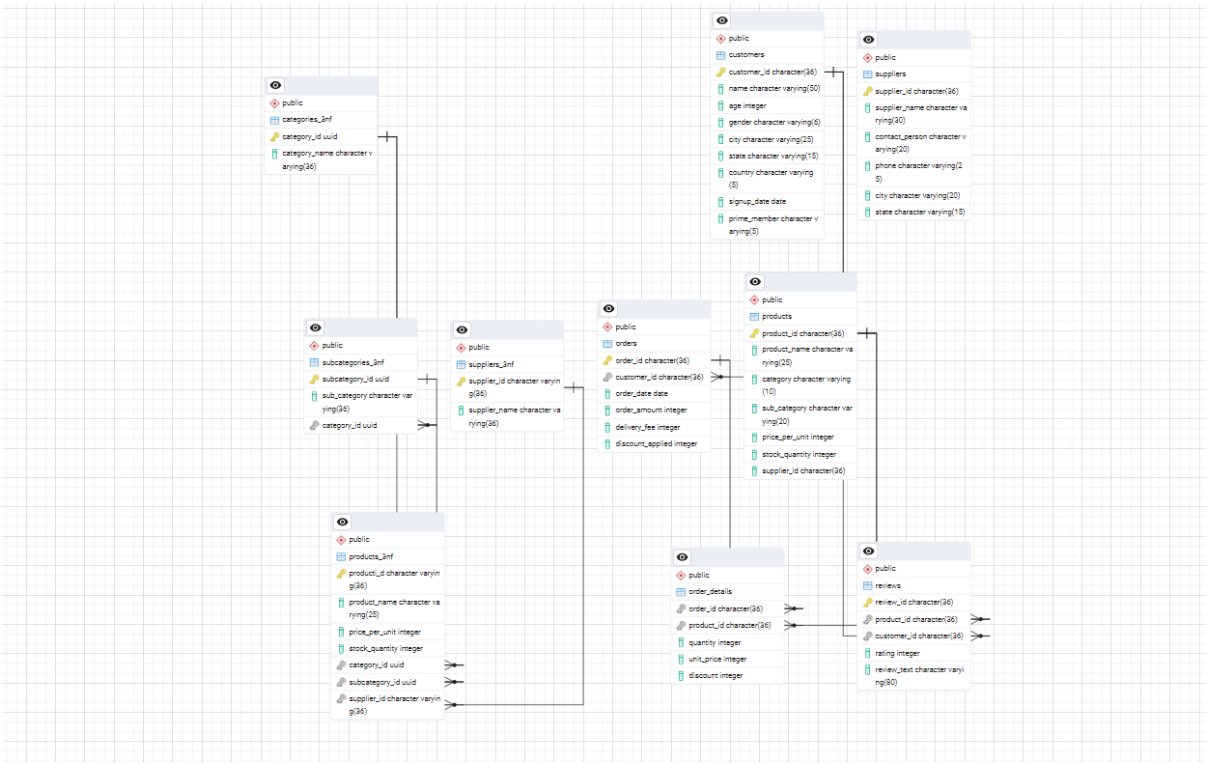
Project Title: Amazon Fresh Analytics

Data Modeling and Basic Queries

Task 1: Create an ER diagram for the Amazon Fresh database to understand the relationships between tables (e.g., Customers, Products, Orders).



Task 2: Identify the primary keys and foreign keys for each table and describe their relationships.

1. customers Table
   - Primary Key (PK): customer_id
   - Foreign Keys (FK): None
   - Relationships:
     - customer_id is referenced in the orders table.
     - customer_id is referenced in the reviews table.

2. orders Table
   - Primary Key (PK): order_id
   - Foreign Keys (FK):
     - customer_id → References customers(customer_id)
   - Relationships:
     - order_id is referenced in the order_details table.

3. order_details Table
   - Primary Key (PK): (order_id, product_id) [Composite Primary Key]

- Foreign Keys (FK):
    - order_id → References orders(order_id)
    - product_id → References products(product_id)

4. products Table
- Primary Key (PK): product_id
- Foreign Keys (FK):
    - sub_category → References subcategories_2nf(sub_category_id)
    - supplier_id → References suppliers(supplier_id)
- Relationships:
    - product_id is referenced in order_details.
    - product_id is referenced in reviews.

5. suppliers Table
- Primary Key (PK): supplier_id
- Foreign Keys (FK): None
- Relationships:
    - supplier_id is referenced in products.

