROM Products;
```

Query Result

SQL | All Rows Fetched: 5 in 0.007 seconds

	PRODUCT_ID	PRODUCT_NAME	CATEGORY	PRICE	DISCOUNT
1	1	Laptop	Electronics	10000	2000
2	2	Smartphone	Electronics	5000	1000
3	3	Jeans	Clothing	1300	1300
4	4	Shirt	Clothing	1900	1900
5	5	Blender	Appliances	2000	2000

| Line 63 Column 1 | Insert | Modified | Windows: CR

Type here to search

ENG US 20:37 15-09-2024

7.Regional Trends: Analyze trends over time by region to inform marketing strategies.

- Selecting and Formatting Data
- Joining Tables
- Filtering Data:
- Grouping Data:
- Ordering Results:

QUERY

```
SELECT
  TO_CHAR(s.Sale_Date, 'YYYY-MM') AS Month,
  st.Region,
  SUM(s.Total_Sale_Amount) AS Total_Sales
FROM Sales s
JOIN Stores st ON s.Store_ID = st.Store_ID
GROUP BY
  TO_CHAR(s.Sale_Date, 'YYYY-MM'),
  st.Region
ORDER BY
  Month,
  st.Region;
```

OUTPUT:

Oracle SQL Developer : hr

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections
- hr
  - Tables (Filtered)
  - Views
  - Indexes
  - Packages
  - Procedures
  - Functions
  - Operators
  - Queues
  - Queues Tables
  - Triggers
  - Types
  - Sequences

Reports

All Reports
- Analytic View Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

hr

Worksheet | Query Builder

```sql
SELECT
    TO_CHAR(s.Sale_Date, 'YYYY-MM') AS Month,
    st.Region,
    SUM(s.Total_Sale_Amount) AS Total_Sales
FROM
    Sales s
JOIN
    Stores st ON s.Store_ID = st.Store_ID
WHERE
    EXTRACT(YEAR FROM s.Sale_Date) = EXTRACT(YEAR FROM SYSDATE)
GROUP BY
    TO_CHAR(s.Sale_Date, 'YYYY-MM'),
    st.Region
ORDER BY
    Month,
    st.Region;
```

Query Result

SQL | All Rows Fetched: 5 in 0.005 seconds

	MONTH	REGION	TOTAL_SALES
1	2024-06	North	2500
2	2024-07	West	5000
3	2024-08	East	800
4	2024-09	North	450
5	2024-10	South	1400

Line 135 Column 17 | Insert | Modified | Windows: CI

Type here to search

ENG US 23:03 15-09-2024

THANK YOU