EXP 1 : Case Study: Building a Sales Data Warehouse/Data Mart

Statement of Problem: A retail company, XYZ Retailers, is experiencing difficulties in analyzing and reporting on their sales data. The company operates multiple stores across the country and has been accumulating large volumes of transactional data over the years. Their current systems are unable to efficiently handle this data, resulting in slow query performance, limited reporting capabilities, and difficulties in gaining valuable insights into their business.

The problems they are facing include:

- Slow query performance: The existing database infrastructure is struggling to provide timely insights into sales trends and customer behaviors due to the sheer volume of data.
- Data silos: Data is stored in various formats and locations, making it challenging to consolidate and analyze information effectively.
- Lack of historical data analysis: The company cannot perform trend analysis, year-over-year comparisons, or forecast future sales due to limited historical data access.
- Inefficient reporting: Business analysts have difficulties generating accurate and timely reports for decision-making.
- Inconsistent data quality: Data quality issues such as duplicate records and missing information hinder accurate reporting.

To address these problems, XYZ Retailers has decided to build a Sales Data Warehouse and Data Mart to centralize their data, improve data quality, and provide a comprehensive platform for analytics and reporting.

Creating a Dimensional Model (Star Schema):

A star schema is a common design for data warehouses, and it will serve as the foundation for the XYZ Retailers Sales Data Warehouse. In this schema, you have a central fact table surrounded by dimension tables. Here's a simplified version of a star schema for XYZ Retailers:

1. Fact Table: Sales Transactions (Fact_Sales)

- SalesID (Primary Key)
- DateKey (Foreign Key)
- ProductKey (Foreign Key)
- StoreKey (Foreign Key)
- CustomerKey (Foreign Key)
- SalesAmount
- QuantitySold
-

2. Dimension Tables:

a. Date Dimension (Dim_Date)

- DateKey (Primary Key)
- Date
- DayOfWeek
- Month
- Quarter
- Year

b. Product Dimension (Dim_Product)

- ProductKey (Primary Key)
- ProductID
- ProductName
- Category
- Brand • Supplier

c. Store Dimension (Dim_Store)

- StoreKey (Primary Key)
- StoreID
- StoreName
- City • Region

d. Customer Dimension (Dim_Customer)

- CustomerKey (Primary Key)
- CustomerID
- FirstName
- LastName
- Email
- Phone

Creating a Snowflake Schema:

In a snowflake schema, the dimension tables are normalized, which can reduce redundancy but also complicate queries. In this case, we'll start with the star schema and normalize the product dimension table:

1) Dimension Tables (Snowflake Schema): a.

Date Dimension (Dim_Date)

Same as in the star schema

b. Product Dimension (Dim_Product)

- ProductKey (Primary Key)
- ProductID
- ProductName
- CategoryKey (Foreign Key)
- BrandKey (Foreign Key)
- SupplierKey (Foreign Key)

c. Category Dimension (Dim_Category)

- CategoryKey (Primary Key)
- CategoryName

d. Brand Dimension (Dim_Brand)

- BrandKey (Primary Key)

- BrandName

e. Supplier Dimension (Dim_Supplier)

- SupplierKey (Primary Key)
- SupplierName

f. Store Dimension (Dim_Store)

- Same as in the star schema

g. Customer Dimension (Dim_Customer)

- Same as in the star schema

The snowflake schema reduces data redundancy by normalizing the product information into separate tables. However, it may require more complex queries with additional joins.

This dimensional model can serve as the foundation for designing and building the Sales Data Warehouse and Data Mart for XYZ Retailers, allowing them to efficiently store, manage, and analyze their sales data.

Exp 2: Implementation of all dimension tables and fact tables related to case study mentioned in the first experiment. In this experiment , we write the SQL queries for, 1.Creating the dimension table 2.Creating the Fact table 3. Inserting values in both tables. 4. Displaying the tables. 5.Draw the Fact Constellation Schema.

