FINAL

Course name: DBMS
Topic: Farmer's Market Managing System
Student: Ayazbek Abdanur
Group: BDA-2409
Instructor: Shaiken Olzhas

I. Database Design & Conceptualization

1.1. Overview of the Business Case

The Farmer's Market Managing System is a digital solution designed to streamline the management and daily operations of farmers' markets. It serves as a centralized platform for organizing vendor activities, tracking product inventory, managing sales transactions, and facilitating communication between vendors and customers. The system ensures accurate monitoring of stock levels, transparent financial reporting, and efficient vendor stall allocation, ultimately improving the overall experience for all stakeholders involved in the market. By digitizing these processes, the system minimizes manual errors, enhances operational efficiency, and fosters better coordination among market participants.

The primary users of the *Farmer's Market Managing System* include **vendors**, **administrators**, and **customers**. Vendors, typically farmers or small business owners, use the system to manage their product inventory, update stock levels, and keep track of daily sales. Market administrators oversee the overall functioning of the market, including vendor registrations, stall assignments, and financial reporting. Customers interact with the system indirectly by viewing product availability, submitting feedback, and participating in promotions or loyalty programs. Each user group has distinct roles and permissions within the system to ensure smooth collaboration and transparency.

The main goal of the Farmer's Market Managing System is to address common operational challenges in traditional farmers' markets. These include inefficient inventory management, poor tracking of sales records, lack of real-time data on vendor performance, and limited customer engagement. By providing tools for accurate sales reporting, automated stock tracking, and streamlined communication, the system minimizes operational bottlenecks and reduces the dependency on manual paperwork. Additionally, it fosters customer satisfaction through feedback collection and ensures vendors can make data-driven decisions to improve their sales performance.

1.2. Key Features of the System

Vendor Management: Register and manage vendor profiles, including product categories and stall locations.

Inventory Tracking: Monitor product stock levels and track restocking needs.

Sales Recording: Keep detailed records of sales transactions for vendors and administrators.

Customer Feedback System: Collect and manage customer reviews and ratings for vendors.

Financial Reporting: Generate reports on sales, profits, and vendor performance.

1.3. Data Requirements
Vendors: Vendor ID, Name, Contact Details, Product Category, Stall Location.
Products: Product ID, Name, Category, Price, Quantity Available.
Sales Records: Sale ID, Vendor ID, Product ID, Quantity Sold, Sale Date, Total Amount.
Customers: Customer ID, Name, Contact Details, Purchase History.
Feedback: Feedback ID, Customer ID, Vendor ID, Rating, Comment, Feedback Date.

1.4. Entities, Attributes, and Constraints

1. **Vendors**
   - **Attributes:**

   VendorID (SERIAL, Primary Key, NOT NULL)
   VendorName (VARCHAR, NOT NULL)
   ContactDetails (VARCHAR)
   ProductCategory (VARCHAR)
   StallLocation (VARCHAR)
   - **Constraints:**

   VendorID must be unique and serve as the primary identifier for each vendor.
   VendorName cannot be null.

2. **Products**
   - **Attributes:**

   ProductID (SERIAL, Primary Key, NOT NULL)
   ProductName (VARCHAR, NOT NULL)
   Category (VARCHAR)
   Price (DECIMAL, NOT NULL)
   QuantityAvailable (INT)
   VendorID (INT, Foreign Key references Vendors(VendorID))
   - **Constraints:**

   ProductID must be unique and serve as the primary identifier for each product.
   Price cannot be null.
   VendorID ensures each product is associated with a valid vendor.

3. **Customers**
   - **Attributes:**

   CustomerID (SERIAL, Primary Key, NOT NULL)
   CustomerName (VARCHAR, NOT NULL)
   ContactDetails (VARCHAR)
   - **Constraints:**

   CustomerID must be unique and serve as the primary identifier for each customer.
   CustomerName cannot be null.

4. **Sales**
    - **Attributes:**

SaleID (SERIAL, Primary Key, NOT NULL)
VendorID (INT, Foreign Key references Vendors(VendorID))
ProductID (INT, Foreign Key references Products(ProductID))
CustomerID (INT, Foreign Key references Customers(CustomerID))
QuantitySold (INT)
SaleDate (DATE)
TotalAmount (DECIMAL, NOT NULL)

    - **Constraints:**

SaleID must be unique and serve as the primary identifier for each sale.
VendorID, ProductID, and CustomerID must reference valid entries in their respective tables.
TotalAmount cannot be null.

5. **Feedback**
    - **Attributes:**

FeedbackID (SERIAL, Primary Key, NOT NULL)
CustomerID (INT, Foreign Key references Customers(CustomerID))
VendorID (INT, Foreign Key references Vendors(VendorID))
Rating (INT, CHECK BETWEEN 1 AND 5)
Comment (TEXT)
FeedbackDate (DATE)

    - **Constraints:**

FeedbackID must be unique and serve as the primary identifier for each feedback.
Rating must be an integer between 1 and 5.
CustomerID and VendorID must reference valid entries in their respective tables.

1.5. Third Normal Form (3NF)

- **Vendors:**

Super Key: {VendorID, VendorName, ContactDetails, ProductCategory, StallLocation}
Candidate Key: {VendorID}
Non-Prime Attributes: {VendorName, ContactDetails, ProductCategory, StallLocation}
The table is in 3NF because all non-prime attributes depend only on the primary key.

- **Products:**

Super Key: {ProductID, ProductName, Category, Price, QuantityAvailable, VendorID}
Candidate Key: {ProductID}
Non-Prime Attributes: {ProductName, Category, Price, QuantityAvailable, VendorID}
The table is in 3NF because non-prime attributes depend only on the primary key.

- **Customers:**

Super Key: {CustomerID, CustomerName, ContactDetails}

Candidate Key: {CustomerID}
Non-Prime Attributes: {CustomerName, ContactDetails}
The table is in 3NF because non-prime attributes depend only on the primary key.

- **Sales:**

Super Key: {SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount}
Candidate Key: {SaleID}
Non-Prime Attributes: {VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount}
The table is in 3NF because non-prime attributes depend only on the primary key.

- **Feedback:**

Super Key: {FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate}
Candidate Key: {FeedbackID}
Non-Prime Attributes: {CustomerID, VendorID, Rating, Comment, FeedbackDate}
The table is in 3NF because non-prime attributes depend only on the primary key.

1.6. Relationships on the Tables

1. **Vendors ↔ Products:** One vendor can have many products, but each product belongs to one vendor. *(1:N Relationship)*
2. **Vendors ↔ Sales:** One vendor can have many sales, but each sale is linked to one vendor. *(1:N Relationship)*
3. **Customers ↔ Sales:** One customer can make many purchases, but each sale is linked to one customer. *(1:N Relationship)*
4. **Products ↔ Sales:** One product can appear in many sales records, but each sale refers to one product. *(1:N Relationship)*
5. **Customers ↔ Feedback:** One customer can provide feedback multiple times, but each feedback entry belongs to one customer. *(1:N Relationship)*
6. **Vendors ↔ Feedback:** One vendor can have many feedback entries, but each feedback entry refers to one vendor. *(1:N Relationship)*

1.7. Cardinality for Each Relationship

1. **Vendor ↔ Products:** (1:N)
2. **Vendor ↔ Sales:** (1:N)
3. **Customer ↔ Sales:** (1:N)
4. **Product ↔ Sales:** (1:N)
5. **Customer ↔ Feedback:** (1:N)

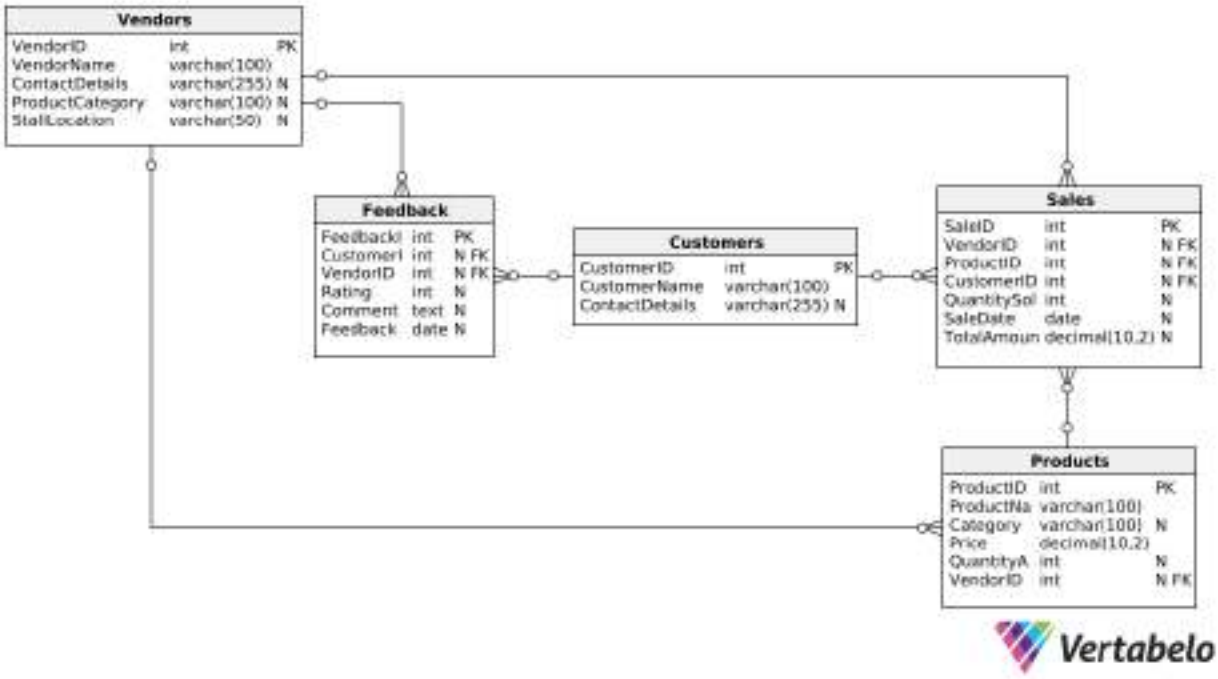
6. **Vendor ↔ Feedback:** (1:N)

1.8. Primary and Foreign Keys

- **Primary Keys:**
  - Vendors: VendorID
  - Products: ProductID
  - Customers: CustomerID
  - Sales: SaleID
  - Feedback: FeedbackID

- **Foreign Keys:**
  - Products: VendorID → Vendors(VendorID)
  - Sales: VendorID → Vendors(VendorID)
  - Sales: ProductID → Products(ProductID)
  - Sales: CustomerID → Customers(CustomerID)
  - Feedback: CustomerID → Customers(CustomerID)
  - Feedback: VendorID → Vendors(VendorID)

1.9. ER-Diagram



[1]

II. Database Implementation & Data Population

2.1. DDL Implementation (SQL Schema Design)

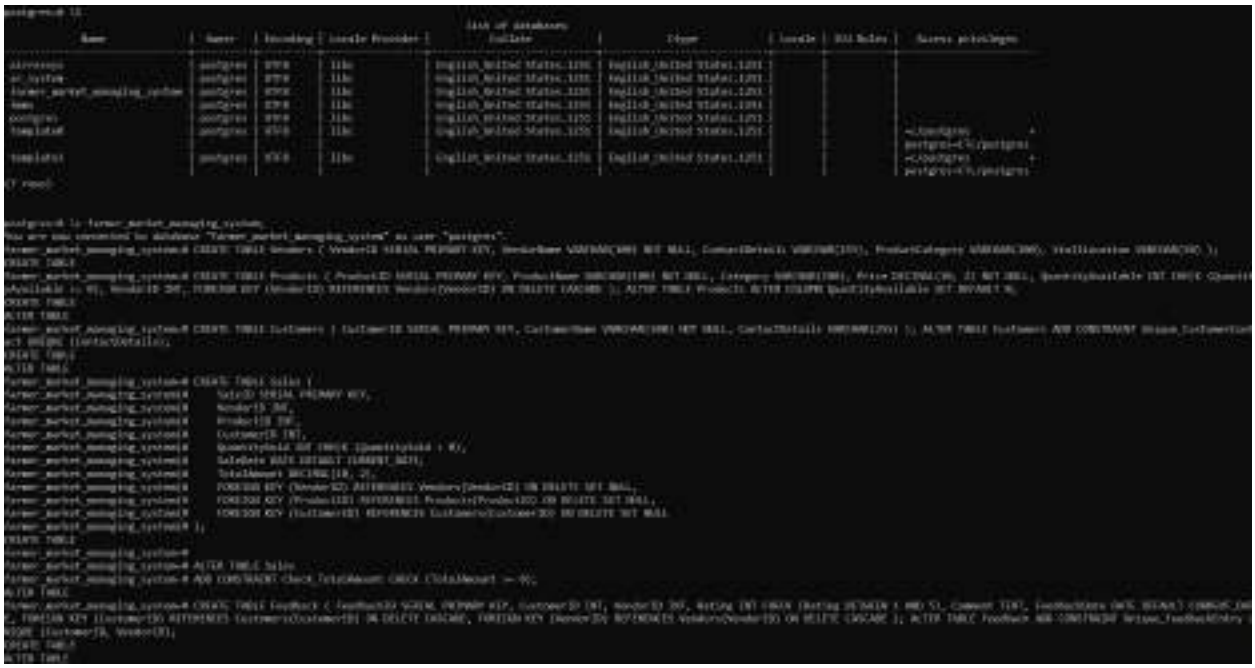
```
CREATE TABLE Vendors (
    VendorID SERIAL PRIMARY KEY,
    VendorName VARCHAR(100) NOT NULL,
    ContactDetails VARCHAR(255),
    ProductCategory VARCHAR(100),
```

```
    StallLocation VARCHAR(50)
);

CREATE TABLE Products (
    ProductID SERIAL PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL,
    Category VARCHAR(100),
    Price DECIMAL(10, 2) NOT NULL,
    QuantityAvailable INT CHECK (QuantityAvailable >= 0),
    VendorID INT,
        FOREIGN KEY (VendorID) REFERENCES Vendors(VendorID) ON DELETE
CASCADE
);

ALTER TABLE Products
ALTER COLUMN QuantityAvailable SET DEFAULT 0;

CREATE TABLE Customers (
    CustomerID SERIAL PRIMARY KEY,
    CustomerName VARCHAR(100) NOT NULL,
    ContactDetails VARCHAR(255)
);

ALTER TABLE Customers
ADD CONSTRAINT Unique_CustomerContact UNIQUE (ContactDetails);

CREATE TABLE Sales (
    SaleID SERIAL PRIMARY KEY,
    VendorID INT,
    ProductID INT,
    CustomerID INT,
    QuantitySold INT CHECK (QuantitySold > 0),
    SaleDate DATE DEFAULT CURRENT_DATE,
    TotalAmount DECIMAL(10, 2),
      FOREIGN KEY (VendorID) REFERENCES Vendors(VendorID) ON DELETE SET
NULL,
      FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE SET
NULL,
      FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE
SET NULL
);

ALTER TABLE Sales
```

ADD CONSTRAINT Check_TotalAmount CHECK (TotalAmount >= 0);

CREATE TABLE Feedback (
FeedbackID SERIAL PRIMARY KEY,
CustomerID INT,
VendorID INT,
Rating INT CHECK (Rating BETWEEN 1 AND 5),
Comment TEXT,
FeedbackDate DATE DEFAULT CURRENT_DATE,
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,
FOREIGN KEY (VendorID) REFERENCES Vendors(VendorID) ON DELETE CASCADE
);

ALTER TABLE Feedback
ADD CONSTRAINT Unique_FeedbackEntry UNIQUE (CustomerID, VendorID);



