**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).**

A screenshot of a computer screen

AI-generated content may be incorrect.

**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';

1. **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);

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

ALTER TABLE Customers

ADD CONSTRAINT Age\_Check CHECK (Age >= 18);

1. **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);

1. **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';

1. **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;

1. **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;

1. **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

1. **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;

1. **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;

1. **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;