Creating Dimension Tables:

```
-- Create Date Dimension
CREATE TABLE Dim_Date (
    DateKey INT PRIMARY KEY,
    Date DATE,
    DayOfWeek VARCHAR(10),
    Month VARCHAR(10),
    Quarter VARCHAR(10),
    Year INT
);
-- Create Product Dimension
CREATE TABLE Dim_Product (
```

```
    ProductKey INT PRIMARY KEY,
    ProductID VARCHAR(20),
    ProductName VARCHAR(255),
    Category VARCHAR(50),
Brand VARCHAR(50),
    Supplier VARCHAR(100)
);
-- Create Store Dimension
CREATE TABLE Dim_Store (
    StoreKey INT PRIMARY KEY,
    StoreID VARCHAR(20),
    StoreName VARCHAR(255),
    City VARCHAR(50),
    Region VARCHAR(50)
);
-- Create Customer Dimension
CREATE TABLE Dim_Customer (
    CustomerKey INT PRIMARY KEY,
    CustomerID VARCHAR(20),
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    Phone VARCHAR(20)
);
```

Creating Fact Table:

```
-- Create Sales Fact Table
CREATE TABLE Fact_Sales (
    SalesID INT PRIMARY KEY,
    DateKey INT,
    ProductKey INT,
    StoreKey INT,
    CustomerKey INT,
    SalesAmount DECIMAL(10, 2),
    QuantitySold INT
);
```

Inserting Values:

```
-- Insert values into Date Dimension
INSERT INTO Dim_Date (DateKey, Date, DayOfWeek, Month, Quarter, Year)
VALUES
    (1, '2023-01-01', 'Sunday', 'January', 'Q1', 2023),
    (2, '2023-01-02', 'Monday', 'January', 'Q1', 2023),
    -- Add more date records here...

-- Insert values into Product Dimension
INSERT INTO Dim_Product (ProductKey, ProductID, ProductName, Category, Brand, Supplier)
VALUES
    (1, 'P001', 'Product 1', 'Electronics', 'Brand A', 'Supplier X'),
    (2, 'P002', 'Product 2', 'Clothing', 'Brand B', 'Supplier Y'),
    -- Add more product records here...

-- Insert values into Store Dimension
INSERT INTO Dim_Store (StoreKey, StoreID, StoreName, City, Region)
VALUES
```

```
    (1, 'S001', 'Store 1', 'New York', 'East'),
(2, 'S002', 'Store 2', 'Los Angeles', 'West'),
    -- Add more store records here...

-- Insert values into Customer Dimension
INSERT INTO Dim_Customer (CustomerKey, CustomerID, FirstName, LastName, Email, Phone)
VALUES
    (1, 'C001', 'John', 'Doe', 'johndoe@email.com', '123-456-7890'),
(2, 'C002', 'Jane', 'Smith', 'janesmith@email.com', '987-654-3210');
    -- Add more customer records here...

-- Insert values into Sales Fact Table
INSERT INTO Fact_Sales (SalesID, DateKey, ProductKey, StoreKey, CustomerKey, SalesAmount, QuantitySold)
VALUES
    (1, 1, 1, 1, 1, 100.00, 2),
    (2, 2, 2, 2, 2, 150.00, 3);
    -- Add more sales records here...
```

Displaying Tables:

```
-- Display contents of the Date Dimension table
SELECT * FROM Dim_Date;

-- Display contents of the Product Dimension table
SELECT * FROM Dim_Product;

-- Display contents of the Store Dimension table
SELECT * FROM Dim_Store;

-- Display contents of the Customer Dimension table
SELECT * FROM Dim_Customer;

-- Display contents of the Sales Fact table
```

SELECT * FROM Fact_Sales;

Exp 3: Implementation of OLAP operation in SQL Environment

Create a Sample Sales Data Table

```
CREATE TABLE Sales (
    SalesDate DATE,
    ProductCategory VARCHAR(50),
    Revenue DECIMAL(10, 2)
);
-- Insert sample data
INSERT INTO Sales (SalesDate, ProductCategory, Revenue)
VALUES
    ('2023-01-01', 'Electronics', 1000.00),
    ('2023-01-01', 'Clothing', 500.00),
    ('2023-01-02', 'Electronics', 800.00),
    ('2023-01-02', 'Clothing', 600.00),
    -- Add more sales records here...
```