2.2. Data Insertion (At Least 30 Rows Per Table)

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (1, 'Mason', 'mbirney0@reddit.com', 'Crafts', '12273 Luster Center');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (2, 'Jamison', 'jmccolgan1@slideshare.net', 'Bakery', '991 Riverside Place');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (3, 'Scot', 'slitt2@printfriendly.com', 'Vegetables', '4603 Golf Course Lane');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (4, 'Myrtle', 'monion3@tmall.com', 'Vegetables', '481 Graedel Terrace');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (5, 'Barnie', 'bching4@foxnews.com', 'Fruits', '2521 Shasta Parkway');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (6, 'Fiorenze', 'fgritten5@nifty.com', 'Meat', '4 Riverside Center');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (7, 'Kliment', 'kfearn6@washingtonpost.com', 'Honey', '6 Briar Crest Road');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (8, 'Vin', 'vjobbins7@disqus.com', 'Eggs', '9 High Crossing Parkway');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (9, 'Ynez', 'ypenwright8@fema.gov', 'Bakery', '58266 South Circle');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (10, 'Gearalt', 'gwadsworth9@g.co', 'Herbs', '5 Pepper Wood Place');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (11, 'Ashia', 'aolleya@vinaora.com', 'Dairy', '91844 Continental Court');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (12, 'Freddy', 'fnadinb@engadget.com', 'Dairy', '4 Gale Plaza');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (13, 'Far', 'fcolesonc@flickr.com', 'Meat', '503 Chinook Alley');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (14, 'Sam', 'smcmanamend@technorati.com', 'Bakery', '0223 Briar Crest Park');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (15, 'Tiphani', 'trapseye@shop-pro.jp', 'Flowers', '32 Muir Park');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (16, 'Philippa', 'pwinspearef@hexun.com', 'Crafts', '5 Dennis Plaza');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (17, 'Barbette', 'bbarnishg@youku.com', 'Meat', '7 Carey Way');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (18, 'Lee', 'lhuddlestoneh@meetup.com', 'Herbs', '0110 Red Cloud Crossing');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (19, 'Evita', 'eohagertyi@friendfeed.com', 'Honey', '940 Kenwood Plaza');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (20, 'Bennett', 'bmcvityj@mail.ru', 'Vegetables', '3472 Thierer Plaza');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (21, 'Anestassia', 'aterrazzok@jiathis.com', 'Dairy', '335 Fulton Drive');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (22, 'Aurelea', 'atodarellol@addthis.com', 'Meat', '29 Brown Court');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (23, 'Germayne', 'gprattym@sun.com', 'Vegetables', '13 Columbus Parkway');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (24, 'Barbe', 'bwillingalen@telegraph.co.uk', 'Honey', '3 Laurel Alley');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (25, 'Margalo', 'mrozziero@de.vu', 'Meat', '48 Golden Leaf Alley');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (26, 'Bancroft', 'bgerhtsp@blogs.com', 'Bakery', '1970 Golf Way');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (27, 'Codie', 'cthorq@mozilla.org', 'Crafts', '30 Ridge Oak Plaza');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (28, 'Godfree', 'gseccomber@fastcompany.com', 'Eggs', '38443 Memorial Hill');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (29, 'Lind', 'llemins@slashdot.org', 'Bakery', '232 Crowley Point');

insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (30, 'Matias', 'mdunfordt@ask.com', 'Dairy', '893 1st Lane');

insert into customers (CustomerID, CustomerName, ContactDetails) values (1, 'Georgeanna Cuttin', '94-540-6270');

insert into customers (CustomerID, CustomerName, ContactDetails) values (2, 'Olwen de Leon', '60-171-7504');

insert into customers (CustomerID, CustomerName, ContactDetails) values (3, 'Auria Bever', '20-653-9161');

insert into customers (CustomerID, CustomerName, ContactDetails) values (4, 'Carver Midgley', '16-367-6417');

insert into customers (CustomerID, CustomerName, ContactDetails) values (5, 'Vlad Flood', '74-909-2107');

insert into customers (CustomerID, CustomerName, ContactDetails) values (6, 'Johnna Calcutt', '78-859-6203');

insert into customers (CustomerID, CustomerName, ContactDetails) values (7, 'Amye Wiffill', '66-182-8339');

insert into customers (CustomerID, CustomerName, ContactDetails) values (8, 'Raviv Bennell', '85-592-2552');

insert into customers (CustomerID, CustomerName, ContactDetails) values (9, 'Brett Ingarfill', '32-918-4349');

insert into customers (CustomerID, CustomerName, ContactDetails) values (10, 'Constanta Baroche', '27-675-2792');

insert into customers (CustomerID, CustomerName, ContactDetails) values (11, 'Maximilianus Lautie', '97-738-5933');

insert into customers (CustomerID, CustomerName, ContactDetails) values (12, 'Colet Leathley', '22-810-1010');

insert into customers (CustomerID, CustomerName, ContactDetails) values (13, 'Kirstin Harlick', '85-991-5501');
insert into customers (CustomerID, CustomerName, ContactDetails) values (14, 'Bunny Gilstoun', '41-491-2615');
insert into customers (CustomerID, CustomerName, ContactDetails) values (15, 'Louise McCay', '66-592-8153');
insert into customers (CustomerID, CustomerName, ContactDetails) values (16, 'Addy Karlsson', '44-977-7864');
insert into customers (CustomerID, CustomerName, ContactDetails) values (17, 'Ruby Cunliffe', '64-764-6488');
insert into customers (CustomerID, CustomerName, ContactDetails) values (18, 'Adoree Saunders', '93-606-2199');
insert into customers (CustomerID, CustomerName, ContactDetails) values (19, 'Austen Madison', '14-697-1089');
insert into customers (CustomerID, CustomerName, ContactDetails) values (20, 'Josiah Balderstone', '06-024-7131');
insert into customers (CustomerID, CustomerName, ContactDetails) values (21, 'Tobe Winton', '94-950-7573');
insert into customers (CustomerID, CustomerName, ContactDetails) values (22, 'Woodrow Absalom', '13-795-3307');
insert into customers (CustomerID, CustomerName, ContactDetails) values (23, 'Beryle Leger', '96-610-5516');
insert into customers (CustomerID, CustomerName, ContactDetails) values (24, 'Adelheid Fridlington', '94-404-5984');
insert into customers (CustomerID, CustomerName, ContactDetails) values (25, 'Layne Stopp', '46-624-1272');
insert into customers (CustomerID, CustomerName, ContactDetails) values (26, 'Dion Spurdens', '14-286-9720');
insert into customers (CustomerID, CustomerName, ContactDetails) values (27, 'Cleon Jeannon', '68-717-7897');
insert into customers (CustomerID, CustomerName, ContactDetails) values (28, 'Marquita Richings', '79-974-0287');
insert into customers (CustomerID, CustomerName, ContactDetails) values (29, 'Wilton MacMenamy', '66-369-7899');
insert into customers (CustomerID, CustomerName, ContactDetails) values (30, 'Latashia Quinnelly', '99-481-7244');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (1, 1, 1, 2, 'Great for managing market layout', '2021-04-25 18:53:52');
insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (2, 2, 2, 4, 'Great app', '2024-04-11 21:21:50');
insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (3, 3, 3, 2, 'Great for managing market layout', '2021-03-31 09:19:04');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (4, 4, 4, 3, 'Great for tracking customer feedback', '2022-07-26 04:15:02');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (5, 5, 5, 5, 'Great app', '2019-07-11 14:29:19');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (6, 6, 6, 2, 'Great for organizing market events', '2021-07-13 07:33:48');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (7, 7, 7, 3, 'Useful for tracking vendor performance', '2021-10-25 12:20:38');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (8, 8, 8, 1, 'Great for managing multiple markets', '2020-11-22 15:48:09');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (9, 9, 9, 5, 'Great for organizing market events', '2022-06-25 02:29:06');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (10, 10, 10, 4, 'Needs better integration with social media', '2021-07-29 02:43:30');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (11, 11, 11, 2, 'Great for tracking customer feedback', '2025-02-07 10:43:50');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (12, 12, 12, 4, 'Could use better customer support', '2022-12-24 22:24:13');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (13, 13, 13, 2, 'Needs more features for tracking sales', '2023-06-18 14:01:52');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (14, 14, 14, 4, 'Great for managing multiple markets', '2019-07-05 23:18:03');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (15, 15, 15, 5, 'Love the design and layout', '2021-08-31 07:33:54');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (16, 16, 16, 4, 'Needs better support for different languages', '2019-08-11 20:43:16');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (17, 17, 17, 5, 'Could use better customer support', '2020-01-09 20:50:51');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (18, 18, 18, 1, 'Helpful for managing vendor applications', '2018-07-25 06:29:38');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (19, 19, 19, 2, 'Great for managing multiple markets', '2023-11-17 10:07:04');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (20, 20, 20, 5, 'Love the integration with payment systems', '2023-03-06 06:57:40');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (21, 21, 21, 2, 'Great for managing multiple markets', '2019-12-07 10:32:55');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (22, 22, 22, 4, 'Helpful for managing vendor applications', '2020-10-24 04:28:44');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (23, 23, 23, 5, 'Could use more marketing tools', '2019-02-05 06:26:20');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (24, 24, 24, 4, 'Love the mobile app version', '2020-11-11 16:47:22');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (25, 25, 25, 4, 'Could use more analytics features', '2019-02-02 20:47:58');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (26, 26, 26, 1, 'Love the scheduling features', '2021-01-10 17:46:49');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (27, 27, 27, 5, 'Love the integration with payment systems', '2024-04-14 05:40:21');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (28, 28, 28, 2, 'Helpful for managing inventory', '2020-05-26 10:07:22');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (29, 29, 29, 3, 'Love the design and layout', '2019-12-19 21:17:49');

insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (30, 30, 30, 1, 'Useful for tracking vendor performance', '2023-10-30 13:29:40');

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (1, 'Fresh Eggs', 'Fruits', 19.0, 1, 1);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (2, 'Handmade Soap', 'Honey', 15.37, 9, 2);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (3, 'Organic Apples', 'Dairy', 79.96, 7, 3);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (4, 'Handcrafted Jewelry', 'Sauces', 20.62, 6, 4);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (5, 'Farm-Fresh Milk', 'Baked Goods', 46.3, 13, 5);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (6, 'Handwoven Baskets', 'Herbs', 43.22, 6, 6);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (7, 'Homemade Jam', 'Honey', 75.43, 12, 7);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (8, 'Artisan Bread', 'Soaps', 90.18, 13, 8);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (9, 'Local Honey', 'Meat', 23.13, 2, 9);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (10, 'Organic Apples', 'Fruits', 17.8, 5, 10);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (11, 'Handwoven Baskets', 'Pickles', 81.31, 1, 11);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (12, 'Homemade Jam', 'Dairy', 15.67, 9, 12);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (13, 'Handwoven Baskets', 'Dairy', 26.19, 9, 13);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (14, 'Natural Skincare Products', 'Soaps', 9.36, 12, 14);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (15, 'Gourmet Sausages', 'Sauces', 36.45, 3, 15);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (16, 'Herbal Tea Blends', 'Jams and Jellies', 89.25, 6, 16);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (17, 'Fresh Eggs', 'Crafts', 95.97, 3, 17);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (18, 'Local Honey', 'Vegetables', 21.16, 8, 18);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (19, 'Local Honey', 'Crafts', 51.48, 15, 19);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (20, 'Fresh Eggs', 'Honey', 42.82, 7, 20);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (21, 'Gourmet Sausages', 'Fruits', 88.64, 7, 21);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (22, 'Artisan Bread', 'Jams and Jellies', 24.76, 14, 22);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (23, 'Homemade Jam', 'Eggs', 55.31, 6, 23);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (24, 'Handcrafted Jewelry', 'Eggs', 25.15, 12, 24);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (25, 'Natural Skincare Products', 'Sauces', 97.99, 12, 25);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (26, 'Hand-poured Candles', 'Eggs', 11.44, 6, 26);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (27, 'Fresh Eggs', 'Pickles', 7.71, 6, 27);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (28, 'Handwoven Baskets', 'Crafts', 14.29, 12, 28);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (29, 'Fresh Eggs', 'Dairy', 40.23, 15, 29);

insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (30, 'Farmhouse Cheese', 'Soaps', 81.24, 5, 30);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (1, 1, 1, 1, 15, '2023-06-22 09:52:41', 47.87);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (2, 2, 2, 2, 7, '2023-09-30 21:38:26', 18.22);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (3, 3, 3, 3, 3, '2023-03-20 09:27:17', 44.5);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (4, 4, 4, 4, 6, '2023-03-12 07:01:54', 31.09);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (5, 5, 5, 5, 8, '2022-02-21 03:46:56', 13.33);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (6, 6, 6, 6, 2, '2024-12-24 06:04:22', 6.47);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (7, 7, 7, 7, 6, '2022-01-13 08:56:07', 21.31);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (8, 8, 8, 8, 13, '2024-08-23 11:37:03', 8.37);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (9, 9, 9, 9, 5, '2024-12-01 19:10:15', 43.35);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (10, 10, 10, 10, 15, '2023-02-08 06:00:38', 33.28);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (11, 11, 11, 11, 7, '2023-01-31 06:10:17', 35.9);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (12, 12, 12, 12, 15, '2023-04-10 00:29:49', 5.29);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (13, 13, 13, 13, 3, '2024-09-13 15:44:12', 3.52);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (14, 14, 14, 14, 6, '2023-06-26 20:37:18', 21.2);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (15, 15, 15, 15, 8, '2023-07-20 11:47:47', 8.42);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (16, 16, 16, 16, 7, '2023-11-28 09:06:12', 30.97);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (17, 17, 17, 17, 8, '2023-01-11 08:13:12', 32.94);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (18, 18, 18, 18, 14, '2024-12-03 18:11:53', 14.51);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (19, 19, 19, 19, 4, '2023-07-20 06:27:30', 43.74);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (20, 20, 20, 20, 3, '2023-09-07 11:30:54', 15.11);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (21, 21, 21, 21, 11, '2023-09-05 06:32:48', 14.01);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (22, 22, 22, 22, 6, '2022-01-04 12:36:30', 42.25);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (23, 23, 23, 23, 1, '2024-05-23 22:17:50', 50.02);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (24, 24, 24, 24, 1, '2025-01-05 15:09:00', 15.34);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (25, 25, 25, 25, 12, '2023-09-22 00:49:14', 6.58);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (26, 26, 26, 26, 14, '2024-10-30 05:39:15', 24.75);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (27, 27, 27, 27, 13, '2022-08-14 06:38:20', 11.47);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (28, 28, 28, 28, 1, '2022-01-31 21:58:19', 21.55);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (29, 29, 29, 29, 2, '2024-01-07 16:20:17', 35.06);

insert into Sales (SaleID, VendorID, ProductID, CustomerID, QuantitySold, SaleDate, TotalAmount) values (30, 30, 30, 30, 8, '2022-11-27 11:11:39', 43.36);

```
farmer_market_managing_system> insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (20, 'Sadiree', 'guerrumbere@kaitrompany.com', 'Eggs', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (19, 'Liad', 'liamrae@plantmail.org', 'Bakery', '[illegible] Street Point');
INSERT 0 1
farmer_market_managing_system> insert into Vendors (VendorID, VendorName, ContactDetails, ProductCategory, StallLocation) values (18, 'Melisa', 'mmarkerh@mail.com', 'Dairy', '[illegible] Lane');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (1, 'Georgianna Cartin', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (2, 'Glenn de Leon', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (3, 'Maria Reese', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (4, 'Carrie Wiggley', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (5, 'Alex Flood', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (6, 'Lukmon Salvato', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (7, '[illegible]', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (8, 'Maria Mamaril', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (9, 'Brett Jaggrilli', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (10, 'Constanta Ramone', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (11, 'Dwanalisana Lookie', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (12, 'Colet Lathing', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (13, 'Kirstie Merlin', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (14, 'Tomoz Kilstone', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (15, 'Louise McCay', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (16, 'Adds Harkins', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (17, 'Bobs Cuttiffs', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (18, 'Andrew Saunders', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (19, 'Jasiem Rankiam', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (20, 'Josiah Bolderstone', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (21, 'Luke Wooton', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (22, 'Hawkins Abrajam', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (23, 'Deryle Lugar', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (24, 'Adrfield Fridlington', '[illegible]');
INSERT 0 1
```

```
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (25, 'Lapor Brown', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (26, 'Dian Spurdens', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (27, 'Clear Sparrow', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (28, 'Marquite Hirbings', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (29, 'Wallace Rackhoway', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into customers (CustomerID, CustomerName, ContactDetails) values (30, 'Lakesha Quinnelly', '[illegible]');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (1, 1, 1, 1, 'Great for managing market layout', '2021-04-21 18:11:12');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (2, 2, 2, 4, 'Great app', '2024-04-11 11:21:50');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (3, 3, 3, 3, 'Great for managing market layout', '2021-05-11 09:15:04');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (4, 4, 4, 4, 'Great for tracking customer feedback', '2022-07-24 09:31:02');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (5, 5, 5, 5, 'Great app', '2023-07-11 14:20:19');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (6, 6, 6, 3, 'Great for organizing market events', '2021-07-11 07:13:40');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (7, 7, 7, 5, 'Useful for tracking vendor performance', '2021-10-11 12:20:18');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (8, 8, 8, 3, 'Great for managing multiple markets', '2020-11-22 11:18:09');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (9, 9, 9, 5, 'Great for organizing market events', '2021-05-15 01:10:20');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (10, 10, 10, 4, 'Needs better integration with social media', '2011-07-19 01:01:10');

INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (11, 11, 11, 3, 'Great for tracking customer feedback', '2022-02-07 10:11:16');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (12, 12, 12, 4, 'Could use better customer support', '2022-11-24 11:11:13');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (13, 13, 13, 3, 'Needs more features for tracking sales', '2022-05-18 14:01:52');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (14, 14, 14, 4, 'Great for managing multiple markets', '2015-07-01 21:10:07');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (15, 15, 15, 5, 'Love the design and layout', '2022-08-11 01:11:12');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (16, 16, 16, 4, 'Needs better support for different languages', '2019-08-11 20:01:16'
);
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (17, 17, 17, 5, 'Could use better customer support', '2018-01-09 19:10:11');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (18, 18, 18, 1, 'Helpful for managing vendor applications', '2024-07-21 01:11:10');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (19, 19, 19, 1, 'Great for managing multiple markets', '2021-11-17 10:07:04');
INSERT 0 1
farmer_market_managing_system> insert into feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (20, 20, 20, 5, 'Love the integration with payment systems', '2011-01-05 04:57:40');
INSERT 0 1
```

```
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (21, 22, 24, 4, 'Great for managing multiple markets', '2023-12-07 18:12:31');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (22, 23, 24, 4, 'Helpful for managing vendor applications', '2020-10-24 02:10:02');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (23, 23, 23, 5, 'Could use more marketing tools', '2019-02-07 06:23:28');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (24, 24, 24, 4, 'Love the mobile app version', '2020-11-13 15:07:12');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (25, 25, 25, 4, 'Could use more analytics features', '2019-02-02 10:47:10');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (26, 26, 26, 5, 'Love the scheduling features', '2021-01-10 17:01:09');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (27, 27, 27, 5, 'Love the integration with payment systems', '2019-04-14 01:40:11');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (28, 28, 28, 5, 'Helpful for managing inventory', '2020-05-20 10:07:11');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (29, 29, 29, 5, 'Love the design and layout', '2019-12-19 21:17:07');
INSERT 0 1
farmer_market_managing_system-0 insert into Feedback (FeedbackID, CustomerID, VendorID, Rating, Comment, FeedbackDate) values (30, 30, 30, 4, 'Useful for tracking vendor performance', '2022-09-30 13:29:09');
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (1, 'Fresh Eggs', 'Fresh', 35.0, 1, 1);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (2, 'Handmade Soap', 'Honey', 13.77, 0, 2);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (3, 'Organic Apples', 'Dairy', 79.90, 7, 3);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (4, 'Handcrafted Jewelry', 'Flowers', 89.60, 4, 4);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (5, 'Farm-Fresh Milk', 'Baked Goods', 46.3, 15, 5);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (6, 'Handmade Baskets', 'Herbs', 65.22, 6, 6);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (7, 'Homemade Jam', 'Honey', 75.41, 12, 7);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (8, 'Artisan Bread', 'Soaps', 90.30, 11, 8);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (9, 'Local Honey', 'Meat', 33.13, 3, 9);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (10, 'Organic Apples', 'Fresh', 37.0, 5, 10);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (11, 'Handmade Baskets', 'Fields', 83.0, 1, 11);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (12, 'Homemade Jam', 'Dairy', 13.67, 8, 12);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (13, 'Handmade Baskets', 'Dairy', 26.95, 4, 13);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (14, 'Natural Skincare Products', 'Soaps', 9.36, 11, 14);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (15, 'Gourmet Sausages', 'Sauces', 36.45, 6, 15);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (16, 'Herbal Tea Blends', 'Jams and Jellies', 89.21, 4, 16);
INSERT 0 1
farmer_market_managing_system-0 insert into Products (ProductID, ProductName, Category, Price, QuantityAvailable, VendorID) values (17, 'Fresh Eggs', 'Crafts', 95.97, 1, 17);
INSERT 0 1
```

III. Data Querying & Manipulation

3.1. WHERE Filtration

CASE 1: Select vendors from a specific location

Description: Retrieves vendors based in "12273 Luster Center."

