**E-Commerce Nexus - Uniting Customers and Products through Data**

Group 2

Team Coordinator: Niharika Madhadi

Team Members: Niharika Madhadi

Pranavi Gantla

Navya Manyamgiri

Akhila Bollapalli

Instructor: Mr. Tozammel Hossain

Data Modeling for Information Professionals INFO 5707

Term Project

**Description**

The usage of e-commerce websites has been rapidly increasing these days. Lot of data is being generated in e-commerce, so it has become crucial to manage this data. Efficient management of data is crucial for the success of e-commerce platforms. This project will create a sophisticated database to unite customers and products through data, ensuring smooth transactions and a positive user experience.

This database will have several entities like Customers, Products, Orders, Reviews, Payments, and Shipping. These entities are related to each other by one to one, one to many or many to many relationships. The system will provide robust data storage, retrieval, and manipulation capabilities to support day-to-day e-commerce activities while ensuring data integrity, security, and scalability.

**Objectives**  
  
The major goal of the E-Commerce Nexus database system is to handle and analyse the massive volume of data created during e-commerce activities. The database's goal is to bring customers and items together through data, simplifying easy transactions and improving user experience on e-commerce platforms. The major goals include:

**Efficient Data Management:** The database system is intended to store, retrieve, and manage a variety of e-commerce data, such as customer information, product details, orders, reviews, payments, shipping records, and sales data from many channels.

**Accurate Information:** Ensure that product data, including descriptions, prices, and quantities, is consistently updated and reliable, minimizing errors and discrepancies.  
  
**Data research and Insights:** The database system allows for in-depth research of e-commerce sales patterns, profitability, pricing strategies, and customer behaviour. It enables retailers and digital marketers to gain useful insights for increasing sales performance and improving business strategy by delivering strong data analysis capabilities.

**Data integrity and security**: These are essential objectives of the database system. It uses processes to ensure the accuracy, consistency, and dependability of e-commerce data, as well as security measures to protect sensitive information from unwanted access or breaches.   
  
**Optimized Inventory Management**: Enhancing control over stock levels, recognizing slow- and fast-moving products and forecasting shift in demand trends.

**Enhanced Customer Satisfaction:** Refine the customer experience through the maintenance of precise product details and availability of items, thereby elevating satisfaction levels.

**Scope**  
  
The E-Commerce Nexus database's scope includes a variety of entities, datasets, and functions required for effective e-commerce management and analytics. The main elements of the database scope are:

**Entities:** To represent the basic parts of e-commerce transactions and interactions, the database contains entities such as Customers, Products, Orders, Reviews, Payments, Shipping, and Financial Data (e.g., costs, profits).

**Database Schema:** All the entities, its primary keys, foreign keys, and relationships and connectivity with other entities are defined in ERD and Data Dictionary.

**Database Normalization:** This process involves organizing the database tables and their relationships to minimize duplication of data and to prevent anomalies such as insertion, deletion, and update anomalies.

**Functionality:** The database system provides a variety of functions to support e-commerce activities such as data storage, retrieval, analysis, and reporting. Users can analyse sales trends, compare profitability across many channels, evaluate pricing strategies, track inventory levels, and optimize expenses.

**Analytical Capabilities:** The database supports advanced analytics and reporting to generate meaningful insights from e-commerce data. Users can undertake detailed assessments of sales performance, customer behaviour, product profitability, and market trends to help with strategic decision-making and business optimization.

**Security and Compliance:** The database uses strong security measures to secure sensitive e-commerce data while also adhering to privacy rules and industry norms. To ensure the integrity and confidentiality of data, access controls, encryption, and auditing measures are used.

To summarize, the E-Commerce Nexus database system promises to provide a comprehensive platform for monitoring, evaluating, and optimizing e-commerce operations by connecting customers and items through data. It has vast capabilities for handling varied datasets, generating important insights, and assisting with informed decision-making in the dynamic and competitive e-commerce industry.