1. Drill Down

```
-- Drill down from Month to Day
SELECT Date, SUM(SalesAmount) AS TotalSales
FROM Sales
WHERE Date BETWEEN '2023-01-01' AND '2023-01-31'
GROUP BY Date
ORDER BY Date;
```

2. Rollup

```
-- Roll up from Day to Month
SELECT DATE_FORMAT(Date, '%Y-%m') AS Month, SUM(SalesAmount) AS TotalSales
FROM Sales
GROUP BY Month
ORDER BY Month;
```

3.slice

```
-- Slice for 'Electronics' Product Category
SELECT ProductCategory, SUM(SalesAmount) AS TotalSales
FROM Sales
WHERE ProductCategory = 'Electronics'
GROUP BY ProductCategory;
```

4.Dice:

```
-- Dice for 'Electronics' in 'West' Region
SELECT Region, SUM(SalesAmount) AS TotalSales
FROM Sales
WHERE ProductCategory = 'Electronics' AND Region = 'West'
GROUP BY Region;
```

5. Pivot

```
-- Pivot to view by Product Category and Region
SELECT ProductCategory, Region, SUM(SalesAmount) AS TotalSales
FROM Sales
GROUP BY ProductCategory, Region;
```

EXP 4 :
Implementation of Data Discretization (any one) & Visualization (any one) dbms

```
-- Create the EmployeeData table
CREATE TABLE EmployeeData (
    Salary DECIMAL(10, 2)
);
-- Insert sample values into the EmployeeData table
```

```
INSERT INTO EmployeeData (Salary) VALUES
  (40000.00),
  (60000.00),
  (55000.00),
  (75000.00),
  (35000.00),
  (90000.00),
  (42000.00),
  (58000.00);



-- Create a temporary table with discretized data
CREATE TEMPORARY TABLE DiscretizedData AS
SELECT
  CASE
    WHEN Salary < 40000 THEN 'Low'
    WHEN Salary >= 40000 AND Salary < 80000 THEN 'Medium'
    ELSE 'High'    END
AS IncomeBracket
FROM EmployeeData;

-- Count the number of employees in each income bracket
SELECT IncomeBracket, COUNT(*) AS Count
FROM DiscretizedData
GROUP BY IncomeBracket;
```