Code: *SELECT * FROM Vendors WHERE StallLocation = '12273 Luster Center';*

Scenario: The admin wants to identify all vendors operating in "12273 Luster Center."

Expected Result: A list of vendors located in "12273 Luster Center" is displayed.

```
farmer_market_managing_system=# SELECT * FROM Vendors WHERE StallLocation = '12273 Luster Center';
 vendorid | vendorname |    contactdetails     | productcategory |     stalllocation
----------+------------+-----------------------+-----------------+-----------------------
        1 | Mason      | mbirney0@reddit.com   | Crafts          | 12273 Luster Center
(1 row)
```

CASE 2: Find products below a specific price

Description: Lists all products priced under $20.
Code: *SELECT * FROM Products WHERE Price < 20;*
Scenario: A customer wants to see affordable products priced below
Expected Result: A list of products priced below 20 is displayed

```
farmer_market_managing_system=# SELECT * FROM Products WHERE Price < 20;
 productid |        productname         | category | price | quantityavailable | vendorid
-----------+----------------------------+----------+-------+-------------------+----------
         1 | Fresh Eggs                 | Fruits   | 19.00 |                 1 |        1
         2 | Handmade Soap              | Honey    | 15.37 |                 9 |        2
        10 | Organic Apples             | Fruits   | 17.80 |                 5 |       10
        12 | Homemade Jam               | Dairy    | 15.67 |                 9 |       12
        14 | Natural Skincare Products  | Soaps    |  9.36 |                12 |       14
        26 | Hand-poured Candles        | Eggs     | 11.44 |                 6 |       26
        27 | Fresh Eggs                 | Pickles  |  7.71 |                 6 |       27
        28 | Handwoven Baskets          | Crafts   | 14.29 |                12 |       28
(8 rows)
```

CASE 3: Identify high stock products
Description: Retrieves products with stock quantities exceeding 10 units.
Code: *SELECT * FROM Products WHERE QuantityAvailable > 10;*
Scenario: The market manager needs to know which products are overstocked
Expected Result: A list of products with stock quantities above 10 is displayed

```
farmer_market_managing_system=# SELECT * FROM Products WHERE QuantityAvailable > 10;
 productid |        productname         |     category     | price | quantityavailable | vendorid
-----------+----------------------------+------------------+-------+-------------------+----------
         5 | Farm-Fresh Milk            | Baked Goods      | 46.30 |                13 |        5
         7 | Homemade Jam               | Honey            | 75.43 |                12 |        7
         8 | Artisan Bread              | Soaps            | 90.18 |                13 |        8
        14 | Natural Skincare Products  | Soaps            |  9.36 |                12 |       14
        19 | Local Honey                | Crafts           | 51.48 |                15 |       19
        22 | Artisan Bread              | Jams and Jellies | 24.76 |                14 |       22
        24 | Handcrafted Jewelry        | Eggs             | 25.15 |                12 |       24
        25 | Natural Skincare Products  | Sauces           | 97.99 |                12 |       25
        28 | Handwoven Baskets          | Crafts           | 14.29 |                12 |       28
        29 | Fresh Eggs                 | Dairy            | 40.23 |                15 |       29
(10 rows)
```

CASE 4: Filter vendors by name pattern
Description: Retrieves vendors whose names start with "M."
Code: *SELECT * FROM Vendors WHERE VendorName LIKE 'M%';*
Scenario: The admin wants to identify vendors whose names start with the letter "M."
Expected Result: A list of vendors whose names start with "M" is displayed

```
farmer_market_managing_system=# SELECT * FROM Vendors WHERE VendorName LIKE 'M%';
 vendorid | vendorname |   contactdetails    | productcategory |    stalllocation
----------+------------+---------------------+------------------+----------------------
        1 | Mason      | mbirney0@reddit.com | Crafts           | 12273 Luster Center
        4 | Myrtle     | monion3@tmall.com   | Vegetables       | 481 Graedel Terrace
       25 | Margalo    | mrozziero@de.vu     | Meat             | 48 Golden Leaf Alley
       30 | Matias     | mdunfordt@ask.com   | Dairy            | 893 1st Lane
(4 rows)
```

CASE 5: Find sales on a specific date
Description: Retrieves all sales that occurred on '2023-09-22.'
Code: *SELECT * FROM Sales WHERE SaleDate = '2023-09-22';*

Scenario: The market manager needs to review sales transactions for September 22, 2023
Expected Result: A list of sales transactions from September 22, 2023, is displayed

```
farmer_market_managing_system=# SELECT * FROM Sales WHERE SaleDate = '2023-09-22';
 saleid | vendorid | productid | customerid | quantitysold |  saledate  | totalamount
--------+----------+-----------+------------+--------------+------------+-------------
     25 |       25 |        25 |         25 |           12 | 2023-09-22 |        6.58
(1 row)
```

3.2. String Functions

CASE 1: Convert vendor names to uppercase.
Description: Retrieves vendor names in uppercase for consistent formatting.
Code: *SELECT UPPER(VendorName) AS uppercased_name FROM Vendors;*
Scenario: The admin wants to display all vendor names in uppercase for a promotional banner.
Expected Result: A list of vendor names in uppercase is displayed.

```
farmer_market_managing_system=# SELECT UPPER(VendorName) AS uppercased_name FROM Vendors;
 uppercased_name
-----------------
 MASON
 JAMISON
 SCOT
 MYRTLE
 BARNIE
 FIORENZE
 KLIMENT
 VIN
 YNEZ
 GEARALT
 ASHIA
 FREDDY
 FAR
 SAM
 TIPHANI
 PHILIPPA
 BARBETTE
 LEE
 EVITA
 BENNETT
 ANESTASSIA
 AURELEA
 GERMAYNE
 BARBE
 MARGALO
 BANCROFT
 CODIE
 GODFREE
 LIND
 MATIAS
(30 rows)
```

CASE 2: Extract the first 3 characters of product names
Description: Retrieves the first 3 characters of each product's name for a short code.
Code: *SELECT SUBSTRING(ProductName, 1, 3) AS product_code FROM Products;*

Scenario: The market system needs to generate short codes for products.
Expected Result: A list of product codes (first 3 characters of each product's name) is displayed.

```
farmer_market_managing_system=# SELECT SUBSTRING(ProductName, 1, 3) AS product_code FROM Products;
 product_code
--------------
 Fre
 Han
 Org
 Han
 Far
 Han
 Hom
 Art
 Loc
 Org
 Han
 Hom
 Han
 Nat
 Gou
 Her
 Fre
 Loc
 Loc
 Fre
 Gou
 Art
 Hom
 Han
 Nat
 Han
 Fre
 Han
 Fre
 Far
(30 rows)
```

CASE 3: Replace a specific word in vendor names
Description: Replaces "Mason" with "Jonathan" in vendor names.
Code: *SELECT REPLACE(VendorName, 'Mason', 'Jonathan') AS updated_name FROM Vendors;*
Scenario: A vendor requested to update their name from "Mason" to "Jonathan."
Expected Result: A list of vendor names with "Mason" replaced by "Jonathan" is displayed.

```
farmer_market_managing_system=# SELECT REPLACE(VendorName, 'Mason', 'Jonathan') AS updated_name FROM Vendors;
 updated_name
--------------
 Jonathan
 Jamison
 Scot
 Myrtle
 Barnie
 Fiorenza
 Kilbent
 Vin
 Ynez
 Gearalt
 Ashia
 Freddy
 Far
 Sam
 Tiphani
 Philippa
 Barbette
 Lee
 Evita
 Bennett
 Anastassia
 Aurelea
 Germayne
 Barbe
 Margalo
 Bancroft
 Codie
 Godfree
 Lind
 Matias
(30 rows)
```

CASE 4: Count the length of product names

Description: Calculates the number of characters in each product's name.

Code: *SELECT ProductName, LENGTH(ProductName) AS name_length FROM Products;*

Scenario: The admin needs to know the length of product names to optimize database storage.

Expected Result: A list of product names and their respective lengths is displayed.

```
farmer_market_managing_system=# SELECT ProductName, LENGTH(ProductName) AS name_length FROM Products;
        productname         | name_length
----------------------------+-------------
 Fresh Eggs                 |          10
 Handmade Soap              |          13
 Organic Apples             |          14
 Handcrafted Jewelry        |          19
 Farm-Fresh Milk            |          15
 Handwoven Baskets          |          17
 Homemade Jam               |          12
 Artisan Bread              |          13
 Local Honey                |          11
 Organic Apples             |          14
 Handwoven Baskets          |          17
 Homemade Jam               |          12
 Handwoven Baskets          |          17
 Natural Skincare Products  |          25
 Gourmet Sausages           |          16
 Herbal Tea Blends          |          17
 Fresh Eggs                 |          10
 Local Honey                |          11
 Local Honey                |          11
 Fresh Eggs                 |          10
 Gourmet Sausages           |          16
 Artisan Bread              |          13
 Homemade Jam               |          12
 Handcrafted Jewelry        |          19
 Natural Skincare Products  |          25
 Hand-poured Candles        |          19
 Fresh Eggs                 |          10
 Handwoven Baskets          |          17
 Fresh Eggs                 |          10
 Farmhouse Cheese           |          16
(30 rows)
```

CASE 5: Concatenate vendor names with their locations

Description: Combines vendor names and their locations into a single field.

Code: *SELECT CONCAT(VendorName, ' - ', StallLocation) AS vendor_location FROM Vendors;*

Scenario: The system needs to display vendor names alongside their locations in a single field.

Expected Result: A list of concatenated strings showing vendor names and their locations is displayed.

```
farmer_market_managing_system=# SELECT CONCAT(VendorName, ' - ', StallLocation) AS vendor_location FROM Vendors;
         vendor_location
--------------------------------
 Mason - 12273 Luster Center
 Jamison - 991 Riverside Place
 Scot - 4603 Golf Course Lane
 Myrtle - 481 Graedel Terrace
 Barnie - 2521 Shasta Parkway
 Fiorenze - 4 Riverside Center
 Kliment - 6 Briar Crest Road
 Vin - 9 High Crossing Parkway
 Ynez - 58266 South Circle
 Gearalt - 5 Pepper Wood Place
 Ashia - 91044 Continental Court
 Freddy - 4 Gale Plaza
 Far - 503 Chinook Alley
 Sam - 8223 Briar Crest Park
 Tiphani - 32 Muir Park
 Philippa - 5 Dennis Plaza
 Barbette - 7 Carey Way
 Lee - 8110 Red Cloud Crossing
 Evita - 940 Kenwood Plaza
 Bennett - 3472 Thierer Plaza
 Anastassia - 335 Fulton Drive
 Aurelea - 29 Brown Court
 Germayne - 13 Columbus Parkway
 Barbe - 3 Laurel Alley
 Margalo - 48 Golden Leaf Alley
 Bancroft - 1970 Golf Way
 Codie - 30 Ridge Oak Plaza
 Godfree - 38443 Memorial Hill
 Lind - 232 Crowley Point
 Matias - 893 1st Lane
(30 rows)
```

3.3. Date Functions

CASE 1: Extract the year from sale dates

Description: Retrieves the year part from the SaleDate field in the Sales table.

Code: *SELECT SaleDate, EXTRACT(YEAR FROM SaleDate) AS sale_year FROM Sales;*

Scenario: The admin wants to analyze sales trends by year.

Expected Result:

```
farmer_market_managing_system=# SELECT SaleDate, EXTRACT(YEAR FROM SaleDate) AS sale_year FROM Sales;
  saledate   | sale_year
-------------+-----------
 2023-06-22 |      2023
 2023-09-30 |      2023
 2023-03-20 |      2023
 2023-03-12 |      2023
 2022-02-21 |      2022
 2024-12-24 |      2024
 2022-01-13 |      2022
 2024-08-23 |      2024
 2024-12-01 |      2024
 2023-02-08 |      2023
 2023-01-31 |      2023
 2023-04-10 |      2023
 2024-09-13 |      2024
 2023-06-26 |      2023
 2023-07-20 |      2023
 2023-11-28 |      2023
 2023-01-11 |      2023
 2024-12-03 |      2024
 2023-07-20 |      2023
 2023-09-07 |      2023
 2023-09-05 |      2023
 2022-01-04 |      2022
 2024-05-23 |      2024
 2025-01-05 |      2025
 2023-09-22 |      2023
 2024-10-30 |      2024
 2022-08-14 |      2022
 2022-01-31 |      2022
 2024-01-07 |      2024
 2022-11-27 |      2022
(30 rows)
```

CASE 2: Calculate the difference in days between sale dates and today

Description: Calculates the number of days between the SaleDate and the current date.

Code: *SELECT SaleDate, CURRENT_DATE - SaleDate AS days_since_sale FROM Sales;*

Scenario: The admin needs to identify how many days have passed since each sale.

Expected Result: A list of sale dates along with the number of days since each sale is displayed.

```
farmer_market_managing_system=# SELECT SaleDate, CURRENT_DATE - SaleDate AS days_since_sale FROM Sales;
  saledate   | days_since_sale
-------------+-----------------
 2023-06-22 |             608
 2023-09-30 |             508
 2023-03-20 |             702
 2023-03-12 |             710
 2022-02-21 |            1094
 2024-12-24 |              57
 2022-01-13 |            1133
 2024-08-23 |             180
 2024-12-01 |              80
 2023-02-08 |             742
 2023-01-31 |             750
 2023-04-10 |             681
 2024-09-13 |             159
 2023-06-26 |             604
 2023-07-20 |             580
 2023-11-28 |             449
 2023-01-11 |             770
 2024-12-03 |              78
 2023-07-20 |             580
 2023-09-07 |             531
 2023-09-05 |             533
 2022-01-04 |            1142
 2024-05-23 |             272
 2025-01-05 |              45
 2023-09-22 |             516
 2024-10-30 |             112
 2022-08-14 |             920
 2022-01-31 |            1115
 2024-01-07 |             409
 2022-11-27 |             815
(30 rows)
```

CASE 3: Format sale dates

Description: Formats sale dates in the Sales table to display as "Day-Month-Year".

Code: *SELECT SaleDate, TO_CHAR(SaleDate, 'DD-Month-YYYY') AS formatted_date FROM Sales;*

Scenario: The admin wants to view sales distribution by month.

Expected Result: A list of sale dates formatted as "Day-Month-Year" is displayed.

```
farmer_market_managing_system=# SELECT SaleDate, TO_CHAR(SaleDate, 'DD-Month-YYYY') AS formatted_date FROM Sales;
  saledate   |  formatted_date
-------------+------------------
 2023-06-22 | 22-June     -2023
 2023-09-30 | 30-September-2023
 2023-03-20 | 20-March    -2023
 2023-03-12 | 12-March    -2023
 2022-02-21 | 21-February -2022
 2024-12-24 | 24-December -2024
 2022-01-13 | 13-January  -2022
 2024-08-23 | 23-August   -2024
 2024-12-01 | 01-December -2024
 2023-02-08 | 08-February -2023
 2023-01-31 | 31-January  -2023
 2023-04-10 | 10-April    -2023
 2024-09-13 | 13-September-2024
 2023-06-26 | 26-June     -2023
 2023-07-20 | 20-July     -2023
 2023-11-28 | 28-November -2023
 2023-01-11 | 11-January  -2023
 2024-12-03 | 03-December -2024
 2023-07-20 | 20-July     -2023
 2023-09-07 | 07-September-2023
 2023-09-05 | 05-September-2023
 2022-01-04 | 04-January  -2022
 2024-05-23 | 23-May      -2024
 2025-01-05 | 05-January  -2025
 2023-09-22 | 22-September-2023
 2024-10-30 | 30-October  -2024
 2022-08-14 | 14-August   -2022
 2022-01-31 | 31-January  -2022
 2024-01-07 | 07-January  -2024
 2022-11-27 | 27-November -2022
(30 rows)
```

CASE 4: Extract the month name from sale dates

Description: Retrieves the full month name from the SaleDate field.

Code: *SELECT SaleDate, TO_CHAR(SaleDate, 'Month') AS sale_month FROM Sales;*

Scenario: The admin wants to view sales distribution by month.