**User Requirements**

1. **User Information Management:** Businesses should be able to retrieve the user information like their name, age, contact, address, and the sales made by them. Also, they should be able to edit them without any update anomalies.
2. **Product Management:** Admins should be able to add, edit and delete products. These products have attributes like name, description, price, quantity in stock, offers available that will help user in their buying decision.
3. **Order Processing:** Businesses should be able to view and manage orders that tells them about the order information like Order ID, placed date, price, customer and product information for that order. This will help them to invest in products that are ordered more frequently.
4. **Search and Filtering:** To efficiently search the products and get the accurate information, and to be able to filter products based on consumer preferences. This allows the users to browse and search for products based on various criteria such as category, price range and brand.
5. **Inventory Management:** The database should track inventory levels and let business know if any product falls below a certain threshold. This will manage inventory levels and update product availability in real time to prevent overselling.
6. **Order Processing:** Admins should be able to update order status, mark them as shipped and track the order with help of Shipping entity that has its attributes like location, expected arrival date.
7. **Reporting and analytics:** Product owners should be able to access and view the product analytics like sales performance, popular products, customer demographics and the reviews written by customer which will stored in Reviews entity. These insights derived will help them in making better decisions.
8. **Payment:** The database stores all financial transactions, including details such as Payment ID, Customer ID, payment amount, and payment date. This enables the payments to be processed securely and effectively.
9. **Security Measures:** To implement security measures to protect sensitive user information and payment data. This data should be encrypted to ensure privacy and should be protected against SQL injection.
10. **Scalability and Performance:** The system needs to efficiently manage a significant volume of simultaneous users and transactions. It should incorporate performance enhancements like indexing and query optimization to guarantee swift response times.
11. **User Friendly Interface:** Allow users to add items to a shopping cart, modify quantities, and proceed to checkout seamlessly, providing a seamless shopping experience for customers and efficient management tools for admins.

By meeting these user needs, the E-Commerce Nexus database system will enable users to manage and optimize e-commerce operations, generate revenue development, and improve the entire consumer experience.

**Business Rules**

**Products**

* Each product is distinguished by a unique ProductID.
* All products must be categorized under at least one product category defined in the ProductCategories entity.
* Essential product details include its name, description, and cost per unit.

**Inventory**

* Inventory entries are each marked with a unique InventoryID.
* Every inventory entry links back to a specific product.
* The inventory count for any item should never fall below zero.

**Orders**

* Orders are each identified by a unique OrderID.
* Every order is connected to a customer listed in the Customers entity.
* Orders must adhere to specified status options (e.g., Pending, Completed, Cancelled).

**OrderDetails**

* Each item within an order is given a unique OrderDetailID.
* These items are linked to both an order and a product.
* The quantity ordered for any item must be at least one.

**Customers**

* Customers are identified by a unique CustomerID.
* At a minimum, customer records must include a name and contact details.
* While customers can have multiple orders, each order is associated with only one customer.

**Payment**

* Each financial transaction is given a unique PaymentID.
* Transactions are associated with specific orders and the customers who placed them.
* Details such as the transaction amount, payment method, and status are mandatory for each transaction.

**ProductCategories**

* Each product category is given a unique CategoryID
* Each product category must have at least one product in it and can have many products as well.

**Reviews**

* It has unique ReviewID column that act as primary key.
* Each review corresponds to customer and product from Customers and Products entities.
* It will have rating, comment and date as other details given in the entity.

**Shipping**

* Each item is uniquely identifiable using the ShippingID column
* It refers to Orders entity and each order can have more than one ShippingID, but each ShippingID corresponds to a single Order.
* ShippedDate, ShippingMethod and TrackingNumber must be given mandatorily for each entry.

The E-Commerce Nexus database system ensures successful e-commerce administration, integrity, security, and compliance by adhering to certain business rules; this promotes confidence, reliability, and success in the online marketplace.

**Entity Relationship Diagram (ERD)**

A diagram of a computer program

Description automatically generated with medium confidence

The below ERD tells about the entities, their relationship, connectivity with each other.