Exp 5: Implementation of Naïve Bayes Algorithm using any programming language like Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
data = {
'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast',
'Sunny', 'Sunny',
'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
'Temp': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool',
'Mild', 'Mild', 'Mild',
'Hot', 'Mild'],
'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',
'High', 'Normal',
'Normal', 'Normal', 'High', 'Normal', 'High'],
'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
'Weak', 'Weak',
'Strong', 'Strong', 'Weak', 'Strong'],
'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',
'Yes', 'Yes', 'Yes', 'No']
}
df = pd.DataFrame(data)
df_encoded = pd.get_dummies(df.iloc[:, :-1])
X = df_encoded.values
y = df['Play'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=87)
gaussian_naive_bayes = GaussianNB()
gaussian_naive_bayes.fit(X_train, y_train)
y_pred = gaussian_naive_bayes.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
outlook_input = input("Enter outlook (Sunny/Overcast/Rain): ")
temp_input = input("Enter temperature (Hot/Mild/Cool): ")
humidity_input = input("Enter humidity (High/Normal): ")
wind_input = input("Enter wind (Weak/Strong): ")
user_input = pd.DataFrame({
'Outlook_' + outlook_input: 1,
'Temp_' + temp_input: 1,
'Humidity_' + humidity_input: 1,
'Wind_' + wind_input: 1
}, index=[0])
user_input_encoded = user_input.reindex(columns=df_encoded.columns,
fill_value=0)
prediction = gaussian_naive_bayes.predict(user_input_encoded.values)
print(prediction)
```

EXPERIMENT NO. 7

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class KMeans {

    public static void main(String[] args) {
        // Number of clusters
        int k = 3;

        // Number of data points
        int n = 100;

        // Generate some random data points
        List<Point> data = generateData(n);

        // Initialize the centroids
        List<Point> centroids = initializeCentroids(data, k);

        // K-Means clustering
        List<List<Point>> clusters = kMeans(data, centroids, k, 100);

        // Print the clusters
        for (int i = 0; i < k; i++) {
            System.out.println("Cluster " + (i + 1));
            for (Point point : clusters.get(i)) {
                System.out.println(point);
            }
            System.out.println();
        }
    }

    public static List<Point> generateData(int n) {
        List<Point> data = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < n; i++) {
            double x = random.nextDouble() * 100;
            double y = random.nextDouble() * 100;
            data.add(new Point(x, y));
        }
        return data;
    }

    public static List<Point> initializeCentroids(List<Point> data, int k) {
        List<Point> centroids = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < k; i++) {
```

```
            int randomIndex = random.nextInt(data.size());
            centroids.add(data.get(randomIndex));
        }
        return centroids;
    }

    public static List<List<Point>> kMeans(List<Point> data, List<Point>
centroids, int k, int maxIterations) {
        List<List<Point>> clusters = new ArrayList<>();

        for (int iteration = 0; iteration < maxIterations; iteration++) {
            // Initialize empty clusters
            clusters.clear();
            for (int i = 0; i < k; i++) {
                clusters.add(new ArrayList<>());
            }

            // Assign data points to the nearest centroid
            for (Point point : data) {
                int nearestCentroidIndex = findNearestCentroid(point,
centroids);
                clusters.get(nearestCentroidIndex).add(point);
            }

            // Update centroids
            for (int i = 0; i < k; i++) {
                centroids.set(i, calculateCentroid(clusters.get(i)));
            }
        }

        return clusters;
    }

    public static int findNearestCentroid(Point point, List<Point> centroids)
{
        int nearestCentroidIndex = 0;
        double minDistance = Double.MAX_VALUE;
        for (int i = 0; i < centroids.size(); i++) {
            double distance = point.distanceTo(centroids.get(i));
            if (distance < minDistance) {
                minDistance = distance;
                nearestCentroidIndex = i;
            }
        }
        return nearestCentroidIndex;
    }

    public static Point calculateCentroid(List<Point> cluster) {
```

```
        double sumX = 0.0;
        double sumY = 0.0;
        for (Point point : cluster) {
            sumX += point.getX();
            sumY += point.getY();
        }
        int size = cluster.size();
        return new Point(sumX / size, sumY / size);
    }

    static class Point {
        private double x;
        private double y;

        public Point(double x, double y) {
            this.x = x;
            this.y = y;
        }

        public double getX() {
            return x;
        }

        public double getY() {
            return y;
        }

        public double distanceTo(Point other) {
            double dx = x - other.x;
            double dy = y - other.y;
            return Math.sqrt(dx * dx + dy * dy);
        }

        @Override
        public String toString() {
            return "(" + x + ", " + y + ")";
        }
    }
}
```

EXPERIMENT NO . 10

```
def calc_pagerank(nodes, df=0.85, iterations = 20):
    num_nodes = len(nodes)
    pagerank = {node : 1.0 for node in nodes}

    for _ in range (iterations):
```

```
        new_pr={}
        for node in nodes:
            new_pr[node] = (1 - df ) / num_nodes
            for other_node, properties in nodes.items():
                if other_node != node and properties ['incoming_links'] > 0:
                    new_pr[node] += (df * pagerank[other_node] /
properties['incoming_links'])
        pagerank = new_pr

    return pagerank

nodes = {
    'A' : {'out_links': 2, 'incoming_links': 1},
    'B': {'out_links': 1, 'incoming_links': 1},
    'C': {'out_links': 1, 'incoming_links': 3},
    'D': {'out_links': 1, 'incoming_links': 0}
}

pr_values = calc_pagerank(nodes)

for node, pr in pr_values.items():
    print(f"PR({node}) = {pr:.4f}")
```