Expected Result: A list of sale dates along with their respective month names is displayed.

```
farmer_market_managing_system=# SELECT SaleDate, TO_CHAR(SaleDate, 'Month') AS sale_month FROM Sales;
  saledate   | sale_month
-------------+------------
 2023-06-22 | June
 2023-09-30 | September
 2023-03-20 | March
 2023-03-12 | March
 2022-02-21 | February
 2024-12-24 | December
 2022-01-13 | January
 2024-08-23 | August
 2024-12-01 | December
 2023-02-08 | February
 2023-01-31 | January
 2023-04-10 | April
 2024-09-13 | September
 2023-06-26 | June
 2023-07-20 | July
 2023-11-28 | November
 2023-01-11 | January
 2024-12-03 | December
 2023-07-20 | July
 2023-09-07 | September
 2023-09-05 | September
 2022-01-04 | January
 2024-05-23 | May
 2025-01-05 | January
 2023-09-22 | September
 2024-10-30 | October
 2022-08-14 | August
 2022-01-31 | January
 2024-01-07 | January
 2022-11-27 | November
(30 rows)
```

CASE 5: Add 7 days to each sale date

Description: Calculates a new date by adding 7 days to each SaleDate.

Code: *SELECT SaleDate, SaleDate + INTERVAL '7 days' AS next_week FROM Sales;*

Scenario: The admin wants to identify dates a week after each sale for follow-up purposes.

Expected Result: A list of sale dates along with their respective dates 7 days later is displayed.

```
farmer_market_managing_system=# SELECT SaleDate, SaleDate + INTERVAL '7 days' AS next_week FROM Sales;
  saledate   |       next_week
-------------+---------------------
 2023-06-22  | 2023-06-29 00:00:00
 2023-09-30  | 2023-10-07 00:00:00
 2023-03-20  | 2023-03-27 00:00:00
 2023-03-12  | 2023-03-19 00:00:00
 2022-02-21  | 2022-02-28 00:00:00
 2024-12-24  | 2024-12-31 00:00:00
 2022-01-13  | 2022-01-20 00:00:00
 2024-08-23  | 2024-08-30 00:00:00
 2024-12-01  | 2024-12-08 00:00:00
 2023-02-08  | 2023-02-15 00:00:00
 2023-01-31  | 2023-02-07 00:00:00
 2023-04-10  | 2023-04-17 00:00:00
 2024-09-13  | 2024-09-20 00:00:00
 2023-06-26  | 2023-07-03 00:00:00
 2023-07-20  | 2023-07-27 00:00:00
 2023-11-28  | 2023-12-05 00:00:00
 2023-01-11  | 2023-01-18 00:00:00
 2024-12-03  | 2024-12-10 00:00:00
 2023-07-20  | 2023-07-27 00:00:00
 2023-09-07  | 2023-09-14 00:00:00
 2023-09-05  | 2023-09-12 00:00:00
 2022-01-04  | 2022-01-11 00:00:00
 2024-05-23  | 2024-05-30 00:00:00
 2025-01-05  | 2025-01-12 00:00:00
 2023-09-22  | 2023-09-29 00:00:00
 2024-10-30  | 2024-11-06 00:00:00
 2022-08-14  | 2022-08-21 00:00:00
 2022-01-31  | 2022-02-07 00:00:00
 2024-01-07  | 2024-01-14 00:00:00
 2022-11-27  | 2022-12-04 00:00:00
(30 rows)
```

3.4. Data Modification

CASE 1: Update the price of a product
Description: Changes the Price of a product in the Products table.
Code: *UPDATE Products SET Price = 4.0 WHERE ProductID = 2;*
Scenario: The price of a product with ID 2 needs to be updated to $4.0.
Expected Result: The Price for the product with ID 2 is updated to 4.0.

CASE 2: Increase the stock quantity of a product
Description: Increments the QuantityAvailable of a product in the Products table.
Code: *UPDATE Products SET QuantityAvailable = QuantityAvailable + 50 WHERE ProductID = 3;*
Scenario: The admin received additional stock of a product with ID 3 and wants to update the inventory.
Expected Result: The QuantityAvailable for the product with ID 3 increases by 50.

CASE 3: Mark a product as discontinued
Description: Updates the status of a product to "0" in the Products table.
Code: *UPDATE Products SET QuantityAvailable = 0 WHERE ProductID = 4;*
Scenario: A product with ID 4 is no longer being sold, and its status needs to reflect this change.
Expected Result: The available quantity of the product with ID 4 is updated to "0".

CASE 4: Change a vendor's contact information
Description: Updates the ContactDetails field for a specific vendor in the Vendors table.
Code: *UPDATE Vendors SET ContactDetails = '987-654-3210' WHERE VendorID = 1;*
Scenario: The contact number of the vendor with ID 1 has changed and needs to be updated.
Expected Result: The ContactDetails of the vendor with ID 1 is updated to "987-654-3210"

CASE 5: Update the sale record to correct a mistake
Description: Modifies the QuantitySold and TotalAmount for a specific sale in the Sales table.
Code: *UPDATE Sales SET QuantitySold = 15, TotalAmount = 45.0 WHERE SaleID = 5;*
Scenario: A sale record with ID 5 contains an error and needs correction for the quantity sold and total amount.
Expected Result: The QuantitySold and TotalAmount for the sale with ID 5 are updated to 15 and 45.0, respectively.

```
farmer_market_managing_system=# UPDATE Products SET Price = 4.0 WHERE ProductID = 2;
UPDATE 1
farmer_market_managing_system=# UPDATE Products SET QuantityAvailable = QuantityAvailable + 50 WHERE ProductID = 3;
UPDATE 1
farmer_market_managing_system=# UPDATE Products SET QuantityAvailable = 0 WHERE ProductID = 4;
UPDATE 1
farmer_market_managing_system=# UPDATE Vendors SET ContactDetails = '987-654-3210' WHERE VendorID = 1;
UPDATE 1
farmer_market_managing_system=# UPDATE Sales SET QuantitySold = 15, TotalAmount = 45.0 WHERE SaleID = 5;
UPDATE 1
```

3.5. Data Deletion

CASE 1: Delete a product
Description: Removes a product from the Products table.
Code: *DELETE FROM Products WHERE ProductID = 5;*
Scenario: A product with ID 5 is no longer available and needs to be removed from the system.
Expected Result: The row corresponding to the product with ID 5 is deleted from the Products table.

CASE 2: Remove a vendor from the system
Description: Deletes a vendor's record from the Vendors table.
Code: *DELETE FROM Vendors WHERE VendorID = 3;*
Scenario: A vendor with ID 3 has left the market and their information must be removed.
Expected Result: The row corresponding to the vendor with ID 3 is deleted from the Vendors table.

CASE 3: Delete sales records for a specific product
Description: Deletes all sales records related to a specific product in the Sales table.
Code: *DELETE FROM Sales WHERE ProductID = 2;*
Scenario: Sales records for the product with ID 2 need to be purged from the system.
Expected Result: All rows in the Sales table associated with the product ID 2 are removed.

CASE 4: Remove discontinued products

Description: Deletes products that have been marked as "discontinued" in the Products table.
Code: *SELECT * FROM products WHERE price_per_unit < 5;*
Scenario: Products marked as "discontinued" in the status field need to be removed from the inventory.
Expected Result: All rows in the Products table with the status "discontinued" are deleted.

CASE 5: Delete vendors from a specific location
Description: Deletes all vendors from a specified location in the Vendors table.
Code: *DELETE FROM Vendors WHERE StallLocation = 'Old Town';*
Scenario: Vendors operating in "Old Town" have ceased business, and their records must be deleted.
Expected Result: All rows in the Vendors table associated with the location "Old Town" are removed.

```
farmer_market_managing_system=# DELETE FROM Products WHERE ProductID = 5;
DELETE 1
farmer_market_managing_system=# DELETE FROM Vendors WHERE VendorID = 3;
DELETE 1
farmer_market_managing_system=# DELETE FROM Sales WHERE ProductID = 2;
DELETE 1
farmer_market_managing_system=# DELETE FROM Sales WHERE ProductID = 2;
DELETE 0
farmer_market_managing_system=# DELETE FROM Vendors WHERE StallLocation = 'Old Town';
DELETE 0
```

IV. SQL JOINs & Data Retrieval

4.1. INNER JOIN

CASE 1: Retrieve vendor names along with the products they sell
Description: This query joins the Vendors and Products tables to list each vendor and the products they sell.
Code: *SELECT v.VendorName, p.ProductName, p.Category FROM Vendors v INNER JOIN Products p ON v.VendorID = p.VendorID;*
Scenario: The market manager wants to identify which vendors are selling specific products to monitor supply.
Expected Result: Displays a list of vendors, the names of their products, and the product categories.

```
farmer_market_managing_system=# SELECT v.VendorName, p.ProductName, p.Category FROM Vendors v INNER JOIN Products p ON v.VendorID = p.VendorID;
 vendorname |        productname         |     category
------------+----------------------------+------------------
 Maxon      | Fresh Eggs                 | Fruits
 Fiorenza   | Handwoven Baskets          | Herbs
 Kliment    | Homemade Jam               | Honey
 Vin        | Artisan Bread              | Soaps
 Yvez       | Local Honey                | Meat
 Gwaralt    | Organic Apples             | Fruits
 Ashia      | Handwoven Baskets          | Pickles
 Freddy     | Homemade Jam               | Dairy
 Fur        | Handwoven Baskets          | Dairy
 Sam        | Natural Skincare Products  | Soaps
 Tiphani    | Gourmet Sausages           | Sauces
 Philippa   | Herbal Tea Blends          | Jams and Jellies
 Barbetta   | Fresh Eggs                 | Crafts
 Lew        | Local Honey                | Vegetables
 Evita      | Local Honey                | Crafts
 Bennett    | Fresh Eggs                 | Honey
 Anastasia  | Gourmet Sausages           | Fruits
 Aurelea    | Artisan Bread              | Jams and Jellies
 Germayne   | Homemade Jam               | Eggs
 Bambe      | Handcrafted Jewelry        | Eggs
 Margalo    | Natural Skincare Products  | Sauces
 Bancroft   | Hand-poured Candles        | Eggs
 Codie      | Fresh Eggs                 | Pickles
 Godfree    | Handwoven Baskets          | Crafts
 Lind       | Fresh Eggs                 | Dairy
 Matias     | Farmhouse Cheese           | Soaps
 Jamison    | Handmade Soap              | Honey
 Myrtle     | Handcrafted Jewelry        | Sauces
(18 rows)
```

CASE 2: Retrieve sales details along with the corresponding product names

Description: This query joins the Products and Sales tables to retrieve the names of products sold, the quantity sold, and the total price.

Code: *SELECT p.ProductName, s.QuantitySold, s.TotalAmount, s.SaleDate FROM Products p INNER JOIN Sales s ON p.ProductID = s.ProductID;*

Scenario: The market manager needs to analyze sales performance by reviewing the quantity and total revenue for each product sold on specific dates.

Expected Result: Displays a list of products sold, the quantity, total price, and the sale date.

```
farmer_market_managing_system=# SELECT p.ProductName, s.QuantitySold, s.TotalAmount, s.SaleDate FROM Products p INNER JOIN Sales s ON p.ProductID = s.ProductID;
        productname        | quantitysold | totalamount |  saledate
---------------------------+--------------+-------------+------------
 Fresh Eggs                |           15 |       67.83 | 2023-06-12
 Handcrafted Jewelry       |            6 |       51.89 | 2025-05-12
 Handwoven Baskets         |            2 |        6.43 | 2026-12-24
 Homemade Jam              |            6 |       23.31 | 2022-03-13
 Artisan Bread             |           13 |        8.37 | 2026-08-23
 Local Honey               |            5 |       43.35 | 2024-12-05
 Organic Apples            |           15 |       37.28 | 2023-02-08
 Handwoven Baskets         |            7 |       35.98 | 2025-03-31
 Homemade Jam              |           15 |        5.29 | 2023-04-10
 Handwoven Baskets         |            3 |        3.51 | 2024-08-13
 Natural Skincare Products |            6 |       21.28 | 2023-06-26
 Gourmet Sausages          |            8 |        8.42 | 2023-07-20
 Herbal Tea Blends         |            7 |       30.07 | 2023-11-28
 Fresh Eggs                |            8 |       52.94 | 2025-03-11
 Local Honey               |           14 |       14.51 | 2026-12-03
 Local Honey               |            4 |       43.24 | 2023-07-20
 Fresh Eggs                |            3 |       15.13 | 2023-08-07
 Gourmet Sausages          |           11 |       14.01 | 2023-09-05
 Artisan Bread             |            6 |       42.25 | 2022-03-04
 Homemade Jam              |            1 |       50.02 | 2024-05-13
 Handcrafted Jewelry       |            3 |       15.16 | 2025-03-05
 Natural Skincare Products |           12 |        6.58 | 2023-09-12
 Hand-poured Candles       |           14 |       24.75 | 2024-10-30
 Fresh Eggs                |           13 |       11.47 | 2022-08-14
 Handwoven Baskets         |            1 |       33.55 | 2022-03-11
 Fresh Eggs                |            2 |       35.06 | 2026-03-07
 Farmhouse Cheese          |            8 |       43.38 | 2022-11-27
(27 rows)
```

4.2. FULL JOIN

CASE 1: Retrieve all vendors and the products they sell (if any)

Description: This query uses a FULL JOIN between the Vendors and Products tables to list all vendors and the products they sell, including vendors without products and products not associated with any vendor.

Code: *SELECT v.VendorName, p.ProductName, p.Category FROM Vendors v FULL JOIN Products p ON v.VendorID = p.VendorID;*

Scenario: The admin wants a complete view of vendors and their associated products.

Expected Result: Displays all vendors with their corresponding products, including vendors without products and products without assigned vendors.

```
farmer_market_managing_system=# SELECT v.VendorName, p.ProductName, p.Category FROM Vendors v FULL JOIN Products p ON v.VendorID = p.VendorID;
 vendorname |        productname        |     category
------------+---------------------------+-----------------
 Moson      | Fresh Eggs                | Fruits
 Fiorenzo   | Handwoven Baskets         | Herbs
 Elbert     | Homemade Jam              | Honey
 Vin        | Artisan Bread             | Soaps
 Yves       | Local Honey               | Meat
 Gearalt    | Organic Apples            | Fruits
 Ashla      | Handwoven Baskets         | Pickles
 Freddy     | Homemade Jam              | Dairy
 Far        | Handwoven Baskets         | Dairy
 Sam        | Natural Skincare Products | Soaps
 Tiphani    | Gourmet Sausages          | Sauces
 Philippa   | Herbal Tea Blends         | Jams and Jellies
 Barbette   | Fresh Eggs                | Crafts
 Leo        | Local Honey               | Vegetables
 Evita      | Local Honey               | Crafts
 Bennett    | Fresh Eggs                | Honey
 Anastassia | Gourmet Sausages          | Fruits
 Aurelea    | Artisan Bread             | Jams and Jellies
 Germayne   | Homemade Jam              | Eggs
 Barbe      | Handcrafted Jewelry       | Eggs
 Margalo    | Natural Skincare Products | Sauces
 Bancroft   | Hand-poured Candles       | Eggs
 Codie      | Fresh Eggs                | Pickles
 Godfree    | Handwoven Baskets         | Crafts
 Lind       | Fresh Eggs                | Dairy
 Matias     | Farmhouse Cheese          | Soaps
 Jamison    | Handmade Soap             | Honey
 Myrtle     | Handcrafted Jewelry       | Sauces
 Bernie     |                           |
(29 rows)
```

CASE 2: Retrieve all products and their sales details (if any)

Description: This query uses a FULL JOIN between the Products and Sales tables to include all products and their sales details, even if some products have not been sold or some sales records have no associated product.

Code: *SELECT p.ProductName, s.QuantitySold, s.TotalAmount, s.SaleDate FROM Products p FULL JOIN Sales s ON p.ProductID = s.ProductID;*

Scenario: The market manager wants a comprehensive report of all products, including those not yet sold.

Expected Result: Displays all products with their sales details, including products without sales and sales without a matching product.

```
farmer_market_managing_system=# SELECT p.ProductName, s.QuantitySold, s.TotalAmount, s.SaleDate FROM Products p FULL JOIN Sales s ON p.ProductID = s.ProductID;
       productname        | quantitysold | totalamount |  saledate
--------------------------+--------------+-------------+------------
 Fresh Eggs               |           15 |       47.87 | 2023-06-22
 Handcrafted Jewelry      |            6 |       31.09 | 2023-01-12
 Handwoven Baskets        |            2 |        6.47 | 2024-12-24
 Homemade Jam             |            6 |       21.31 | 2023-01-13
 Artisan Bread            |           13 |        8.37 | 2024-08-23
 Local Honey              |            5 |       43.35 | 2024-12-01
 Organic Apples           |           15 |       33.28 | 2023-02-08
 Handwoven Baskets        |            7 |       35.00 | 2023-01-31
 Homemade Jam             |           15 |        5.29 | 2023-04-10
 Handwoven Baskets        |            1 |        3.52 | 2024-09-13
 Natural Skincare Products |            6 |       21.28 | 2023-06-26
 Gourmet Sausages         |            8 |        8.02 | 2023-07-24
 Herbal Tea Blends        |            7 |       38.97 | 2023-11-28
 Fresh Eggs               |            8 |       32.94 | 2023-01-31
 Local Honey              |           14 |       54.51 | 2024-12-03
 Local Honey              |            4 |       43.74 | 2023-07-28
 Fresh Eggs               |            3 |       15.11 | 2023-09-07
 Gourmet Sausages         |           11 |       14.01 | 2023-09-05
 Artisan Bread            |            6 |       42.25 | 2022-01-04
 Homemade Jam             |            1 |       50.02 | 2024-05-23
 Handcrafted Jewelry      |            1 |       25.38 | 2025-01-05
 Natural Skincare Products |           12 |        6.58 | 2023-09-22
 Hand-poured Candles      |           14 |       26.75 | 2024-10-30
 Fresh Eggs               |           15 |       51.47 | 2022-08-54
 Handwoven Baskets        |            3 |       21.55 | 2023-01-31
 Fresh Eggs               |            2 |       35.00 | 2024-01-07
 Farmhouse Cheese         |            8 |       43.36 | 2022-11-27
                          |           15 |       45.00 | 2023-02-21
                          |            1 |       46.50 | 2023-01-24
 Handmade Soap            |              |             |
(39 rows)
```

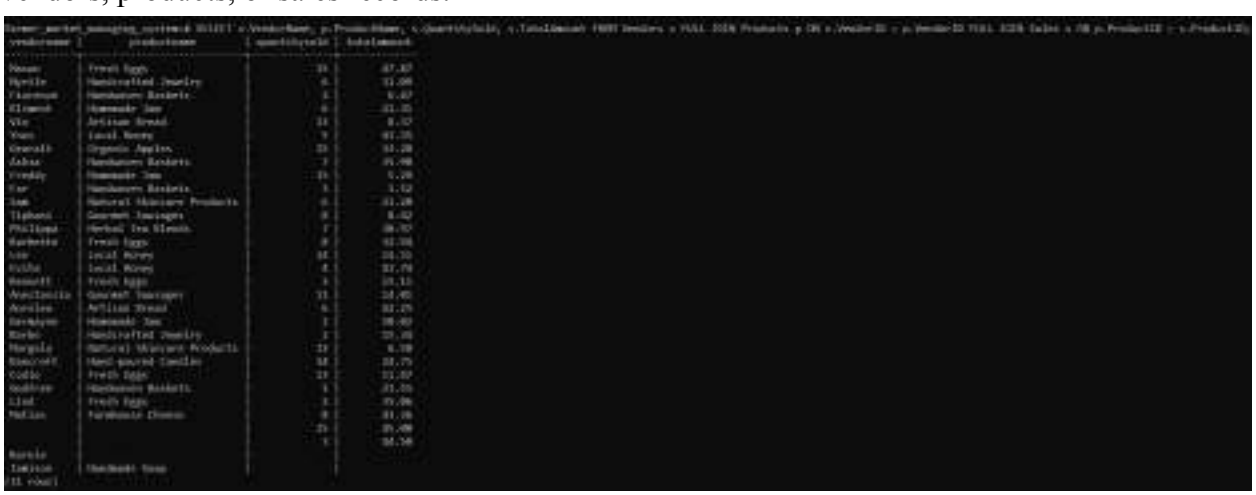
CASE 3: Retrieve all vendors and their sales through their products (if any)

Description: This query performs a FULL JOIN on the Vendors, Products, and Sales tables to retrieve a complete list of vendors and the sales details of their products, including vendors without products and products without sales.