**The Database has below 9 entities. They are:**

Customers

Products

ProductCategory

Orders

OrderDetails

Inventory

Reviews

Payments

Shipping

**Primary keys are depicted as PK and foreign keys are depicted as FK. They are:**

Customers – CustomerID is primary key

Products – ProductID is primary key

CategoryID is foreign key

ProductCategory – CategoryID is primary key

Orders – OrderID is primary key

CustomerID is foreign key

OrderDetails – OrderDetailID is primary key

OrderID is foreign key

ProductID is foreign key

Inventory – InventoryID is primary key

ProductID is foreign key

Reviews – ReviewID is primary key

CustomerID is foreign key

ProductID is foreign key

Payments – PaymentID is primary key

OrderID is foreign key

CustomerID is foreign key

Shipping – ShippingID is primary key

OrderID is foreign key

**The relationships and their cardinalities are as below:**

* **Customers** can place one or many **Orders** (one-to-many).
* Each **Order** is placed by one **Customer** (many-to-one).
* Each **Order** can have one or many **OrderDetails** (one-to-many).
* Each **OrderDetail** belongs to one **Order** (many-to-one).
* Each **Order** can have one **Shipment** (one-to-one).
* Each **Shipment** is associated with one **Order** (one-to-one).
* Each **Product** can be a part of one or many **Orders** (one-to-many) through the **OrderDetails** entity.
* Each **OrderDetail** is associated with one **Product** (many-to-one).
* Each **Product** belongs to one **ProductCategory** (one-to-many).
* Each **ProductCategory** can have many **Products** (one-to-many).
* Each **Product** can have one or many **Reviews** (one-to-many).
* Each **Review** is associated with one **Product** (many-to-one).
* Each **Order** can have one or many **Payments** (one-to-many).
* Each **Payment** is associated with one **Order** (many-to-one).
* Each **Customer** can make one or many **Payments** (one-to-many).
* Each **Payment** is made by one **Customer** (many-to-one).
* Each **Inventory** record is for one **Product** (one-to-one).
* Each **Product** has one **Inventory** record (one-to-one).

**Data Dictionary**

A data dictionary is a detailed catalogue of data elements in a database or system, providing information on the data type, format, and constraints of each element. It typically also includes insights into how the data is utilized and any connections or relationships among the data elements.

Key Roles of a Data Dictionary:

* Documentation: Serves as a comprehensive reference for understanding the structure, relationships, and usage of data in a database.
* Data Integrity and Quality: Ensures consistency and accuracy in data handling by defining data types, formats, and constraints.
* Compliance and Governance: Facilitates adherence to data standards and regulatory requirements by providing clear definitions and data lineage.

Data Dictionary:

A screenshot of a document

Description automatically generated

A close-up of a document

Description automatically generated

A screenshot of a document

Description automatically generated

**Entity Generation and Data Entry**

MySQL is our choice of DBMS. All the below queries are given in MySQL Workbench.

Initially, create a database where we can store all the entities related to the project.

Command: **CREATE DATABASE Project;**

A screenshot of a computer

Description automatically generated

We can check if our database is created or not, using **SHOW DATABASES** command.

A screenshot of a computer

Description automatically generated

Now we can enter tables or entities into this database.

**ProductCategory Table**

**Create Table Statement**

**CREATE TABLE ProductCategory (**

**CategoryID INT PRIMARY KEY,**

**Name VARCHAR(25)**

**);**

****

**Insert Statement**

**INSERT INTO ProductCategory (CategoryID, Name) VALUES**

(1,'Electronics'),

(2,'Clothing'),

(3,'Books'),

(4,'Home Appliances'),

(5,'Furniture'),

(6,'Toys'),

(7,'Sporting goods'),

(8,'Beauty'),

(9,'Grocery'),

(10,'Health\_Wellness');

A white background with blue text

Description automatically generated

**Result**

**SELECT \* FROM ProductCategory;**

**A screenshot of a computer

Description automatically generated**

**Description of ProductCategory table:**

CategoryID (Primary Key, INT): This column serves as the unique identifier for each category in the table. It is an integer value that auto-increments for each new category added.

Name (VARCHAR): This column stores the name of each category. It is a variable-length string that describes the category of products.

This table is designed to store various product categories. Each category is uniquely identified by a CategoryID, and its name is stored in the Name column. This table serves as a reference for categorizing products in an e-commerce platform, inventory management system, or any other application where categorization of items is required.