6. reviews Table
- Primary Key (PK): review_id
- Foreign Keys (FK):
    - customer_id → References customers(customer_id)
    - product_id → References products(product_id)

7. categories_2nf Table
- Primary Key (PK): category_id
- Foreign Keys (FK): None
- Relationships:
    - category_id is referenced in subcategories_2nf.

8. subcategories_2nf Table
- Primary Key (PK): sub_category_id
- Foreign Keys (FK):
    - category_id → References categories_2nf(category_id)
- Relationships:
    - sub_category_id is referenced in products.

9. suppliers_2nf Table
- Primary Key (PK): supplier_id
- Foreign Keys (FK):
    - None (This might be a redundancy in the design if suppliers already exists.)

Task 3: Write a query to:

1. **Retrieve all customers from a specific city.**

```
Select *
from customers
where city = 'Allenbury';
```

2. **Fetch all products under the "Fruits" category.**

```
Select *
from Products
where category = 'Fruits';
```

Data Definition Language (DDL) and Constraints

Task 4: Write DDL statements to recreate the Customers table with the following constraints:

1. **Customer_ID as the primary key.**

```
ALTER TABLE Customers
ADD CONSTRAINT customers_pk PRIMARY KEY (Customer_ID);
```

2. **Ensure Age cannot be null and must be greater than 18.**

```
ALTER TABLE Customers
ADD CONSTRAINT Age_Check CHECK (Age >= 18);
```

3. **Add a unique constraint for Name.**

```
ALTER TABLE Customers
ADD CONSTRAINT Name UNIQUE (Name);
```

Data Manipulation Language (DML)

Task 5: Insert 3 new rows into the Products table using INSERT statements.

```
-- Insert row 1
INSERT INTO PRODUCTS (Product_ID, Product_Name, Category, Sub_Category,
Price_Per_Unit, Stock_Quantity, Supplier_ID)
VALUES ('2aa28375-c563-41b5-aa33', 'However Fruit', 'Fruits', 'Sub-Fruits-1', 207, 290,
'0658c953-98c4-4d00-bf29-4fbfe4aca4cd');

-- Insert row 2
INSERT INTO PRODUCTS (Product_ID, Product_Name, Category, Sub_Category,
Price_Per_Unit, Stock_Quantity, Supplier_ID)
```

VALUES ('e9282403-e234-4e35-a711', 'Serve Snack', 'Snacks', 'Sub-Snacks-1', 905, 259, 'cb890936-8142-4fa3-ac60-2ecba78f8aa8');

-- Insert row 3
INSERT INTO PRODUCTS (Product_ID, Product_Name, Category, Sub_Category, Price_Per_Unit, Stock_Quantity, Supplier_ID)
VALUES ('d79d1b95-ecdf-4810-aea0', 'Rule Fruit', 'Fruits', 'Sub-Fruits-4', 111, 26, '455b7097-b656-49b8-9cf2-a98d71d3ba88');

Task 6: Update the stock quantity of a product where Product_ID matches a specific ID.

UPDATE PRODUCTS
SET Stock_Quantity = 400
where Product_ID = '2aa28375-c563-41b5-aa33';

Task 7: Delete a supplier from the Suppliers table where their city matches a specific value.

DELETE FROM suppliers
WHERE city = 'West Linda';

SQL Constraints and Operators

Task 8: Use SQL constraints to:

1. **Add a CHECK constraint to ensure that ratings in the Reviews table are between 1 and 5.**

ALTER TABLE REVIEWS
ADD CONSTRAINT Check_Rating_Range
CHECK (Rating >= 1 AND Rating <= 5);

2. **Add a DEFAULT constraint for the Prime Member column in the Customers table (default value: "No").**

ALTER TABLE CUSTOMERS
ALTER COLUMN PRIME_MEMBER SET DEFAULT 'No';

Clauses and Aggregations

Task 9: Write queries using:

1. **WHERE clause to find orders placed after 2024-01-01.**

SELECT * FROM Orders
WHERE Order_Date > '2024-01-01';

2. **HAVING clause to list products with average ratings greater than 4.**

```
Select Product_ID, AVG(Rating) AS AVERAGE_RATING
FROM REVIEWS
GROUP BY PRODUCT_ID
HAVING AVG(RATING)>4;
```

3. **GROUP BY and ORDER BY clauses to rank products by total sales.**

```
SELECT
   p.Product_ID,
   p.Product_Name,
   SUM(od.Quantity * p.Price_Per_Unit) AS Total_Sales
FROM order_details od
JOIN products p ON od.Product_ID = p.Product_ID
GROUP BY p.Product_ID, p.Product_Name
ORDER BY Total_Sales DESC;
```