Code: *SELECT v.VendorName, p.ProductName, s.QuantitySold, s.TotalAmount FROM Vendors v FULL JOIN Products p ON v.VendorID = p.VendorID FULL JOIN Sales s ON p.ProductID = s.ProductID;*

Scenario: The admin wants to generate a complete report linking vendors, their products, and sales data.

Expected Result: Displays all vendors, their products, and sales details, including unmatched vendors, products, or sales records.



4.3. LEFT JOIN

CASE 1: Retrieve all vendors and the products they sell (if any)

Description: This query uses a LEFT JOIN between the Vendors and Products tables to list all vendors and the products they sell, ensuring that all vendors are included even if they don't sell any products.

Code: *SELECT v.VendorName, p.ProductName, p.Category FROM Vendors v LEFT JOIN Products p ON v.VendorID = p.VendorID;*

Scenario: The admin wants to ensure that every vendor is included in the report, even if they haven't listed any products yet.

Expected Result: Displays all vendors with their corresponding products, including vendors without any products, which will show NULL for the ProductName and Category columns.

```
farmer_market_managing_system=# SELECT v.VendorName, p.ProductName, p.Category FROM Vendors v LEFT JOIN Products p ON v.VendorID = p.VendorID;
 vendorname |        productname         |     category
------------+----------------------------+-----------------
 Mason      | Fresh Eggs                 | Fruits
 Florence   | Handwoven Baskets          | Herbs
 Kliment    | Homemade Jam               | Honey
 Vio        | Artisan Bread              | Soaps
 Yoo2       | Local Honey                | Meat
 Gwaralt    | Organic Apples             | Fruits
 Achis      | Handwoven Baskets          | Pickles
 Freddy     | Homemade Jam               | Dairy
 Fur        | Handwoven Baskets          | Dairy
 Sam        | Natural Skincare Products  | Soaps
 Tiphani    | Gourmet Sausages           | Sauces
 Philippa   | Herbal Tea Blends          | Jams and Jellies
 Barbetta   | Fresh Eggs                 | Crafts
 Leo        | Local Honey                | Vegetables
 Evita      | Local Honey                | Crafts
 Bennett    | Fresh Eggs                 | Honey
 Anastasia  | Gourmet Sausages           | Fruits
 Aurelea    | Artisan Bread              | Jams and Jellies
 Germayne   | Homemade Jam               | Eggs
 Barbe      | Handcrafted Jewelry        | Eggs
 Margalo    | Natural Skincare Products  | Sauces
 Bancroft   | Hand-poured Candles        | Eggs
 Codie      | Fresh Eggs                 | Pickles
 Godfrey    | Handwoven Baskets          | Crafts
 Lind       | Fresh Eggs                 | Dairy
 Myrtles    | Farmhouse Cheese           | Soaps
 Jamison    | Handmade Soap              | Honey
 Myrtle     | Handcrafted Jewelry        | Sauces
 Barnie     |                            |
(29 rows)
```

CASE 2: Retrieve all products and their sales details (if any)

Description: This query uses a LEFT JOIN between the Products and Sales tables to list all products and their sales details, ensuring that all products are included even if they haven't been sold yet.

Code: *SELECT p.ProductName, s.QuantitySold, s.TotalAmount, s.SaleDate FROM Products p LEFT JOIN Sales s ON p.ProductID = s.ProductID;*

Scenario: The market manager wants a list of all products, including unsold ones.

Expected Result: Displays all products with their sales details, including products without any sales, which will show NULL for the QuantitySold, TotalAmount, and SaleDate columns.

```
farmer_market_managing_system=# SELECT p.ProductName, s.QuantitySold, s.TotalAmount, s.SaleDate FROM Products p LEFT JOIN Sales s ON p.ProductID = s.ProductID;
      productname        | quantitysold | totalamount |  saledate
-------------------------+--------------+-------------+------------
 Fresh Eggs              |           15 |       47.07 | 2023-06-22
 Handcrafted Jewelry     |            6 |       11.89 | 2023-03-12
 Handwoven Baskets       |            2 |        6.47 | 2026-12-24
 Homemade Jam            |            6 |       11.31 | 2022-01-13
 Artisan Bread           |           11 |        8.37 | 2026-08-23
 Local Honey             |            5 |       43.25 | 2024-12-01
 Organic Apples          |           15 |       13.28 | 2023-02-08
 Handwoven Baskets       |            7 |       15.98 | 2023-01-31
 Homemade Jam            |           15 |        5.28 | 2023-04-10
 Handwoven Baskets       |            3 |        3.53 | 2024-09-13
 Natural Skincare Products |          6 |       11.28 | 2023-06-26
 Gourmet Sausages        |            8 |        8.42 | 2023-07-20
 Herbal Tea Blends       |            7 |       10.97 | 2023-11-18
 Fresh Eggs              |            8 |       12.94 | 2023-01-15
 Local Honey             |           14 |       14.53 | 2026-12-03
 Local Honey             |            4 |       43.74 | 2023-07-10
 Fresh Eggs              |            3 |       15.11 | 2023-09-07
 Gourmet Sausages        |           11 |       14.01 | 2023-09-05
 Artisan Bread           |            6 |       42.25 | 2022-01-04
 Homemade Jam            |            1 |       50.02 | 2024-05-23
 Handcrafted Jewelry     |            1 |       15.14 | 2025-01-05
 Natural Skincare Products |         12 |        6.98 | 2023-09-22
 Hand-poured Candles     |           14 |       14.75 | 2026-10-30
 Fresh Eggs              |           13 |       11.47 | 2022-08-14
 Handwoven Baskets       |            1 |       11.55 | 2022-01-11
 Fresh Eggs              |            2 |       15.06 | 2024-03-07
 Farmhouse Cheese        |            8 |       43.16 | 2022-11-27
 Handmade Soap           |              |             |
(20 rows)
```

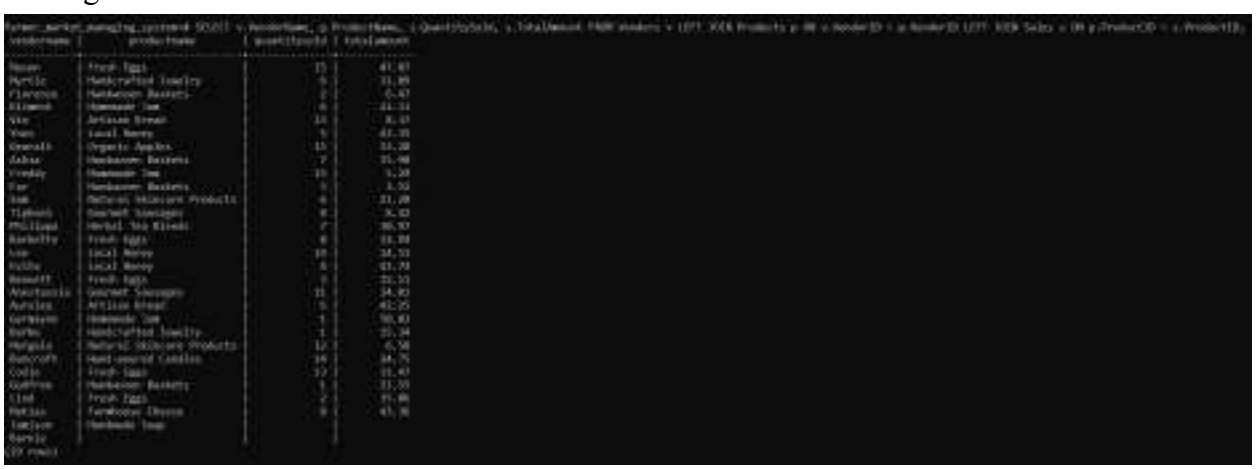
CASE 3: Retrieve all vendors and their sales through their products (if any)

Description: This query uses a LEFT JOIN between the Vendors, Products, and Sales tables to retrieve a list of all vendors and their sales details, ensuring that vendors without products or products without sales are included.

Code: *SELECT v.VendorName, p.ProductName, s.QuantitySold, s.TotalAmount FROM Vendors v LEFT JOIN Products p ON v.VendorID = p.VendorID LEFT JOIN Sales s ON p.ProductID = s.ProductID;*

Scenario: The admin wants to generate a report with all vendors, their products, and sales data.

Expected Result: Displays all vendors, their products, and sales details, including vendors without products and products without sales, where unmatched data will show NULL for the missing columns.



4.4. RIGHT JOIN

CASE 1: Retrieve all products and the vendors who sell them (if any)

Description: This query uses a RIGHT JOIN between the Products and Vendors tables to list all products and the vendors who sell them, ensuring that all products are included even if no vendor is currently associated with them.

Code: *SELECT p.ProductName, v.VendorName FROM Products p RIGHT JOIN Vendors v ON p.VendorID = v.VendorID;*

Scenario: The admin wants to ensure that every product is included in the report, even if it is not assigned to a vendor.

Expected Result: Displays all products with the corresponding vendor's name, including products without any assigned vendors, which will show NULL for the VendorName column.

```
farmer_market_managing_system=# SELECT p.ProductName, v.VendorName FROM Products p RIGHT JOIN Vendors v ON p.VendorID = v.VendorID;
        productname         | vendorname
----------------------------+------------
 Fresh Eggs                 | Mason
 Handwoven Baskets          | Florence
 Homemade Jam               | Klement
 Artisan Bread              | Vin
 Local Honey                | Yuez
 Organic Apples             | Gaarait
 Handwoven Baskets          | Ashia
 Homemade Jam               | Freddy
 Handwoven Baskets          | Far
 Natural Skincare Products  | Saw
 Gourmet Sausages           | Tiphani
 Herbal Tea Blends          | Philippa
 Fresh Eggs                 | Barbette
 Local Honey                | Luz
 Local Honey                | Evita
 Fresh Eggs                 | Bennett
 Gourmet Sausages           | Anestassia
 Artisan Bread              | Aurelea
 Homemade Jam               | Germayne
 Handcrafted Jewelry        | Barbe
 Natural Skincare Products  | Margaln
 Hand-poured Candles        | Bancroft
 Fresh Eggs                 | Codie
 Handwoven Baskets          | Godfree
 Fresh Eggs                 | Lind
 Farmhouse Cheese           | Matias
 Handmade Soap              | Jamison
 Handcrafted Jewelry        | Myrtle
                            | Bernie
(29 rows)
```

CASE 2: Retrieve all sales and the products they are associated with (if any)

Description: This query uses a RIGHT JOIN between the Sales and Products tables to list all sales and the corresponding products, ensuring that all sales records are included even if no matching product is found.

Code: *SELECT s.SaleDate, s.QuantitySold, p.ProductName FROM Sales s RIGHT JOIN Products p ON s.ProductID = p.ProductID;*

Scenario: The market manager wants to ensure all sales are included in the report, even if some sales might have missing product details.

Expected Result: Displays all sales with the corresponding product details, including sales without matching products, which will show NULL for the ProductName column.

```
farmer_market_managing_system=# SELECT s.SaleDate, s.QuantitySold, p.ProductName FROM Sales s RIGHT JOIN Products p ON s.ProductID = p.ProductID;
  saleDate  | quantitysold |       productname
------------+--------------+--------------------------
 2023-06-22 |           15 | Fresh Eggs
 2023-03-12 |            6 | Handcrafted Jewelry
 2024-12-24 |            2 | Handwoven Baskets
 2022-01-13 |            6 | Homemade Jam
 2024-08-23 |           13 | Artisan Bread
 2024-12-01 |            5 | Local Honey
 2023-02-08 |           15 | Organic Apples
 2023-01-11 |            7 | Handwoven Baskets
 2023-04-10 |           15 | Homemade Jam
 2024-09-13 |            3 | Handwoven Baskets
 2023-06-26 |            6 | Natural Skincare Products
 2023-07-20 |            8 | Gourmet Sausages
 2023-11-28 |            7 | Herbal Tea Blends
 2023-01-11 |            8 | Fresh Eggs
 2024-12-03 |           14 | Local Honey
 2023-07-20 |            4 | Local Honey
 2023-09-07 |            3 | Fresh Eggs
 2023-09-05 |           11 | Gourmet Sausages
 2022-01-04 |            6 | Artisan Bread
 2024-05-23 |            1 | Homemade Jam
 2025-01-05 |            1 | Handcrafted Jewelry
 2023-09-22 |           12 | Natural Skincare Products
 2024-10-30 |           14 | Hand-poured Candles
 2022-08-14 |           15 | Fresh Eggs
 2022-01-11 |            1 | Handwoven Baskets
 2024-01-07 |            2 | Fresh Eggs
 2022-11-27 |            8 | Farmhouse Cheese
            |              | Handmade Soap
(20 rows)
```

CASE 3: Retrieve all sales and the vendors who sold those products (if any)

Description: This query uses a RIGHT JOIN between the Sales, Products, and Vendors tables to list all sales and the corresponding vendors, ensuring that all sales are included even if no matching product or vendor is found.

Code: *SELECT s.SaleDate, s.QuantitySold, v.VendorName FROM Sales s RIGHT JOIN Products p ON s.ProductID = p.ProductID RIGHT JOIN Vendors v ON p.VendorID = v.VendorID;*

Scenario: The admin wants to ensure every sale is included in the report, even if some sales are associated with products that aren't linked to any vendor.

Expected Result: Displays all sales with the corresponding vendor details, including sales with missing vendor or product data, which will show NULL for those columns.

4.5. CROSS JOIN

CASE 1: Retrieve all combinations of vendors and products

Description: cThis query uses a CROSS JOIN between the Vendors and Products tables to generate a Cartesian product, showing all possible combinations of vendors and products.