**Products Table**

**Create Table Statement**

**CREATE TABLE Products (**

**ProductID INT PRIMARY KEY,**

**ProductName VARCHAR(25),**

**Description VARCHAR(25),**

**Price DECIMAL(10, 2),**

**CategoryID INT,**

**FOREIGN KEY (CategoryID) REFERENCES ProductCategory(CategoryID)**

**);**

**A close-up of a computer screen

Description automatically generated**

**Insert Statement**

**INSERT INTO Products VALUES**

(101,'SmartPhone','Electronic Item',599.99,1),

(102,'Laptop','Electronic Item',999.99,1),

(103,'Earphones','Electronic Item',149.99,1),

(104,'T Shirt','Clothing Item',19.99,2),

(105,'Denim Jeans','Clothing Item',49.99,2),

(106,'Fiction Book','Fun to Read',12.99,3),

(107,'Non-fiction','Informative',15.99,3),

(108,'Tablet','Electronic Item',399.99,1),

(109,'Dress','Clothing Item',39.99,2),

(110,'Biography','Motivational book',29.99,3),

(111,'Microwave','For heating',129.99,4),

(112,'Coffee Maker','To prepare coffee',79.99,4),

(113,'Sofa','Furniture Item',599.99,5),

(114,'Dining Table','Furniture Item',299.99,5),

(115,'Action Figure','Toy',9.99,6),

(116,'Puzzle game','Game',29.99,6),

(117,'Soccer ball','Play soccer',19.99,7),

(118,'Yoga Mat','For Yoga',29.99,7),

(119,'Mascara','Lifts lashes',9.99,8),

(120,'Shampoo','Hair wash',18.99,8),

(121,'Cereal','Breakfast',5.99,9),

(122,'Pasta','Italian',9.99,9),

(123,'Vitamin Supplement','For health',19.99,10),

(124,'Herbal Tea','For weight loss',4.99,10),

(125,'Vacuum Cleaner','Removes dust',199.99,4),

(126,'Desk Chair','Office or study',129.99,5),

(127,'Board game','Fun',24.99,6),

(128,'Tennis Racket','Play tennis',59.99,7),

(129,'Lipstick','Color lips',12.99,8),

(130,'Protein bar','Energy',1.99,9);

A white background with text

Description automatically generated

A white background with numbers and letters

Description automatically generated with medium confidence

**Result**

**SELECT \* FROM Products;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description of the Product table**

ProductID (Primary Key, INT): This column serves as the unique identifier for each product in the table. It is an integer value that auto-increments for each new product added.

CategoryID (Foreign Key, INT): This column represents the category to which the product belongs. It is linked to the CategoryID column in the ProductCategory table, establishing a relationship between products and categories.

ProductName (VARCHAR): This column stores the name or title of each product. It is a variable-length string.

Description (VARCHAR): This column provides a description of the product.

Price (DECIMAL): This column stores the price of each product. It is a decimal value representing the cost of the product.

The Products table is designed to store comprehensive information about various products. Each product is uniquely identified by a ProductID, and its association with a specific category is established through the CategoryID column. ProductName provides a concise title for the product, while Description offers more detailed information. Finally, Price indicates the cost of each product, allowing for easy pricing and financial management within the system.

**Inventory Table**

**Create Table Statement**

**CREATE TABLE Inventory(**

**InventoryID INT PRIMARY KEY,**

**ProductID INT,**

**QuantityAvailable INT,**

**RecordedLevel DATE,**

**FOREIGN KEY (ProductID) REFERENCES Products(ProductID)**

**);**

**A close-up of a white background

Description automatically generated**

**Insert Statement**

**INSERT INTO Inventory VALUES**

(201,103,50,'2024-03-21'),

(202,125,30,'2024-04-24'),

(203,114,100,'2024-04-22'),

(204,109,20,'2024-03-26'),

(205,117,15,'2024-04-19'),

(206,122,40,'2024-04-05'),

(207,106,25,'2024-03-29'),

(208,129,60,'2024-04-12'),

(209,119,10,'2024-04-12'),

(210,105,5,'2024-04-19');

A white background with black dots

Description automatically generated

**Result**

**SELECT \* FROM Inventory;**

**A screenshot of a computer

Description automatically generated**

**Description of Inventory table**

InventoryID (Primary Key, INT): This column serves as the unique identifier for each inventory record in the table. It is an integer value that auto-increments for each new inventory entry.

ProductID (Foreign Key, INT): This column references the ProductID from the Products table, establishing a relationship between inventory records and specific products.

QuantityAvailable (INT): This column stores the quantity of the product available in the inventory. It represents the number of units of the product that are currently in stock and available for sale.

RecordedDate (DATE): This column stores the date when the inventory level was last recorded or updated. It represents the date without the time component.

The Inventory table is designed to track the stock levels of various products. Each inventory record is uniquely identified by an InventoryID, and its association with a specific product is established through the ProductID column. The QuantityAvailable column indicates how many units of the product are currently in stock. RecordedDate stores the date when the inventory level was last updated, facilitating inventory management and tracking of stock changes over time.

**Customers Table**

**Create Table Statement**

**CREATE TABLE Customers(**

**CustomerID INT PRIMARY KEY,**

**FirstName VARCHAR(25),**

**LastName VARCHAR(25),**

**Address VARCHAR(50),**

**ContactNumber INT,**

**EmailAddress VARCHAR(30)**

**);**

**A white background with black and white clouds

Description automatically generated**

**Insert Statement**

**INSERT INTO Customers (CustomerID, FirstName, LastName, Address, ContactNumber, EmailAddress) VALUES**

(301, 'Aarav', 'Agarwal', '123 Main St, Cityville', 1234567890, 'agarwal.aarav@site.com'),

(302, 'Aarushi', 'Bhatnagar', '456 Elm St, Townsville', 887654321, 'bhatnagar.aarushi@site.com'),

(303, 'Abhay', 'Chauhan', '789 Oak St, Villageton', 555123456, 'chauhan.abhay@site.com'),

(304, 'Aditi', 'Desai', '321 Maple Ave, Hamletown', 333444555, 'desai.aditi@site.com'),