ACID Transactions and TCL

Task 10: Write a transaction to:

1. **Deduct stock from the Products table when a product is sold.**
2. **Insert a new row in the OrderDetails table for the sale.**
3. **Rollback the transaction if the stock is insufficient.**
4. **Commit changes otherwise.**

```
DO $$
DECLARE
   v_stock_quantity INT;
   v_price_per_unit NUMERIC;
   v_order_id UUID;
   v_customer_id UUID := '96ed9663-7e5c-4c11-bbf9-c8ccb4c111d7'; -- Replace with actual
customer ID
BEGIN
   -- Start Transaction
   BEGIN
      -- Check if sufficient stock is available
      SELECT Stock_Quantity INTO v_stock_quantity
      FROM Products
      WHERE Product_ID = '2aa28375-c563-41b5-aa33-8e2c2e0f4db9'
      FOR UPDATE;  -- Lock row to prevent race conditions

      -- If stock is insufficient, raise exception
      IF v_stock_quantity IS NULL OR v_stock_quantity < 5 THEN
         RAISE EXCEPTION 'Transaction rolled back: Insufficient stock';
      END IF;

      -- Retrieve price per unit
```

```
    SELECT Price_Per_Unit INTO v_price_per_unit
    FROM Products
    WHERE Product_ID = '2aa28375-c563-41b5-aa33-8e2c2e0f4db9';

    -- Ensure price is not null
    IF v_price_per_unit IS NULL THEN
        RAISE EXCEPTION 'Price per unit not found for product %', '2aa28375-c563-41b5-
aa33-8e2c2e0f4db9';
    END IF;

    -- Generate a unique order ID
    v_order_id := gen_random_uuid();

    -- Insert into Orders table
    INSERT INTO Orders (Order_ID, Customer_ID, Order_Date, Order_Amount,
Delivery_Fee, Discount_Applied)
    VALUES (v_order_id, v_customer_id, CURRENT_DATE, 5 * v_price_per_unit, 321,
81);

    -- Update stock quantity
    UPDATE Products
    SET Stock_Quantity = Stock_Quantity - 5
    WHERE Product_ID = '2aa28375-c563-41b5-aa33-8e2c2e0f4db9';

    -- Insert order details
    INSERT INTO Order_Details (Order_ID, Product_ID, Quantity, Unit_Price, Discount)
    VALUES (
        v_order_id,
        '2aa28375-c563-41b5-aa33-8e2c2e0f4db9',
        5,
        v_price_per_unit,
        81
    );

    -- If everything is successful, print success message
    RAISE NOTICE 'Transaction committed: Stock updated and order recorded with
Order_ID: %', v_order_id;

  EXCEPTION
    WHEN OTHERS THEN
        -- Rollback automatically handled, just print error message
        RAISE NOTICE 'Transaction rolled back due to error: %', SQLERRM;
        RETURN;
  END;
END $$;
```

Task 10: Identifying High-Value Customers Scenario:
Amazon Fresh wants to identify top customers based on their total spending. We will:

1. Calculate each customer's total spending.
2. Rank customers based on their spending.
3. Identify customers who have spent more than ₹5,000.

```
SELECT
   c.Customer_ID,
   c.Name,
   SUM(o.Order_Amount) AS Total_Spending,
   RANK() OVER (ORDER BY SUM(o.Order_Amount) DESC) AS Rank
FROM Customers c
JOIN Orders o ON c.Customer_ID = o.Customer_ID
GROUP BY c.Customer_ID, c.Name
HAVING SUM(o.Order_Amount) > 5000
```

Complex Aggregations and Joins

Task 11: Use SQL to:

1. **Join the Orders and OrderDetails tables to calculate total revenue per order.**

```
SELECT
   o.Order_ID,
   o.Customer_ID,
   o.Order_Date,
   o.Order_Amount,
   o.Delivery_Fee,
   o.Discount_Applied,
   SUM (od.Quantity * od.Unit_Price - od.Discount) AS Total_Revenue
FROM Orders o
JOIN Order_Details od ON o.Order_ID = od.Order_ID
GROUP BY o.Order_ID, o.Customer_ID, o.Order_Date, o.Order_Amount, o.Delivery_Fee,
o.Discount_Applied;
```