Code: *SELECT v.VendorName, p.ProductName FROM Vendors v CROSS JOIN Products p;*

Scenario: The admin wants to analyze potential product offerings for each vendor.
Expected Result: Displays a list where each vendor is paired with every product, resulting in a large set of rows.

```
farmer_market_managing_system=# SELECT v.VendorName, p.ProductName FROM Vendors v CROSS JOIN Products p;
 vendorname |        productname
------------+---------------------------
 Jamison    | Fresh Eggs
 Myrtle     | Fresh Eggs
 Barnie     | Fresh Eggs
 Fiorenze   | Fresh Eggs
 Kliment    | Fresh Eggs
 Vin        | Fresh Eggs
 Ynez       | Fresh Eggs
 Gearalt    | Fresh Eggs
 Ashia      | Fresh Eggs
 Freddy     | Fresh Eggs
 Far        | Fresh Eggs
 Sam        | Fresh Eggs
 Tiphani    | Fresh Eggs
 Philippa   | Fresh Eggs
 Barbette   | Fresh Eggs
 Lee        | Fresh Eggs
 Evita      | Fresh Eggs
 Bennett    | Fresh Eggs
 Anestassia | Fresh Eggs
 Aurelea    | Fresh Eggs
 Germayne   | Fresh Eggs
 Barbe      | Fresh Eggs
 Margalo    | Fresh Eggs
 Bancroft   | Fresh Eggs
 Codie      | Fresh Eggs
 Godfree    | Fresh Eggs
 Lind       | Fresh Eggs
 Matias     | Fresh Eggs
 Mason      | Fresh Eggs
 Jamison    | Handwoven Baskets
 Myrtle     | Handwoven Baskets
 Barnie     | Handwoven Baskets
 Fiorenze   | Handwoven Baskets
 Kliment    | Handwoven Baskets
 Vin        | Handwoven Baskets
 Ynez       | Handwoven Baskets
 Gearalt    | Handwoven Baskets
 Ashia      | Handwoven Baskets
 Freddy     | Handwoven Baskets
 Far        | Handwoven Baskets
 Sam        | Handwoven Baskets
 Tiphani    | Handwoven Baskets
 Philippa   | Handwoven Baskets
 Barbette   | Handwoven Baskets
 Lee        | Handwoven Baskets
 Evita      | Handwoven Baskets
 Bennett    | Handwoven Baskets
 Anestassia | Handwoven Baskets
 Aurelea    | Handwoven Baskets
 Germayne   | Handwoven Baskets
 Barbe      | Handwoven Baskets
```

```
 Anestassia | Handcrafted Jewelry
 Aurelea    | Handcrafted Jewelry
 Germayne   | Handcrafted Jewelry
 Barbe      | Handcrafted Jewelry
 Margalo    | Handcrafted Jewelry
 Bancroft   | Handcrafted Jewelry
 Codie      | Handcrafted Jewelry
 Godfree    | Handcrafted Jewelry
 Lind       | Handcrafted Jewelry
 Matias     | Handcrafted Jewelry
 Mason      | Handcrafted Jewelry
(812 rows)
```

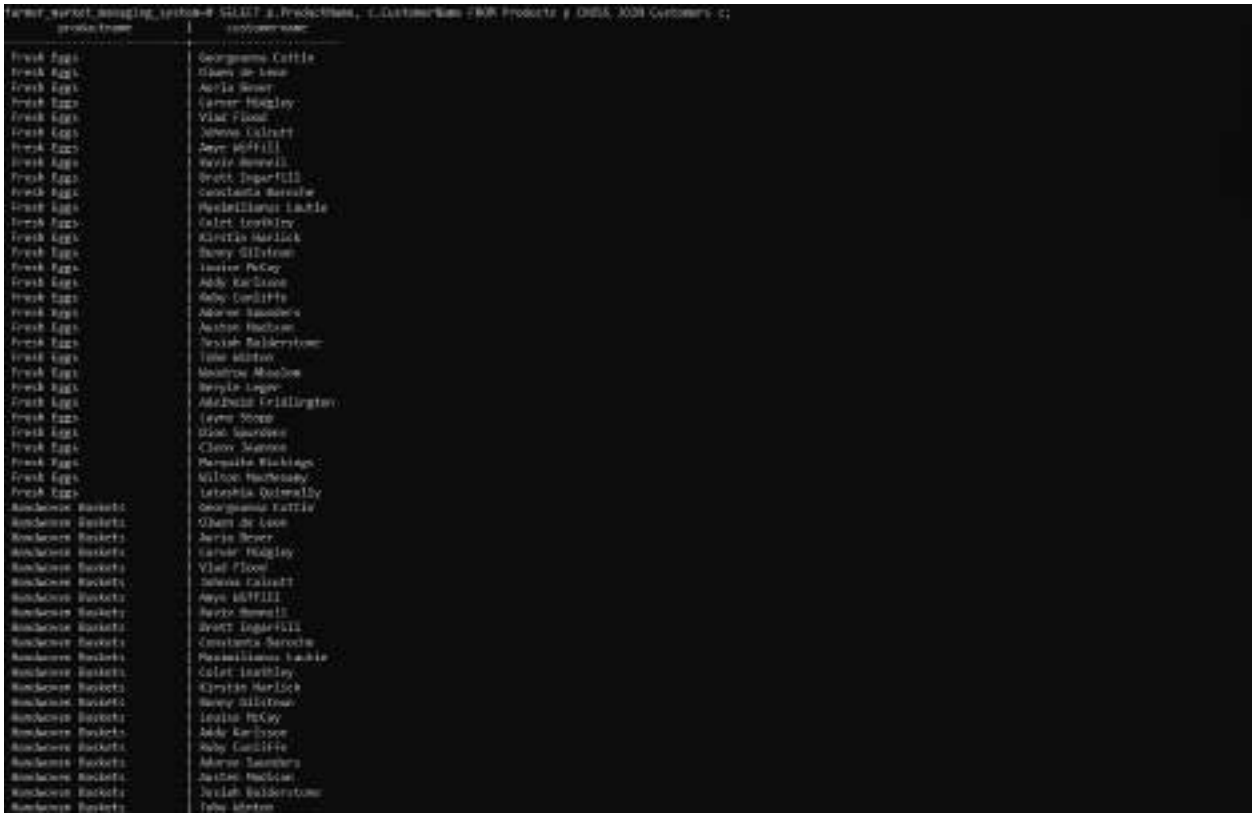
CASE 2: Retrieve all combinations of products and customers

Description: This query uses a CROSS JOIN between the Products and Customers tables to generate all possible combinations of products and customers.

Code: *SELECT p.ProductName, c.CustomerName FROM Products p CROSS JOIN Customers c;*

Scenario: The sales team wants to consider all customers for each product.

Expected Result: Displays a list of every product paired with every customer.



```
 Handcrafted Jewelry        | Adoree Saunders
 Handcrafted Jewelry        | Austen Madison
 Handcrafted Jewelry        | Josiah Balderstone
 Handcrafted Jewelry        | Tobe Winton
 Handcrafted Jewelry        | Woodrow Absalom
 Handcrafted Jewelry        | Beryle Leger
 Handcrafted Jewelry        | Adelheid Fridlington
 Handcrafted Jewelry        | Layne Stopp
 Handcrafted Jewelry        | Dion Spurdens
 Handcrafted Jewelry        | Cleon Jeannon
 Handcrafted Jewelry        | Marquita Richings
 Handcrafted Jewelry        | Wilton MacMenamy
 Handcrafted Jewelry        | Latashia Quinnelly
(840 rows)
```

CASE 3: Retrieve all combinations of vendors and customers

Description: This query uses a CROSS JOIN between the Vendors and Customers tables to generate a Cartesian product of all vendors and customers.

Code: *SELECT v.VendorName, c.CustomerName FROM Vendors v CROSS JOIN Customers c;*

Scenario: The admin wants to analyze potential customer-vendor relationships.

Expected Result: Displays a list where each vendor is paired with every customer.

```
 Barbette    | Latashia Quinnelly
 Lee         | Latashia Quinnelly
 Evita       | Latashia Quinnelly
 Bennett     | Latashia Quinnelly
 Anestassia  | Latashia Quinnelly
 Aurelea     | Latashia Quinnelly
 Germayne    | Latashia Quinnelly
 Barbe       | Latashia Quinnelly
 Margalo     | Latashia Quinnelly
 Bancroft    | Latashia Quinnelly
 Codie       | Latashia Quinnelly
 Godfree     | Latashia Quinnelly
 Lind        | Latashia Quinnelly
 Matias      | Latashia Quinnelly
 Mason       | Latashia Quinnelly
(870 rows)
```

4.6. NATURAL JOIN

CASE 1: Retrieve product and vendor details

Description: This query uses a NATURAL JOIN between the Products and Vendors tables to automatically match columns with the same name (e.g., VendorID) and retrieve the products and the vendors who sell them.

Code: *SELECT p.ProductName, v.VendorName FROM Products p NATURAL JOIN Vendors v;*

Scenario: The admin wants to retrieve a list of products and the corresponding vendors without explicitly specifying the matching column.

Expected Result: Displays a list of products paired with the corresponding vendors.

```
farmer_market_managing_system=#                                    SELECT p.ProductName, v.VendorName FROM Products p NATURAL JOIN Vendors v;
        productname        | vendorname
-------------------------+------------
 Fresh Eggs                | Mason
 Handwoven Baskets         | Florence
 Homemade Jam              | Element
 Artisan Bread             | Vin
 Local Honey               | Yves
 Organic Apples            | Gearalt
 Handwoven Baskets         | Ashia
 Homemade Jam              | Freddy
 Handwoven Baskets         | Far
 Natural Skincare Products | Sam
 Gourmet Sausages          | Tiphani
 Herbal Tea Blends         | Philippe
 Fresh Eggs                | Barbetta
 Local Honey               | Lee
 Local Honey               | Evita
 Fresh Eggs                | Bennett
 Gourmet Sausages          | Anastassia
 Artisan Bread             | Aurelina
 Homemade Jam              | Germayne
 Handcrafted Jewelry       | Berte
 Natural Skincare Products | Margalo
 Hand-poured Candles       | Bancroft
 Fresh Eggs                | Codie
 Handwoven Baskets         | Godfree
 Fresh Eggs                | Lind
 Farmhouse Cheese          | Matias
 Handmade Soap             | Tomlison
 Handcrafted Jewelry       | Myrtle
(28 rows)
```

CASE 2: Retrieve sales and product details

Description: This query uses a NATURAL JOIN between the Sales and Products tables to retrieve sales data along with product details. It relies on the common column ProductID to perform the join.

Code: *SELECT s.SaleDate, s.QuantitySold, p.ProductName FROM Sales s NATURAL JOIN Products p;*

Scenario: The market manager wants to retrieve all sales information along with the product details automatically.

Expected Result: Displays a list of sales transactions with corresponding product details.

```
farmer_market_managing_system=# SELECT s.SaleDate, s.QuantitySold, p.ProductName FROM Sales s NATURAL JOIN Products p;
  saledate   | quantitysold |        productname
-------------+--------------+-------------------------
 2023-06-22  |           15 | Fresh Eggs
 2023-03-12  |            6 | Handcrafted Jewelry
 2024-12-24  |            2 | Handwoven Baskets
 2022-01-13  |            6 | Homemade Jam
 2024-08-23  |           13 | Artisan Bread
 2024-12-01  |            5 | Local Honey
 2023-02-08  |           15 | Organic Apples
 2023-01-31  |            7 | Handwoven Baskets
 2023-04-10  |           15 | Homemade Jam
 2024-09-13  |            3 | Handwoven Baskets
 2023-06-20  |            6 | Natural Skincare Products
 2023-07-20  |            8 | Gourmet Sausages
 2023-11-28  |            7 | Herbal Tea Blends
 2023-01-11  |            8 | Fresh Eggs
 2024-12-03  |           14 | Local Honey
 2023-07-28  |            4 | Local Honey
 2023-09-07  |            3 | Fresh Eggs
 2023-09-05  |           11 | Gourmet Sausages
 2022-01-06  |            6 | Artisan Bread
 2024-05-23  |            1 | Homemade Jam
 2025-01-05  |            1 | Handcrafted Jewelry
 2023-09-22  |           12 | Natural Skincare Products
 2024-10-30  |           14 | Hand-poured Candles
 2022-08-14  |           13 | Fresh Eggs
 2022-01-31  |            1 | Handwoven Baskets
 2024-01-07  |            2 | Fresh Eggs
 2022-11-27  |            8 | Farmhouse Cheese
(27 rows)
```

CASE 3: Retrieve vendor, product, and sales details

Description: This query uses a NATURAL JOIN between the Vendors, Products, and Sales tables to retrieve a combined list of vendors, products, and sales, automatically matching columns with the same names (e.g., VendorID, ProductID).

Code: *SELECT v.VendorName, p.ProductName, s.SaleDate, s.QuantitySold FROM Vendors v NATURAL JOIN Products p NATURAL JOIN Sales s;*

Scenario: The admin wants a comprehensive report showing all vendors, their products, and sales data.

Expected Result: Displays a list of vendors, their products, and the sales data for each product.

```
farmer_market_managing_system=# SELECT v.VendorName, p.ProductName, s.SaleDate, s.QuantitySold FROM Vendors v NATURAL JOIN Products p NATURAL JOIN Sales s;
 vendorname |        productname        |  saledate  | quantitysold
------------+---------------------------+------------+--------------
 Mason      | Fresh Eggs                | 2023-06-22 |           15
 Myrtle     | Handcrafted Jewelry       | 2022-01-12 |            6
 Florence   | Handwoven Baskets         | 2024-12-24 |            2
 Kilment    | Homemade Jam              | 2022-01-13 |            6
 Vin        | Artisan Bread             | 2024-08-23 |           13
 Yves       | Local Honey               | 2024-12-03 |            5
 Gearalt    | Organic Apples            | 2023-02-08 |           15
 Ashla      | Handwoven Baskets         | 2023-01-31 |            7
 Freddy     | Homemade Jam              | 2023-04-10 |           15
 Far        | Handwoven Baskets         | 2024-05-13 |            3
 Sam        | Natural Skincare Products | 2023-06-26 |            6
 Tiphani    | Gourmet Sausages          | 2023-07-20 |            8
 Philiana   | Herbal Tea Blends         | 2023-11-28 |            7
 Harbette   | Fresh Eggs                | 2023-01-11 |            8
 Lee        | Local Honey               | 2024-12-03 |           14
 Erita      | Local Honey               | 2023-07-20 |            4
 Bennett    | Fresh Eggs                | 2023-08-07 |            3
 Anastassia | Gourmet Sausages          | 2023-09-05 |           11
 Aurelea    | Artisan Bread             | 2022-01-04 |            6
 Germayne   | Homemade Jam              | 2024-05-23 |            1
 Barbe      | Handcrafted Jewelry       | 2025-01-05 |            1
 Margalo    | Natural Skincare Products | 2023-06-22 |           12
 Bancroft   | Hand-poured Candles       | 2024-10-20 |           14
 Codie      | Fresh Eggs                | 2022-08-14 |           13
 Godfree    | Handwoven Baskets         | 2022-01-31 |            1
 Lind       | Fresh Eggs                | 2024-01-07 |            2
 Matias     | Farmhouse Cheese          | 2022-11-27 |            8
(27 rows)
```

4.7. SELF JOIN

CASE 1: Retrieve vendors and their stall locations

Description: This query uses a SELF JOIN on the Vendors table to retrieve pairs of vendors who share the same stall location.

Code: *SELECT v1.VendorName AS Vendor1, v2.VendorName AS Vendor2, v1.StallLocation FROM Vendors v1 JOIN Vendors v2 ON v1.StallLocation = v2.StallLocation WHERE v1.VendorID < v2.VendorID;*

Scenario: The admin wants to identify vendors who are located in the same stall area.

Expected Result: Displays pairs of vendors who share the same stall location.

```
farmer_market_managing_system=# SELECT v1.VendorName AS Vendor1, v2.VendorName AS Vendor2, v1.StallLocation FROM Vendors v1 JOIN Vendors v2 ON v1.StallLocation = v2.StallLocation WHERE v1.VendorID < v2.VendorID;
 vendor1 | vendor2 | stallLocation
---------+---------+---------------
(0 rows)
```

(because addresses are randomly generated)

CASE 2: Retrieve products with their categories

Description: This query uses a SELF JOIN on the Products table to find products and their categories.

Code: *SELECT p1.ProductName AS Product1, p2.ProductName AS Product2, p1.Category FROM Products p1 JOIN Products p2 ON p1.Category = p2.Category WHERE p1.ProductID < p2.ProductID;*

Scenario: The admin wants to analyze products within the same category.

Expected Result: Displays pairs of products within the same category.

CASE 3: Retrieve customers and their contact details

Description: This query uses a SELF JOIN on the Customers table to retrieve pairs of customers who share the same contact details.

Code: *SELECT c1.CustomerName AS Customer1, c2.CustomerName AS Customer2, c1.ContactDetails FROM Customers c1 JOIN Customers c2 ON c1.ContactDetails = c2.ContactDetails WHERE c1.CustomerID < c2.CustomerID;*

Scenario: The admin wants to identify customers with duplicate contact information.

Expected Result: Displays pairs of customers who share the same contact details.

(The same issue with CASE 1)

V. Query Execution & Results Analysis

5.1. Grouping Data (GROUP BY Clause)

CASE 1: Basic Aggregation (Grouping Products by Category and Calculating Total Stock)

Description: Groups products by their category and calculates the total stock quantity for each category, ordering the results by the total stock in descending order.

Code: *SELECT Category, SUM(QuantityAvailable) AS TotalStock FROM Products GROUP BY Category ORDER BY TotalStock DESC;*

Scenario: The market manager wants to analyze how many units of products are available in each category to determine stocking strategies.

Expected Result: The query displays a list of categories along with their total stock quantities, sorted from highest to lowest. For example, "Vegetables: 500 units", "Fruits: 300 units", etc.

```
farmer_market_managing_system=# SELECT Category, SUM(QuantityAvailable) AS TotalStock FROM Products GROUP BY Category ORDER BY TotalStock DESC;
     category     | totalstock
------------------+------------
 Dairy            |         33
 Soups            |         30
 Crafts           |         30
 Honey            |         28
 Eggs             |         24
 Jams and Jellies |         20
 Sauces           |         15
 Fruits           |         11
 Vegetables       |          8
 Pickles          |          7
 Herbs            |          6
 Meat             |          2
(12 rows)
```

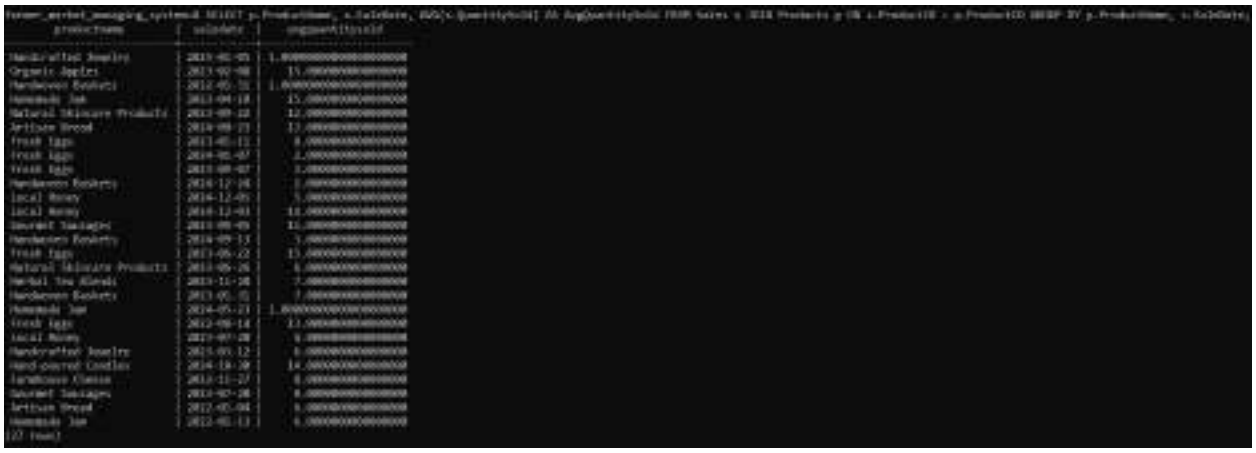
CASE 2: Multiple Column Grouping (Grouping Sales by Product Name and Sale Date to Calculate Average Quantity Sold)

Description: Groups sales by product name and sale date, calculating the average quantity sold for each group.