(305, 'Akash', 'Gupta', '567 Pine St, Townsville', 222333444, 'gupta.aakash@site.com'),

(306, 'Akshay', 'Joshi', '876 Cedar St, Villageton', 666778888, 'joshi.akshay@site.com'),

(307, 'Aman', 'Khanna', '543 Birch St, Hamletown', 999888777, 'khanna.aman@site.com'),

(308, 'Ananya', 'Kumar', '987 Oak St, Cityville', 444555666, 'kumar.ananya@site.com'),

(309, 'Ankit', 'Malhotra', '456 Maple Ave, Townsville',776665555, 'malhotra.ankit@site.com'),

(310, 'Anushka', 'Mehra', '321 Pine St, Villageton', 112223333, 'mehra.anushka@site.com'),

(311, 'Arjun', 'Mishra', '654 Cedar St, Hamletown', 889990000, 'mishra.arjun@site.com'),

(312, 'Ayush', 'Patel', '123 Birch St, Cityville', 221110000, 'patel.ayush@site.com'),

(313, 'Bhavya', 'Rao', '987 Maple Ave, Townsville', 778889999, 'rao.bhavya@site.com'),

(314, 'Devika', 'Reddy', '654 Pine St, Villageton', 334445555, 'reddy.devika@site.com'),

(315, 'Divya', 'Saxena', '789 Cedar St, Hamletown', 556667777, 'saxena.divya@site.com'),

(316, 'Gaurav', 'Sharma', '987 Birch St, Cityville', 990001111, 'sharma.gaurav@site.com'),

(317, 'Ishika', 'Singh', '123 Maple Ave, Townsville', 665554444, 'singh.ishika@site.com'),

(318, 'Kavya', 'Srivastava', '876 Pine St, Villageton', 222334444, 'srivastava.kavya@site.com'),

(319, 'Manish', 'Thakur', '456 Cedar St, Hamletown', 888990000, 'thakur.manish@site.com'),

(320, 'Meera', 'Verma', '654 Birch St, Cityville', 333222111, 'verma.meera@site.com'),

(321, 'Mohan', 'Kapoor', '789 Maple Ave, Townsville', 778889999, 'kapoor.mohan@site.com'),

(322, 'Neha', 'Shah', '876 Pine St, Villageton', 555443333, 'shah.neha@site.com'),

(323, 'Nisha', 'Pandey', '123 Cedar St, Hamletown', 111223333, 'pandey.nisha@site.com'),

(324, 'Pranav', 'Reddy', '987 Birch St, Cityville', 667778888, 'reddy.pranav@site.com'),

(325, 'Priya', 'Chowdary', '654 Maple Ave, Townsville', 990001111, 'chowdary.priya@site.com'),

(326, 'Rahul', 'Rao', '321 Pine St, Villageton', 444555666, 'rao.rahul@site.com'),

(327, 'Raj', 'Patel', '456 Cedar St, Hamletown', 888999000, 'patel.raj@site.com'),

(328, 'Rishi', 'Gupta', '789 Birch St, Cityville', 333445555, 'gupta.rishi@site.com'),

(329, 'Riya', 'Singh', '876 Maple Ave, Townsville', 778889999, 'singh.riya@site.com'),

(330, 'Snigdha', 'Patel', '123 Pine St, Villageton', 554443333, 'patel.snigdha@site.com');

A close up of a computer screen

Description automatically generated

A close up of text

Description automatically generated

**Result**

**SELECT \* FROM Customers;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description of Customers table**

CustomerID (Primary Key, INT): This column serves as the unique identifier for each customer in the table. It is an integer value that auto-increments for each new customer added.

FirstName (VARCHAR): This column stores the first name of each customer. It is a variable-length string that represents the customer's first name.

LastName (VARCHAR): This column stores the last name of each customer. It is a variable-length string that represents the customer's last name.

Address (VARCHAR): This column stores the address of each customer. It is a variable-length string that represents the customer's physical address.

ContactNumber (INT): This column stores the contact number of each customer. It is an integer value that represents the customer's phone number.

EmailAddress (VARCHAR): This column stores the email address of each customer. It is a variable-length string that represents the customer's email address.

The Customers table is designed to store information about individual customers. Each customer is uniquely identified by a CustomerID, and their personal details such as first name, last name, address, contact number, and email address are stored in the respective columns. This table facilitates customer management and enables businesses to maintain a record of their customers' information for communication and marketing purposes.