2. **Identify customers who placed the most orders in a specific time period.**

```
SELECT
   c.Customer_ID,
   c.Name,
   COUNT(o.Order_ID) AS Total_Orders
FROM Orders o
JOIN Customers c ON o.Customer_ID = c.Customer_ID  -- Joining Orders with Customers
table
```

```
WHERE o.Order_Date BETWEEN '2025-01-01' AND '2025-12-31'
GROUP BY c.Customer_ID, c.Name
ORDER BY Total_Orders DESC
LIMIT 10;  -- Get the top 10 customers
```

3. **Find the supplier with the most products in stock.**

```
SELECT
   s.Supplier_ID,
   s.Supplier_Name,
   SUM(p.Quantity_In_Stock) AS Total_Products_In_Stock
FROM Products p
JOIN Suppliers s ON p.Supplier_ID = s.Supplier_ID  -- Joining Products with Suppliers table
GROUP BY s.Supplier_ID, s.Supplier_Name
ORDER BY Total_Products_In_Stock DESC
LIMIT 1; -- Get the supplier with the most products in stock
```

Normalization

Task 12: Normalize the Products table to 3NF:

1. **Separate product categories and subcategories into a new table.**
2. **Create foreign keys to maintain relationships.**

```
CREATE TABLE Categories_3NF (
   Category_ID UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   Category_Name VARCHAR(36) NOT NULL
);

CREATE TABLE Subcategories_3NF (
   SubCategory_ID UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   Sub_Category VARCHAR(36) NOT NULL,
   Category_ID UUID,
   FOREIGN KEY (Category_ID) REFERENCES Categories_3NF(Category_ID)
);

CREATE TABLE Suppliers_3NF (
   Supplier_ID VARCHAR(36) PRIMARY KEY,
   Supplier_Name VARCHAR(36)
);

CREATE TABLE Products_3NF (
   ProductI_D VARCHAR(36) PRIMARY KEY,
   Product_Name VARCHAR(25) NOT NULL,
   Price_Per_Unit INT NOT NULL,
```

```
    Stock_Quantity INT NOT NULL,
    Category_ID UUID,
    SubCategory_ID UUID,
    Supplier_ID VARCHAR(36),
    FOREIGN KEY (Category_ID) REFERENCES Categories_3NF(Category_ID),
    FOREIGN KEY (SubCategory_ID) REFERENCES
Subcategories_3NF(SubCategory_ID),
    FOREIGN KEY (Supplier_ID) REFERENCES Suppliers_3NF(Supplier_ID)
);
```

Subqueries and Nested Queries

Task 13: Write a subquery to:

1. **Identify the top 3 products based on sales revenue.**

```
SELECT product_id, total_revenue
FROM (
    SELECT product_id,
        SUM(quantity * unit_price * (1 - (discount / 100.0))) AS total_revenue,
        RANK() OVER (ORDER BY SUM(quantity * unit_price * (1 - (discount / 100.0)))
DESC) AS revenue_rank
    FROM order_details
    WHERE quantity > 0 AND unit_price > 0  -- Ensure valid values
    GROUP BY product_id
) ranked_products
WHERE revenue_rank <= 3;
```

2. **Find customers who haven't placed any orders yet.**

```
SELECT customer_id, name
FROM customers c
WHERE NOT EXISTS (
    SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id
);
```

Real-World Analysis

Task 14: Provide actionable insights:

1. **Which cities have the highest concentration of Prime members?**

```
SELECT city,
      COUNT(CASE WHEN prime_member = 'Yes' THEN 1 END) AS
prime_member_count,
      COUNT(*) AS total_customers,
```

```
    ROUND(100.0 * COUNT(CASE WHEN prime_member = 'Yes' THEN 1 END) /
COUNT(*), 2) AS prime_member_percentage
FROM customers
GROUP BY city
HAVING COUNT(*) > 0
ORDER BY prime_member_percentage DESC;
```

OR

To **filter cities that have more than one Prime member**, you can use the HAVING clause after grouping the data.

```
SELECT city,
    COUNT(CASE WHEN prime_member = 'Yes' THEN 1 END) AS Prime_Members,
    COUNT(*) AS total_customers,
    ROUND(100.0 * COUNT(CASE WHEN prime_member = 'Yes' THEN 1 END) /
COUNT(*), 2) AS Prime_Concentration
FROM customers
GROUP BY city
HAVING COUNT(CASE WHEN prime_member = 'Yes' THEN 1 END) > 1
ORDER BY Prime_Concentration DESC;
```

2. **What are the top 3 most frequently ordered categories?**

```
SELECT p.Category, COUNT(*) AS total_orders
FROM Order_Details od
JOIN Products p ON od.Product_ID = p.Product_ID
GROUP BY p.Category
ORDER BY total_orders DESC
LIMIT 3;
```