OLAP:

```
CREATE TABLE time_dimension (
    date_id DATE PRIMARY KEY,
    day_of_week VARCHAR(15),
    month VARCHAR(15),
    year INT
);

-- Product Dimension
CREATE TABLE product_dimension (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10,2)
);

-- Location Dimension
CREATE TABLE location_dimension (
    location_id INT PRIMARY KEY,
    city VARCHAR(50),
    state VARCHAR(50),
    country VARCHAR(50)
);

-- Customer Dimension
CREATE TABLE customer_dimension (
    customer_id INT PRIMARY KEY,
```

```
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone VARCHAR(15)
);

-- Salesperson Dimension
CREATE TABLE salesperson_dimension (
    salesperson_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    region VARCHAR(50)
);

-- Payment Method Dimension
CREATE TABLE payment_method_dimension (
    payment_method_id INT PRIMARY KEY,
    payment_method_name VARCHAR(50)
);

-- Promotion Dimension
CREATE TABLE promotion_dimension (
    promotion_id INT PRIMARY KEY,
    promotion_name VARCHAR(100),
    start_date DATE,
    end_date DATE
);

-- Store Dimension
CREATE TABLE store_dimension (
    store_id INT PRIMARY KEY,
    store_name VARCHAR(100),
    city VARCHAR(50),
    state VARCHAR(50),
    country VARCHAR(50)
);

-- Employee Dimension
CREATE TABLE employee_dimension (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    job_title VARCHAR(50)
);

-- Shipper Dimension
CREATE TABLE shipper_dimension (
    shipper_id INT PRIMARY KEY,
    company_name VARCHAR(100),
    contact_name VARCHAR(50),
    contact_phone VARCHAR(15)
);

-- Supplier Dimension
CREATE TABLE supplier_dimension (
    supplier_id INT PRIMARY KEY,
    supplier_name VARCHAR(100),
    city VARCHAR(50),
```

```
    state VARCHAR(50),
    country VARCHAR(50)
);

-- Fact Table for Sales
CREATE TABLE sales_fact (
    date_id DATE,
    product_id INT,
    location_id INT,
    customer_id INT,
    salesperson_id INT,
    payment_method_id INT,
    promotion_id INT,
    store_id INT,
    employee_id INT,
    shipper_id INT,
    supplier_id INT,
    quantity INT,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (date_id) REFERENCES time_dimension(date_id),
    FOREIGN KEY (product_id) REFERENCES product_dimension(product_id),
    FOREIGN KEY (location_id) REFERENCES location_dimension(location_id),
    FOREIGN KEY (customer_id) REFERENCES customer_dimension(customer_id),
    FOREIGN KEY (salesperson_id) REFERENCES
salesperson_dimension(salesperson_id),
    FOREIGN KEY (payment_method_id) REFERENCES
payment_method_dimension(payment_method_id),
    FOREIGN KEY (promotion_id) REFERENCES promotion_dimension(promotion_id),
    FOREIGN KEY (store_id) REFERENCES store_dimension(store_id),
    FOREIGN KEY (employee_id) REFERENCES employee_dimension(employee_id),
    FOREIGN KEY (shipper_id) REFERENCES shipper_dimension(shipper_id),
    FOREIGN KEY (supplier_id) REFERENCES supplier_dimension(supplier_id)
);


DATA:

-- Insert data into time_dimension
INSERT INTO time_dimension (date_id, day_of_week, month, year)
VALUES
    ('2023-10-31', 'Monday', 'October', 2023),
    ('2023-11-01', 'Tuesday', 'November', 2023),
    ('2023-11-02', 'Wednesday', 'November', 2023);