**Orders Table**

**Create Table Statement**

**CREATE TABLE Orders(**

**OrderID INT PRIMARY KEY,**

**CustomerID INT,**

**OrderDate DATE,**

**TotalAmount DECIMAL(10, 2),**

**Status VARCHAR(20),**

**ShippingAddress VARCHAR(100),**

**FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));**

**A white background with colorful text

Description automatically generated**

**Insert Statement**

**INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status, ShippingAddress) VALUES**

(401,305, '2024-04-24', 99.99, 'Pending', '123 Main St, Cityville'),

(402,312, '2024-04-23', 199.99, 'Shipped', '456 Elm St, Townsville'),

(403,324, '2024-04-22', 299.99, 'Delivered', '789 Oak St, Villageton'),

(404,309, '2024-04-21', 399.99, 'Pending', '321 Maple Ave, Hamletown'),

(405,302, '2024-04-20', 499.99, 'Shipped', '567 Pine St, Townsville'),

(406,312, '2024-04-19', 599.99, 'Delivered', '876 Cedar St, Villageton'),

(407,317, '2024-04-18', 699.99, 'Pending', '543 Birch St, Hamletown'),

(408,315, '2024-04-17', 799.99, 'Shipped', '987 Oak St, Cityville'),

(409,326, '2024-04-16', 899.99, 'Delivered', '456 Maple Ave, Townsville'),

(410,302, '2024-04-15', 999.99, 'Pending', '321 Pine St, Villageton'),

(411,330, '2024-04-14', 1099.99, 'Shipped', '654 Cedar St, Hamletown'),

(412,325, '2024-04-13', 1199.99, 'Delivered', '123 Birch St, Cityville'),

(413,315, '2024-04-12', 1299.99, 'Pending', '987 Maple Ave, Townsville'),

(414,312, '2024-04-11', 1399.99, 'Shipped', '654 Pine St, Villageton'),

(415,308, '2024-04-10', 1499.99, 'Delivered', '789 Cedar St, Hamletown'),

(416,304, '2024-04-09', 1599.99, 'Pending', '987 Birch St, Cityville'),

(417,326, '2024-04-08', 1699.99, 'Shipped', '123 Maple Ave, Townsville'),

(418,305, '2024-04-07', 1799.99, 'Delivered', '876 Pine St, Villageton'),

(419,309, '2024-04-06', 1899.99, 'Pending', '456 Cedar St, Hamletown'),

(420,301, '2024-04-05', 1999.99, 'Shipped', '654 Birch St, Cityville'),

(421,330, '2024-04-04', 2099.99, 'Delivered', '789 Maple Ave, Townsville'),

(422,320, '2024-04-03', 2199.99, 'Pending', '876 Pine St, Villageton'),

(423,303, '2024-04-02', 2299.99, 'Shipped', '123 Cedar St, Hamletown'),

(424,324, '2024-04-01', 2399.99, 'Delivered', '987 Birch St, Cityville'),

(425,316, '2024-03-31', 2499.99, 'Pending', '654 Maple Ave, Townsville'),

(426,319, '2024-03-30', 2599.99, 'Shipped', '321 Pine St, Villageton'),

(427,302, '2024-03-29', 2699.99, 'Delivered', '456 Cedar St, Hamletown'),

(428,308, '2024-03-28', 2799.99, 'Pending', '789 Birch St, Cityville'),

(429,317, '2024-03-27', 2899.99, 'Shipped', '876 Maple Ave, Townsville'),

(430,330, '2024-03-26', 2999.99, 'Delivered', '123 Pine St, Villageton');

A screen shot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

**Result**

**SELECT \* FROM Orders;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description of Orders table**

OrderID (Primary Key, INT): This column serves as the unique identifier for each order in the table. It is an integer value that uniquely identifies each order.

CustomerID (Foreign Key, INT): This column references the CustomerID from the Customers table, establishing a relationship between orders and customers. It indicates which customer placed the order.

OrderDate (DATE): This column stores the date when the order was placed. It represents the date without the time component.

TotalAmount (DECIMAL): This column stores the total amount of the order. It is a decimal value representing the total cost of the items in the order.

Status (VARCHAR): This column stores the status of the order, such as "Pending", "Shipped", or "Delivered". It indicates the current status of the order in the order processing workflow.

ShippingAddress (VARCHAR): This column stores the shipping address for the order. It is a variable-length string that represents the address where the order will be delivered.

The Orders table is designed to store information about individual orders placed by customers. Each order is uniquely identified by an OrderID, and its association with a specific customer is established through the CustomerID column. Other details such as the order date, total amount, status, and shipping address are also stored for each order, facilitating order management and tracking.