Code: *SELECT p.ProductName, s.SaleDate, AVG(s.QuantitySold) AS AvgQuantitySold FROM Sales s JOIN Products p ON s.ProductID = p.ProductID GROUP BY p.ProductName, s.SaleDate;*

Scenario: The market manager wants to understand sales trends by analyzing the average quantity sold for each product on specific dates.

Expected Result: The query provides a list of product names, sale dates, and average quantities sold, offering insights into sales performance trends. For instance, "Tomatoes, January 10, 2025, Avg: 8 units".



CASE 3: GROUP BY with HAVING (Grouping Sales by Product Name with Filter for High Sales Volume)

Description: Groups sales by product name and filters the results to only include products with a total quantity sold greater than 20.

Code: *SELECT p.ProductName, SUM(s.QuantitySold) AS TotalQuantitySold FROM Sales s JOIN Products p ON s.ProductID = p.ProductID GROUP BY p.ProductName HAVING SUM(s.QuantitySold) > 20;*

Scenario: The market manager wants to identify the best-performing products by viewing only those that have sold more than 20 units.

Expected Result: The query displays product names and total quantities sold, filtered to show only products with a high sales volume. For instance, "Tomatoes, Total Sold: 35 units".

CASE 4: JOIN + GROUPING (Join Vendors and Products, Group by Vendor Name, and Calculate Total Stock)

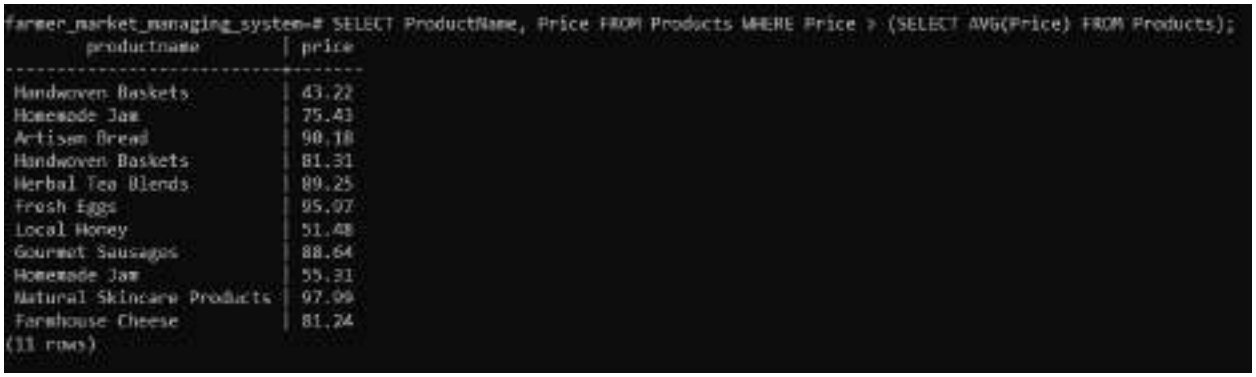
Description: Joins the Vendors and Products tables to group products by vendor, calculating the total stock each vendor currently has.

Code: *SELECT v.VendorName, SUM(p.QuantityAvailable) AS TotalStock FROM Vendors v JOIN Products p ON v.VendorID = p.VendorID GROUP BY v.VendorName ORDER BY TotalStock DESC;*

Scenario: The market manager wants to see how much stock each vendor currently holds.
Expected Result: The query displays vendor names and the sum of their product stocks in descending order. For example, "John Doe, Total Stock: 150 units".

5.2. Subqueries

CASE 1: Scalar Subquery (Find Products Priced Higher than the Average Price)
Description: Uses a scalar subquery to find products whose prices are higher than the average product price.
Code: *SELECT ProductName, Price FROM Products WHERE Price > (SELECT AVG(Price) FROM Products);*
Scenario: The market manager wants to identify premium-priced products.
Expected Result: The query returns the names and prices of products priced above the market average, such as "Organic Apples, $4.5 per unit".



```
farmer_market_managing_system=# SELECT ProductName, Price FROM Products WHERE Price > (SELECT AVG(Price) FROM Products);
        productname         | price
----------------------------+-------
 Handwoven Baskets          | 43.22
 Homemade Jam               | 75.43
 Artisan Bread              | 90.18
 Handwoven Baskets          | 81.31
 Herbal Tea Blends          | 89.25
 Fresh Eggs                 | 95.07
 Local Honey                | 51.48
 Gourmet Sausages           | 88.64
 Homemade Jam               | 55.31
 Natural Skincare Products  | 97.09
 Farmhouse Cheese           | 81.24
(11 rows)
```

CASE 2: IN Subquery (Find Vendors Selling Multiple Products)
Description: Uses the IN operator with a subquery to find vendors who have listed products for sale.
Code: *SELECT VendorName FROM Vendors WHERE VendorID IN (SELECT DISTINCT VendorID FROM Products);*
Scenario: The admin wants to see a list of vendors who are actively selling products.
Expected Result: The query returns the names of vendors who have products in the system, such as "John Doe" and "Jane Smith".

```
farmer_market_managing_system=# SELECT VendorName FROM Vendors WHERE VendorID IN (SELECT DISTINCT VendorID FROM Products);
 vendorname
------------
 Aureles
 Ashia
 Ynez
 Tiphani
 Bancroft
 Evita
 Matias
 Anastassia
 Barbette
 Godfree
 Lind
 Myrtle
 Goarult
 Fiorenze
 Sam
 Far
 Jamison
 Philippa
 Kliment
 Freddy
 Barbe
 Margalo
 Bennett
 Mason
 Lee
 Codie
 Germayne
 Vin
(28 rows)
```

CASE 3: Correlated Subquery (Retrieve Products Priced Higher Than the Average Price per Category)

Description: This query uses a correlated subquery in the WHERE clause to retrieve products priced above the average price for their respective categories.

Code: *SELECT ProductName, Category, Price FROM Products p1 WHERE Price > (SELECT AVG(Price) FROM Products p2 WHERE p1.Category = p2.Category);*

Scenario: The market manager wants to identify premium-priced products compared to the category average.

Expected Result: The query returns a list of products with prices above the category average, such as "Organic Tomatoes" for the "Vegetables" category.

```
farmer_market_managing_system=# SELECT ProductName, Category, Price FROM Products p1 WHERE Price > (SELECT AVG(Price) FROM Products p2 WHERE p1.Category = p2.Category);
        productname        |     category     | price
-------------------------+------------------+-------
 Homemade Jam              | Honey            | 75.45
 Artisan Bread             | Soaps            | 90.10
 Handwoven Baskets         | Pickles          | 81.31
 Herbal Tea Blends         | Jams and Jellies | 89.25
 Fresh Eggs                | Crafts           | 95.97
 Fresh Eggs                | Honey            | 41.82
 Gourmet Sausages          | Fruits           | 88.64
 Homemade Jam              | Eggs             | 55.31
 Natural Skincare Products | Soaps            | 97.90
 Fresh Eggs                | Dairy            | 46.24
 Farmhouse Cheese          | Soaps            | 83.24
(13 rows)
```

CASE 4: EXISTS Subquery (Retrieve Vendors Who Sell Organic Products)

Description: This query uses a subquery with the EXISTS clause to find vendors that sell products in the "Organic" category. The subquery checks for the existence of products with the "Organic" category for each vendor.

Code: *SELECT v.VendorName, v.StallLocation FROM Vendors v WHERE EXISTS (SELECT 1 FROM Products p WHERE p.VendorID = v.VendorID AND p.Category = 'Vegetables');*

Scenario: The market manager wants to identify vendors who sell organic products, as customers are increasingly interested in organic options.

Expected Result: The query returns a list of vendors who have at least one organic product, such as "Organic Roots" located at "Stall C3".

```
farmer_market_managing_system-# SELECT v.VendorName, v.StallLocation
farmer_market_managing_system-# FROM Vendors v
farmer_market_managing_system-# WHERE EXISTS (SELECT 1 FROM Products p WHERE p.VendorID = v.VendorID AND p.Category = 'Vegetables');
 vendorname |      stallLocation
------------+-------------------------
 Lee        | 0130 Red Cloud Crossing
(1 row)
```

CASE 5: ALL Subquery (Retrieve Sales Amounts Greater than All Sales of Vendor 1)

Description: This query compares the sales amounts from all vendors to the sales of Vendor 1. It uses a subquery with the ALL keyword to filter out any sales that are not greater than every sale from Vendor 1.

Code: *SELECT VendorID, TotalAmount FROM Sales WHERE TotalAmount > ALL (SELECT TotalAmount FROM Sales WHERE VendorID = 1);*

Scenario: The market manager wants to identify all sales transactions from vendors where the total sales amount exceeds the highest sale amount of Vendor 1.

Expected Result: The query will return a list of sales transactions where the TotalAmount is greater than all the sales amounts from Vendor 1. For example, it could return sales such as "Vendor 2, TotalAmount 1500; Vendor 3, TotalAmount 2000".

```
farmer_market_managing_system-# SELECT VendorID, TotalAmount FROM Sales WHERE TotalAmount > ALL (SELECT TotalAmount FROM Sales WHERE VendorID = 1);
 vendorid | totalamount
----------+-------------
       21 |       50.02
(1 row)
```

CASE 6: Row Subquery (Retrieve Sales Records with VendorID and TotalAmount for a Specific Sales Date)

Description: This query uses a row subquery to compare multiple columns in the Sales table (i.e., VendorID and TotalAmount) against specific values from a corresponding subquery.

Code: *SELECT VendorID, TotalAmount, SaleDate FROM Sales WHERE (VendorID, TotalAmount) = (SELECT VendorID, TotalAmount FROM Sales WHERE SaleDate = '2023-06-22' LIMIT 1);*

Scenario: A market manager wants to find sales records that match a specific vendor and total amount for a given sales date. The subquery is used to retrieve the values that must be matched in the main query.

Expected Result: The query returns the VendorID, TotalAmount, and SaleDate from the Sales table where the VendorID and TotalAmount match those from a sale on the specified date (2023-06-22). Only the sale that matches this combination will be returned.

```
farmer_market_managing_system-# SELECT VendorID, TotalAmount, SaleDate FROM Sales WHERE (VendorID, TotalAmount) = (SELECT VendorID, TotalAmount FROM Sales WHERE SaleDate = '2023-06-22' LIMIT 1);
 vendorid | totalamount |  saledate
----------+-------------+------------
        1 |       87.87 | 2023-06-22
(1 row)
```

VI. Advanced SQL Techniques

6.1. Window functions

CASE 1: Calculate the difference between the current row's value and the previous row's value in a numeric column.

Description: This query calculates the difference between the current row's value and the previous row's value in the `TotalAmount` column of the `Sales` table. The data is partitioned by `VendorID` and ordered by `SaleDate`. The `LAG` window function is used to retrieve the previous row's value, and the difference is calculated.

Scenario: A vendor wants to track the difference in sales amounts between consecutive sales dates to identify trends or anomalies.

Code: *SELECT SaleID, VendorID, SaleDate, TotalAmount, LAG(TotalAmount) OVER (PARTITION BY VendorID ORDER BY SaleDate) AS PreviousAmount, TotalAmount - LAG(TotalAmount) OVER (PARTITION BY VendorID ORDER BY SaleDate) AS AmountDifference FROM Sales;*

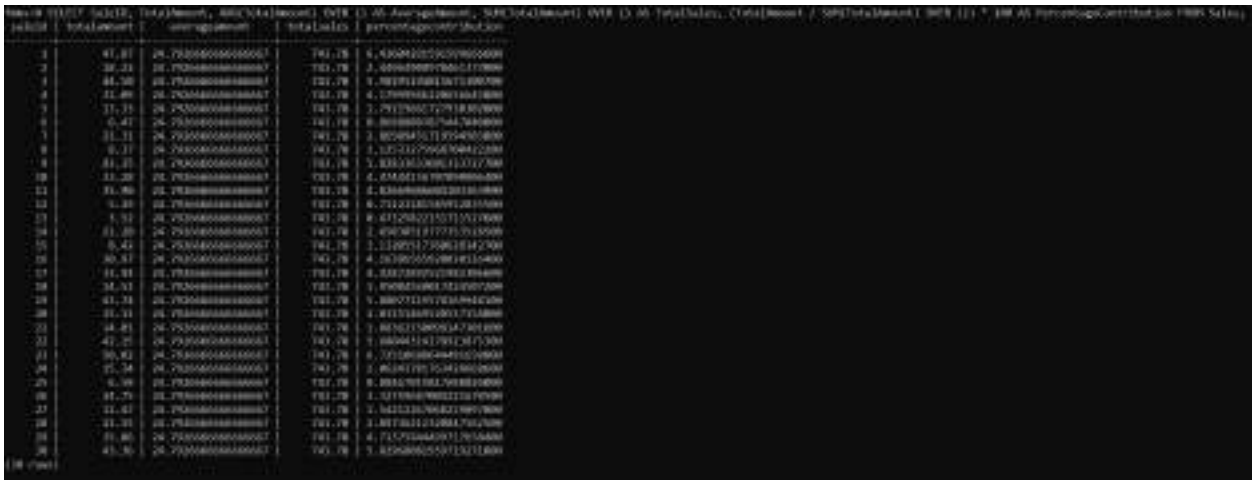
Expected Result: The result will show each sale's `SaleID`, `VendorID`, `SaleDate`, `TotalAmount`, the previous sale's `TotalAmount` (`PreviousAmount`), and the difference between the current and previous amounts (`AmountDifference`).

saleid	vendorid	saledate	totalamount	previousamount	amountdifference
1	1	2023-06-22	47.87		
2	2	2023-09-30	18.22		
3	3	2023-03-20	44.50		
4	4	2023-03-12	31.09		
5	5	2022-02-21	13.33		
6	6	2024-12-24	6.47		
7	7	2022-01-13	21.31		
8	8	2024-08-23	8.37		
9	9	2024-12-01	43.35		
10	10	2023-02-08	33.28		
11	11	2023-01-31	35.90		
12	12	2023-04-10	5.29		
13	13	2024-09-13	3.52		
14	14	2023-06-26	21.20		
15	15	2023-07-20	8.42		
16	16	2023-11-28	30.97		
17	17	2023-01-11	32.94		
18	18	2024-12-03	14.51		
19	19	2023-07-20	43.74		
20	20	2023-09-07	15.11		
21	21	2023-09-05	14.01		
22	22	2022-01-04	42.25		
23	23	2024-05-23	50.02		
24	24	2025-01-05	15.34		
25	25	2023-09-22	6.58		
26	26	2024-10-30	24.75		
27	27	2022-08-14	11.47		
28	28	2022-01-31	21.55		
29	29	2024-01-07	35.06		
30	30	2022-11-27	43.36		

(30 rows)

CASE 2: Calculate the average value of a numeric column across all rows, including each row's value and its percentage contribution to the total.

Description: This query calculates the average `TotalAmount` across all rows in the `Sales` table and includes each row's `TotalAmount` and its percentage contribution to the total. The `AVG` and `SUM` window functions are used to calculate the average and total, respectively.
Scenario: A market manager wants to analyze the percentage contribution of each sale to the total sales and compare it to the average sale amount.
Code: *SELECT SaleID, TotalAmount, AVG(TotalAmount) OVER () AS AverageAmount, SUM(TotalAmount) OVER () AS TotalSales, (TotalAmount / SUM(TotalAmount) OVER ()) * 100 AS PercentageContribution FROM Sales;*
Expected Result: The result will show each sale's `SaleID`, `TotalAmount`, the average `TotalAmount` across all sales (`AverageAmount`), the total sales (`TotalSales`), and the percentage contribution of each sale to the total sales (`PercentageContribution`).