-- Insert data into product_dimension
INSERT INTO product_dimension (product_id, product_name, category, price)
VALUES
    (1, 'Product A', 'Category X', 50.00),
    (2, 'Product B', 'Category Y', 40.00),
    (3, 'Product C', 'Category Z', 60.00);

-- Insert data into location_dimension
INSERT INTO location_dimension (location_id, city, state, country)
VALUES
    (1, 'New York', 'NY', 'USA'),
    (2, 'Los Angeles', 'CA', 'USA'),
    (3, 'London', NULL, 'UK');
```

```
-- Insert data into customer_dimension
INSERT INTO customer_dimension (customer_id, first_name, last_name, email,
phone)
VALUES
    (1, 'John', 'Doe', 'john.doe@example.com', '555-1234'),
    (2, 'Jane', 'Smith', 'jane.smith@example.com', '555-5678'),
    (3, 'Bob', 'Johnson', 'bob.johnson@example.com', '555-9876');

-- Insert data into salesperson_dimension
INSERT INTO salesperson_dimension (salesperson_id, first_name, last_name,
region)
VALUES
    (1, 'Alice', 'Brown', 'East'),
    (2, 'Charlie', 'Green', 'West'),
    (3, 'Eve', 'Black', 'North');

-- Insert data into payment_method_dimension
INSERT INTO payment_method_dimension (payment_method_id,
payment_method_name)
VALUES
    (1, 'Credit Card'),
    (2, 'PayPal'),
    (3, 'Cash');

-- Insert data into promotion_dimension
INSERT INTO promotion_dimension (promotion_id, promotion_name, start_date,
end_date)
VALUES
    (1, 'Summer Sale', '2023-07-01', '2023-08-31'),
    (2, 'Holiday Discount', '2023-11-15', '2023-12-31'),
    (3, 'Black Friday', '2023-11-29', '2023-11-30');

-- Insert data into store_dimension
INSERT INTO store_dimension (store_id, store_name, city, state, country)
VALUES
    (1, 'Store A', 'New York', 'NY', 'USA'),
    (2, 'Store B', 'Los Angeles', 'CA', 'USA'),
    (3, 'Store C', 'London', NULL, 'UK');

-- Insert data into employee_dimension
INSERT INTO employee_dimension (employee_id, first_name, last_name,
job_title)
VALUES
    (1, 'Emily', 'White', 'Manager'),
    (2, 'George', 'Brown', 'Salesperson'),
    (3, 'Ivy', 'Green', 'Cashier');

-- Insert data into shipper_dimension
INSERT INTO shipper_dimension (shipper_id, company_name, contact_name,
contact_phone)
VALUES
    (1, 'ShipCo', 'John Doe', '555-1234'),
    (2, 'Speedy Ship', 'Jane Smith', '555-5678'),
    (3, 'Swift Transport', 'Bob Johnson', '555-9876');

-- Insert data into supplier_dimension
INSERT INTO supplier_dimension (supplier_id, supplier_name, city, state,
country)
```

```
VALUES
    (1, 'Supplier X', 'New York', 'NY', 'USA'),
    (2, 'Supplier Y', 'Los Angeles', 'CA', 'USA'),
    (3, 'Supplier Z', 'London', NULL, 'UK');

-- Insert data into sales_fact
INSERT INTO sales_fact (date_id, product_id, location_id, customer_id,
salesperson_id, payment_method_id, promotion_id, store_id, employee_id,
shipper_id, supplier_id, quantity, total_amount)
VALUES
    ('2023-10-31', 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 100.00),
    ('2023-10-31', 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 80.00),
    ('2023-11-01', 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 120.00),
    ('2023-11-02', 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 150.00);
```