**OrderDetails Table**

**Create Table Statement**

**CREATE TABLE OrderDetails(**

**OrderDetailID INT PRIMARY KEY,**

**OrderID INT,**

**ProductID INT,**

**Quantity INT,**

**UnitPrice INT,**

**FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),**

**FOREIGN KEY (ProductID) REFERENCES Products(ProductID)**

**);**

**A close-up of a computer screen

Description automatically generated**

**Insert Statement**

**INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity, UnitPrice) VALUES**

(501, 401, 101, 3, 19.99),

(502, 402, 102, 2, 29.99),

(503, 403, 103, 1, 39.99),

(504, 404, 104, 4, 9.99),

(505, 405, 105, 2, 49.99),

(506, 406, 106, 1, 59.99),

(507, 407, 107, 3, 14.99),

(508, 408, 108, 2, 24.99),

(509, 409, 109, 1, 34.99),

(510, 410, 110, 5, 7.99),

(511, 411, 111, 2, 44.99),

(512, 412, 112, 1, 54.99),

(513, 413, 113, 3, 12.99),

(514, 414, 114, 2, 27.99),

(515, 415, 115, 1, 37.99),

(516, 416, 116, 4, 8.99),

(517, 417, 117, 2, 47.99),

(518, 418, 118, 1, 57.99),

(519, 419, 119, 3, 16.99),

(520, 420, 120, 2, 22.99),

(521, 421, 121, 1, 32.99),

(522, 422, 122, 5, 6.99),

(523, 423, 123, 2, 42.99),

(524, 424, 124, 1, 52.99),

(525, 425, 125, 3, 11.99),

(526, 426, 126, 2, 28.99),

(527, 427, 127, 1, 38.99),

(528, 428, 128, 4, 7.49),

(529, 429, 129, 2, 46.99),

(530, 430, 130, 1, 56.99);

A screenshot of a computer

Description automatically generated

A white background with black dots

Description automatically generated

**Result**

**SELECT \* FROM OrderDetails;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description of OrderDetails table**

OrderDetailID (Primary Key, INT): This column serves as the unique identifier for each order detail in the table. It is an integer value that uniquely identifies each order detail.

OrderID (Foreign Key, INT): This column references the OrderID from the Orders table, establishing a relationship between order details and orders. It indicates which order the order detail belongs to.

ProductID (Foreign Key, INT): This column references the ProductID from the Products table, establishing a relationship between order details and products. It indicates which product is included in the order detail.

Quantity (INT): This column stores the quantity of the product included in the order detail.

UnitPrice (DECIMAL): This column stores the unit price of the product included in the order detail.

The OrderDetails table is designed to store information about individual order details, representing the products included in each order. Each order detail is uniquely identified by an OrderDetailID. It includes references to the order it belongs to (OrderID) and the product it represents (ProductID), along with the quantity and unit price of the product in the order detail. This table facilitates order management by providing detailed information about the products included in each order.

**Reviews Table**

**Create Table Statement**

**CREATE TABLE Reviews(**

**ReviewID INT PRIMARY KEY,**

**CustomerID INT,**

**ProductID INT,**

**Rating INT,**

**Comment VARCHAR(255),**

**PostedDate DATE,**

**FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),**

**FOREIGN KEY (ProductID) REFERENCES Products(ProductID));**

**A computer screen shot

Description automatically generated**

**Insert Statement**

**INSERT INTO Reviews (ReviewID, CustomerID, ProductID, Rating, Comment, PostedDate) VALUES**

(1, 301, 101, 4, 'Great product, very satisfied!', '2024-04-24'),

(2, 302, 102, 5, 'Excellent quality, highly recommended.', '2024-04-23'),

(3, 303, 103, 3, 'Average product, could be better.', '2024-04-22'),

(4, 304, 104, 2, 'Disappointed with the product.', '2024-04-21'),

(5, 305, 105, 5, 'Absolutely love it!', '2024-04-20'),

(6, 306, 106, 4, 'Good product, worth the price.', '2024-04-19'),

(7, 307, 107, 3, 'Okay product, not as expected.', '2024-04-18'),

(8, 308, 108, 4, 'Satisfied with the purchase.', '2024-04-17'),

(9, 309, 109, 5, 'Impressive product, exceeded expectations.', '2024-04-16'),

(10, 310, 110, 2, 'Poor quality, not recommended.', '2024-04-15');

A screenshot of a computer

Description automatically generated

**Result**

**SELECT \* FROM Reviews;**

**A screenshot of a computer

Description automatically generated**

**Description of Reviews table**

ReviewID (Primary Key, INT): This column serves as the unique identifier for each review in the table. It is an integer value that uniquely identifies each review.