CASE 3: Calculate the moving average of a numeric column over a 3-row window, ordered by a date or timestamp column.
Description: This query calculates the moving average of the `TotalAmount` column over a 3-row window, ordered by `SaleDate`. The data is partitioned by `VendorID` to calculate the moving average for each vendor separately. The `AVG` window function is used with the `ROWS BETWEEN 2 PRECEDING AND CURRENT ROW` clause to define the window.
Scenario: A vendor wants to track the moving average of their sales over the last three sales dates to identify trends.
Code: *SELECT SaleID, VendorID, SaleDate, TotalAmount, AVG(TotalAmount) OVER (PARTITION BY VendorID ORDER BY SaleDate ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAverage FROM Sales;*
Expected Result: The result will show each sale's `SaleID`, `VendorID`, `SaleDate`, `TotalAmount`, and the moving average of `TotalAmount` over the last three sales dates (`MovingAverage`).

```
fmms=# SELECT SaleID, VendorID, SaleDate, TotalAmount, AVG(TotalAmoun
 saleid | vendorid |  saledate  | totalamount |     movingaverage
--------+----------+------------+-------------+---------------------
      1 |        1 | 2023-06-22 |       47.87 | 47.8700000000000000
      2 |        2 | 2023-09-30 |       18.22 | 18.2200000000000000
      3 |        3 | 2023-03-20 |       44.50 | 44.5000000000000000
      4 |        4 | 2023-03-12 |       31.09 | 31.0900000000000000
      5 |        5 | 2022-02-21 |       13.33 | 13.3300000000000000
      6 |        6 | 2024-12-24 |        6.47 |  6.4700000000000000
      7 |        7 | 2022-01-13 |       21.31 | 21.3100000000000000
      8 |        8 | 2024-08-23 |        8.37 |  8.3700000000000000
      9 |        9 | 2024-12-01 |       43.35 | 43.3500000000000000
     10 |       10 | 2023-02-08 |       33.28 | 33.2800000000000000
     11 |       11 | 2023-01-31 |       35.90 | 35.9000000000000000
     12 |       12 | 2023-04-10 |        5.29 |  5.2900000000000000
     13 |       13 | 2024-09-13 |        3.52 |  3.5200000000000000
     14 |       14 | 2023-06-26 |       21.20 | 21.2000000000000000
     15 |       15 | 2023-07-20 |        8.42 |  8.4200000000000000
     16 |       16 | 2023-11-28 |       30.97 | 30.9700000000000000
     17 |       17 | 2023-01-11 |       32.94 | 32.9400000000000000
     18 |       18 | 2024-12-03 |       14.51 | 14.5100000000000000
     19 |       19 | 2023-07-20 |       43.74 | 43.7400000000000000
     20 |       20 | 2023-09-07 |       15.11 | 15.1100000000000000
     21 |       21 | 2023-09-05 |       14.01 | 14.0100000000000000
     22 |       22 | 2022-01-04 |       42.25 | 42.2500000000000000
     23 |       23 | 2024-05-23 |       50.02 | 50.0200000000000000
     24 |       24 | 2025-01-05 |       15.34 | 15.3400000000000000
     25 |       25 | 2023-09-22 |        6.58 |  6.5800000000000000
     26 |       26 | 2024-10-30 |       24.75 | 24.7500000000000000
     27 |       27 | 2022-08-14 |       11.47 | 11.4700000000000000
     28 |       28 | 2022-01-31 |       21.55 | 21.5500000000000000
     29 |       29 | 2024-01-07 |       35.06 | 35.0600000000000000
     30 |       30 | 2022-11-27 |       43.36 | 43.3600000000000000
(30 rows)
```

6.1. Window functions

CASE 1: Group data using GROUPING SETS to calculate the sum of a numeric column.

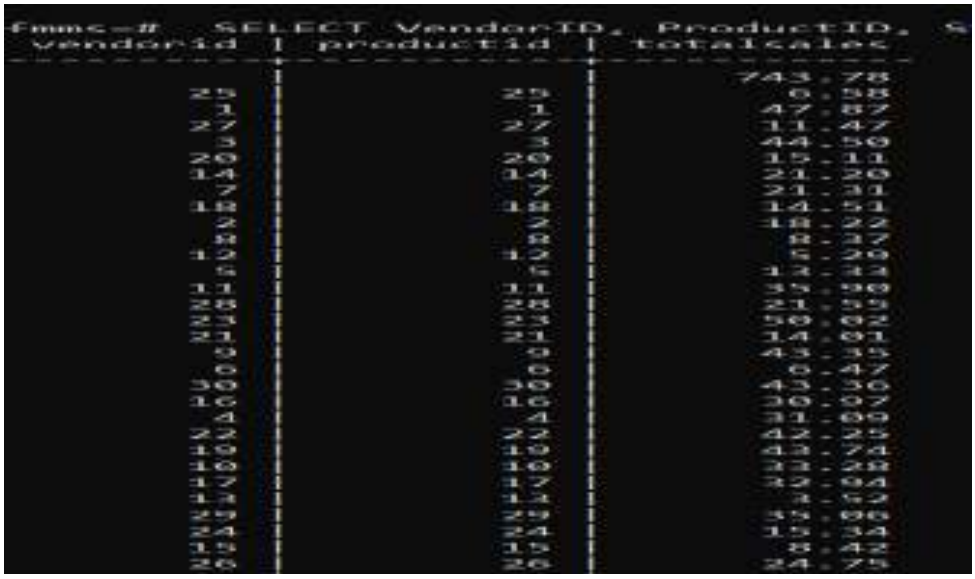
Description: This query uses `GROUPING SETS` to calculate the sum of the `TotalAmount` column in the `Sales` table for:

1. Each `VendorID`.
2. Each `ProductID`.
3. All combinations of `VendorID` and `ProductID`.
4. A grand total.

Scenario: A market manager wants to analyze sales totals by vendor, by product, and by vendor-product combinations, as well as the overall grand total.

Code: *SELECT VendorID, ProductID, SUM(TotalAmount) AS TotalSales FROM Sales GROUP BY GROUPING SETS ((VendorID), (ProductID), (VendorID, ProductID), ());*

Expected Result: The result will show the sum of `TotalAmount` for each `VendorID`, each `ProductID`, each combination of `VendorID` and `ProductID`, and the grand total.



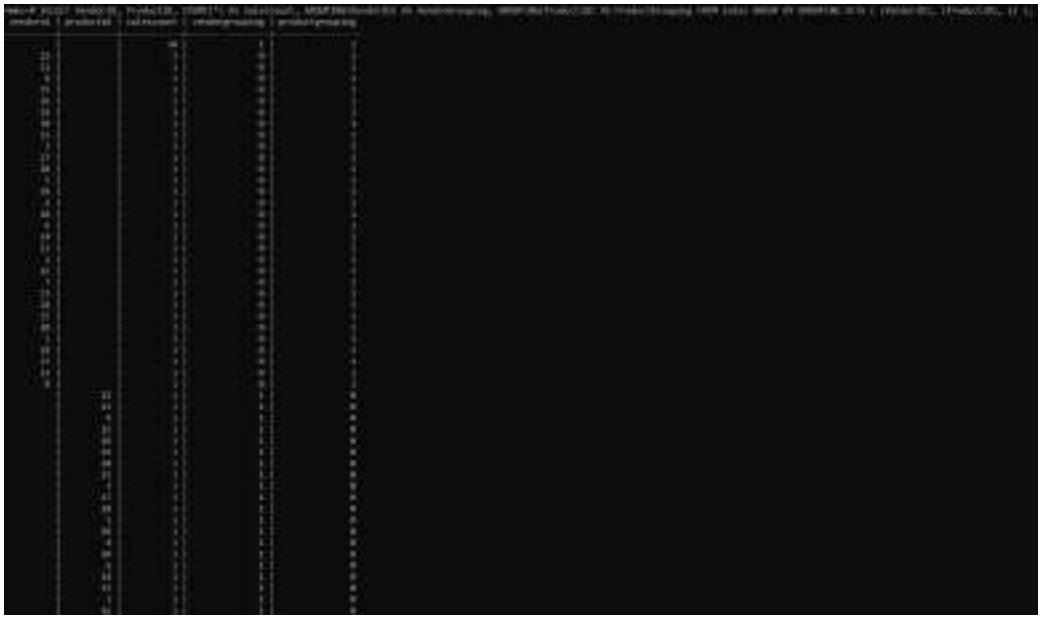
CASE 2: Use GROUPING SETS to calculate the count of rows grouped by a specific column, by a pair of columns, and without grouping.

Description: This query uses `GROUPING SETS` to calculate the count of rows in the `Sales` table grouped by `VendorID`, by `ProductID`, and without grouping (grand total). The `GROUPING` function is used to identify the grouping level.

Scenario: A market manager wants to count the number of sales by vendor, by product, and overall.

Code: *SELECT VendorID, ProductID, COUNT(*) AS SalesCount, GROUPING(VendorID) AS VendorGrouping, GROUPING(ProductID) AS ProductGrouping FROM Sales GROUP BY GROUPING SETS ((VendorID), (ProductID), ());*

Expected Result: The result will show the count of sales for each `VendorID`, each `ProductID`, and the grand total. The `VendorGrouping` and `ProductGrouping` columns will indicate the grouping level.



CASE 3: Use the CUBE operator to calculate all possible totals and subtotals for three columns.

Description: This query uses the `CUBE` operator to calculate all possible totals and subtotals for the `VendorID`, `ProductID`, and `CustomerID` columns in the `Sales` table. The sum of `TotalAmount` is calculated for each combination.
Scenario: A market manager wants to analyze sales totals for all possible combinations of vendors, products, and customers.
Code: SELECT VendorID, ProductID, CustomerID, SUM(TotalAmount) AS TotalSales FROM Sales GROUP BY CUBE (VendorID, ProductID, CustomerID);
Expected Result: The result will show the sum of `TotalAmount` for all possible combinations of `VendorID`, `ProductID`, and `CustomerID`, including subtotals and grand totals.

```
fmms=# SELECT VendorID, ProductID, CustomerID, SUM(TotalAmount) AS TotalSales FROM Sales GROUP BY CUBE (VendorID, ProductID, CustomerID);
 vendorid | productid | customerid | totalsales
----------+-----------+------------+------------
          |           |            |     743.78
       13 |        13 |         13 |       3.52
       28 |        28 |         28 |      21.55
       14 |        14 |         14 |      21.28
       23 |        23 |         23 |      58.02
       15 |        15 |         15 |       8.42
       12 |        12 |         12 |       5.29
        3 |         3 |          3 |      44.50
       10 |        10 |         10 |      33.28
       18 |        18 |         18 |      14.51
       29 |        29 |         29 |      35.06
       27 |        27 |         27 |      11.47
        7 |         7 |          7 |      21.31
        9 |         9 |          9 |      43.35
        1 |         1 |          1 |      47.87
        8 |         8 |          8 |       8.57
        4 |         4 |          4 |      31.09
       17 |        17 |         17 |      12.94
       21 |        21 |         21 |      14.01
       22 |        22 |         22 |      42.25
       16 |        16 |         16 |      38.97
       30 |        30 |         30 |      15.11
       26 |        26 |         26 |      20.75
        6 |         6 |          6 |       6.47
       24 |        24 |         24 |      15.34
       25 |        25 |         25 |       6.58
       30 |        30 |         30 |      43.36
       11 |        11 |         11 |      35.90
        5 |         5 |          5 |      13.33
        2 |         2 |          2 |      18.22
       19 |        19 |         19 |      43.74
```

CASE 4: Use the CUBE operator to calculate the average of a numeric column, grouped by three columns.
Description: This query uses the `CUBE` operator to calculate the average of the `TotalAmount` column in the `Sales` table, grouped by `VendorID`, `ProductID`, and `CustomerID`. All subtotals, including the grand total, are displayed.
Scenario: A market manager wants to analyze the average sales amount for all possible combinations of vendors, products, and customers.
Code: *SELECT VendorID, ProductID, CustomerID, AVG(TotalAmount) AS AverageSales FROM Sales GROUP BY CUBE (VendorID, ProductID, CustomerID);*
Expected Result: The result will show the average `TotalAmount` for all possible combinations of `VendorID`, `ProductID`, and `CustomerID`, including subtotals and grand totals.

```
tms=# SELECT VendorID, ProductID, CustomerID, AVG(TotalAmount) AS AverageSales FROM Sales GROUP BY CUBE (VendorID, ProductID, CustomerID);
 vendorid | productid | customerid |     averagesales
----------+-----------+------------+--------------------
          |           |            | 24.7925666666666667
       13 |        13 |         13 |  3.5200000000000000
       28 |        28 |         28 | 21.5500000000000000
       14 |        14 |         14 | 21.2000000000000000
       23 |        23 |         23 | 50.0200000000000000
       15 |        15 |         15 |  8.4200000000000000
       12 |        12 |         12 |  5.2900000000000000
        3 |         3 |          3 | 44.5000000000000000
       10 |        10 |         10 | 33.2800000000000000
       18 |        18 |         18 | 14.5100000000000000
       29 |        29 |         29 | 35.0600000000000000
       27 |        27 |         27 | 11.4700000000000000
        7 |         7 |          7 | 21.3100000000000000
        9 |         9 |          9 | 43.3500000000000000
        1 |         1 |          1 | 47.8700000000000000
        8 |         8 |          8 |  8.3700000000000000
        4 |         4 |          4 | 31.0000000000000000
       17 |        17 |         17 | 32.9400000000000000
       21 |        21 |         21 | 14.0100000000000000
       22 |        22 |         22 | 42.2500000000000000
       16 |        16 |         16 | 30.9700000000000000
       20 |        20 |         20 | 15.1100000000000000
       26 |        26 |         26 | 24.7500000000000000
        6 |         6 |          6 |  6.4700000000000000
       24 |        24 |         24 | 15.3400000000000000
       25 |        25 |         25 |  6.5800000000000000
       30 |        30 |         30 | 43.3600000000000000
       11 |        11 |         11 | 15.9000000000000000
        5 |         5 |          5 | 13.3300000000000000
        2 |         2 |          2 | 18.2200000000000000
       19 |        19 |         19 | 43.7400000000000000
```

CASE 5: Use the ROLLUP operator to calculate hierarchical totals for two columns.

Description: This query uses the `ROLLUP` operator to calculate hierarchical totals for the `VendorID` and `ProductID` columns in the `Sales` table. The count of rows is calculated for each level of grouping, including the grand total.

Scenario: A market manager wants to analyze the hierarchical sales count by vendor and product.

Code: *SELECT VendorID, ProductID, COUNT(*) AS SalesCount FROM Sales GROUP BY ROLLUP (VendorID, ProductID);*

Expected Result: The result will show the count of sales for each `VendorID`, each combination of `VendorID` and `ProductID`, and the grand total.

```
fmms-# SELECT VendorID, ProductID, COUNT(*) AS SalesCount FROM Sales GROUP BY ROLLUP (VendorID, ProductID
 vendorid | productid | salescount
----------+-----------+------------
          |           |         30
       25 |        25 |          1
        1 |         1 |          1
       27 |        27 |          1
        3 |         3 |          1
       20 |        20 |          1
       14 |        14 |          1
        7 |         7 |          1
       18 |        18 |          1
        2 |         2 |          1
        8 |         8 |          1
       12 |        12 |          1
        5 |         5 |          1
       11 |        11 |          1
       28 |        28 |          1
       23 |        23 |          1
       21 |        21 |          1
        9 |         9 |          1
        6 |         6 |          1
       30 |        30 |          1
       16 |        16 |          1
        4 |         4 |          1
       22 |        22 |          1
       19 |        19 |          1
       10 |        10 |          1
       17 |        17 |          1
       13 |        13 |          1
       29 |        29 |          1
       24 |        24 |          1
       15 |        15 |          1
       26 |        26 |          1
```

CASE 6: Use GROUPING SETS to calculate the count of rows grouped by a specific column, by a pair of columns, and without grouping.

Description: This query uses the `ROLLUP` operator to calculate subtotals and grand totals for the `TotalAmount` and `QuantitySold` columns in the `Sales` table, grouped by `VendorID`.

Scenario: A market manager wants to analyze subtotals and grand totals for sales amounts and quantities sold by vendor.

Code: *SELECT VendorID, SUM(TotalAmount) AS TotalSales, SUM(QuantitySold) AS TotalQuantitySold FROM Sales GROUP BY ROLLUP (VendorID);*

Expected Result: The result will show the sum of `TotalAmount` and `QuantitySold` for each `VendorID`, as well as the grand total.

```
fmms=# SELECT VendorID, SUM(TotalAmount) AS TotalSales, SUM(QuantitySold) AS TotalQuantitySold FROM Sales GROUP BY ROLLUP (VendorID);
 vendorid | totalsales | totalquantitysold
----------+------------+-------------------
          |     783.78 |               224
       22 |      42.25 |                 6
       11 |      35.90 |                 7
        9 |      43.35 |                 5
       15 |       8.42 |                 8
       26 |      24.75 |                14
       19 |      43.74 |                 4
       30 |      43.36 |                 8
       21 |      14.01 |                11
        3 |      44.50 |                 3
       17 |      32.94 |                 8
       28 |      21.55 |                 1
        5 |      13.33 |                 8
       29 |      35.06 |                 2
        4 |      31.09 |                 6
       10 |      33.28 |                15
        6 |       6.47 |                 2
       14 |      21.20 |                 6
       13 |       3.52 |                 3
        2 |      18.22 |                 7
       16 |      30.97 |                 7
        7 |      21.31 |                 6
       12 |       5.29 |                15
       24 |      15.34 |                 1
       25 |       6.58 |                12
       20 |      15.11 |                 3
        1 |      47.87 |                15
       18 |      14.51 |                14
       27 |      11.47 |                13
       23 |      50.02 |                 1
        8 |       8.37 |                13
```

CASE 7: Combine two queries using INTERSECT.

Description: This query combines two queries using the `INTERSECT` operator. The first query selects rows where the `CustomerName` starts with 'A', and the second query selects rows where the `TotalAmount` is less than 50. The result includes rows that meet both conditions.

Scenario: A market manager wants to find customers whose names start with 'A' and who have made purchases with a total amount less than 50.

Code: *SELECT CustomerID, CustomerName, TotalAmount FROM Sales JOIN Customers ON Sales.CustomerID = Customers.CustomerID WHERE CustomerName LIKE 'A%' INTERSECT SELECT CustomerID, CustomerName, TotalAmount FROM Sales JOIN Customers ON Sales.CustomerID = Customers.CustomerID WHERE TotalAmount < 50;*

Expected Result: The result will show the `CustomerID`, `CustomerName`, and `TotalAmount` for customers whose names start with 'A' and who have made purchases with a total amount less than 50.

```
fmms=# SELECT Customers.CustomerID, Customers.CustomerName, Sales.Tot
s.CustomerID, Customers.CustomerName, Sales.TotalAmount FROM Sales JO
 customerid |      customername      | totalamount
------------+-----------------------+-------------
         16 | Addy Karlsson         |       30.97
         24 | Adelheid Fridlington  |       15.34
         19 | Austen Madison        |       43.74
          3 | Auria Bever           |       44.50
         18 | Adoree Saunders       |       14.51
          7 | Amye Wiffill          |       21.31
```

VII. References

1. *Vertabelo - Design your database online*. (n.d.). Vertabelo - Design Your Database Online. https://my.vertabelo.com/model/y7PifPqTWVEuQMV9RKUUBenqhYTx51wt#