QUERY:

Slice:

```
SELECT *
FROM sales_fact
WHERE date_id = '2023-10-31';
```

Dice:

```
SELECT *
FROM sales_fact
WHERE date_id = '2023-10-31'
    AND product_id = 1
    AND location_id = 1;
```

Roll Up:

```
SELECT time_dimension.year, SUM(sales_fact.total_amount) AS total_revenue
FROM sales_fact
JOIN time_dimension ON sales_fact.date_id = time_dimension.date_id
GROUP BY time_dimension.year;
```

Drill DOWN:

```
SELECT time_dimension.year, time_dimension.month,
SUM(sales_fact.total_amount) AS total_revenue
FROM sales_fact
JOIN time_dimension ON sales_fact.date_id = time_dimension.date_id
GROUP BY time_dimension.year, time_dimension.month;
```

PIVOT:

```
SELECT product_dimension.product_name, SUM(sales_fact.total_amount) AS
total_revenue
FROM sales_fact
JOIN product_dimension ON sales_fact.product_id =
product_dimension.product_id
GROUP BY product_dimension.product_name;
```

EXP2:

```
-- Common Dimension Tables
CREATE TABLE common_dimension_1 (
    common_dim1_id INT PRIMARY KEY,
    common_dim1_name VARCHAR(50)
);

CREATE TABLE common_dimension_2 (
    common_dim2_id INT PRIMARY KEY,
    common_dim2_name VARCHAR(50)
);

-- Unique Dimension Tables for Fact Table 1
CREATE TABLE dim1_fact1 (
    dim1_fact1_id INT PRIMARY KEY,
    dim1_fact1_name VARCHAR(50)
);

CREATE TABLE dim2_fact1 (
    dim2_fact1_id INT PRIMARY KEY,
    dim2_fact1_name VARCHAR(50)
);

CREATE TABLE dim3_fact1 (
    dim3_fact1_id INT PRIMARY KEY,
    dim3_fact1_name VARCHAR(50)
);

CREATE TABLE dim4_fact1 (
    dim4_fact1_id INT PRIMARY KEY,
    dim4_fact1_name VARCHAR(50)
);

-- Unique Dimension Tables for Fact Table 2
CREATE TABLE dim1_fact2 (
    dim1_fact2_id INT PRIMARY KEY,
    dim1_fact2_name VARCHAR(50)
);

CREATE TABLE dim2_fact2 (
    dim2_fact2_id INT PRIMARY KEY,
    dim2_fact2_name VARCHAR(50)
);

CREATE TABLE dim3_fact2 (
    dim3_fact2_id INT PRIMARY KEY,
    dim3_fact2_name VARCHAR(50)
);

CREATE TABLE dim4_fact2 (
    dim4_fact2_id INT PRIMARY KEY,
    dim4_fact2_name VARCHAR(50)
);

-- Fact Tables
CREATE TABLE fact_table_1 (
    common_dim1_id INT,
```

```
    common_dim2_id INT,
    dim1_fact1_id INT,
    dim2_fact1_id INT,
    dim3_fact1_id INT,
    dim4_fact1_id INT,
    fact1_measure INT,
    PRIMARY KEY (common_dim1_id, common_dim2_id),
    FOREIGN KEY (common_dim1_id) REFERENCES
common_dimension_1(common_dim1_id),
    FOREIGN KEY (common_dim2_id) REFERENCES
common_dimension_2(common_dim2_id),
    FOREIGN KEY (dim1_fact1_id) REFERENCES dim1_fact1(dim1_fact1_id),
    FOREIGN KEY (dim2_fact1_id) REFERENCES dim2_fact1(dim2_fact1_id),
    FOREIGN KEY (dim3_fact1_id) REFERENCES dim3_fact1(dim3_fact1_id),
    FOREIGN KEY (dim4_fact1_id) REFERENCES dim4_fact1(dim4_fact1_id)
);

CREATE TABLE fact_table_2 (
    common_dim1_id INT,
    common_dim2_id INT,
    dim1_fact2_id INT,
    dim2_fact2_id INT,
    dim3_fact2_id INT,
    dim4_fact2_id INT,
    fact2_measure INT,
    PRIMARY KEY (common_dim1_id, common_dim2_id),
    FOREIGN KEY (common_dim1_id) REFERENCES
common_dimension_1(common_dim1_id),
    FOREIGN KEY (common_dim2_id) REFERENCES
common_dimension_2(common_dim2_id),
    FOREIGN KEY (dim1_fact2_id) REFERENCES dim1_fact2(dim1_fact2_id),
    FOREIGN KEY (dim2_fact2_id) REFERENCES dim2_fact2(dim2_fact2_id),
    FOREIGN KEY (dim3_fact2_id) REFERENCES dim3_fact2(dim3_fact2_id),
    FOREIGN KEY (dim4_fact2_id) REFERENCES dim4_fact2(dim4_fact2_id)
);

-- Insert data into dimension and fact tables
```