CustomerID (Foreign Key, INT): This column references the CustomerID from the Customers table, establishing a relationship between reviews and customers. It indicates which customer posted the review.

ProductID (Foreign Key, INT): This column references the ProductID from the Products table, establishing a relationship between reviews and products. It indicates which product the review is for.

Rating (INT): This column stores the rating given by the customer for the product. It is an integer value typically ranging from 1 to 5, where 1 represents the lowest rating and 5 represents the highest rating.

Comment (VARCHAR): This column stores the comment provided by the customer in the review. It is a variable-length string that contains additional feedback or comments about the product.

PostedDate (DATE): This column stores the date when the review was posted. It represents the date without the time component.

The Reviews table is designed to store information about customer reviews for products. Each review is uniquely identified by a ReviewID and includes details such as the customer who posted the review, the product being reviewed, the rating given, the comment provided, and the date when the review was posted. This table facilitates feedback collection and analysis for products, helping businesses understand customer satisfaction and product performance.

**Shipping Table**

**Create Table Statement**

**CREATE TABLE Shipping (**

**ShippingID INT PRIMARY KEY,**

**OrderID INT,**

**TrackingNumber VARCHAR(50),**

**ShippedDate DATE,**

**ShippingMethod VARCHAR(100),**

**ExpectedDeliveryDate DATE,**

**FOREIGN KEY (OrderID) REFERENCES Orders(OrderID));**

**A white background with black and white clouds

Description automatically generated**

**Insert Statement**

**INSERT INTO Shipping (ShippingID, OrderID, TrackingNumber, ShippedDate, ShippingMethod, ExpectedDeliveryDate) VALUES**

(1, 401, 'TN123456', '2024-04-24', 'Standard Shipping', '2024-05-01'),

(2, 402, 'TN234567', '2024-04-23', 'Express Shipping', '2024-04-28'),

(3, 403, 'TN345678', '2024-04-22', 'Standard Shipping', '2024-04-29'),

(4, 404, 'TN456789', '2024-04-21', 'Standard Shipping', '2024-04-28'),

(5, 405, 'TN567890', '2024-04-20', 'Express Shipping', '2024-04-25'),

(6, 406, 'TN678901', '2024-04-19', 'Standard Shipping', '2024-04-26'),

(7, 407, 'TN789012', '2024-04-18', 'Standard Shipping', '2024-04-27'),

(8, 408, 'TN890123', '2024-04-17', 'Express Shipping', '2024-04-24'),

(9, 409, 'TN901234', '2024-04-16', 'Standard Shipping', '2024-04-23'),

(10, 410, 'TN012345', '2024-04-15', 'Standard Shipping', '2024-04-22'),

(11, 411, 'TN123456', '2024-04-14', 'Express Shipping', '2024-04-21'),

(12, 412, 'TN234567', '2024-04-13', 'Standard Shipping', '2024-04-20'),

(13, 413, 'TN345678', '2024-04-12', 'Standard Shipping', '2024-04-19'),

(14, 414, 'TN456789', '2024-04-11', 'Express Shipping', '2024-04-18'),

(15, 415, 'TN567890', '2024-04-10', 'Standard Shipping', '2024-04-17'),

(16, 416, 'TN678901', '2024-04-09', 'Standard Shipping', '2024-04-16'),

(17, 417, 'TN789012', '2024-04-08', 'Express Shipping', '2024-04-15'),

(18, 418, 'TN890123', '2024-04-07', 'Standard Shipping', '2024-04-14'),

(19, 419, 'TN901234', '2024-04-06', 'Standard Shipping', '2024-04-13'),

(20, 420, 'TN012345', '2024-04-05', 'Express Shipping', '2024-04-12'),

(21, 421, 'TN123456', '2024-04-04', 'Standard Shipping', '2024-04-11'),

(22, 422, 'TN234567', '2024-04-03', 'Standard Shipping', '2024-04-10'),

(23, 423, 'TN345678', '2024-04-02', 'Express Shipping', '2024-04-09'),

(24, 424, 'TN456789', '2024-04-01', 'Standard Shipping', '2024-04-08'),

(25, 425, 'TN567890', '2024-03-31', 'Standard Shipping', '2024-04-07'),

(26, 426, 'TN678901', '2024-03-30', 'Express Shipping', '2024-04-06'),

(27, 427, 'TN789012', '2024-03-29', 'Standard Shipping', '2024-04-05'),

(28, 428, 'TN890123', '2024-03-28', 'Standard Shipping', '2024-04-04'),

(29, 429, 'TN901234', '2024-03-27', 'Express Shipping', '2024-04-03'),

(30, 430, 'TN012345', '2024-03-26', 'Standard Shipping', '2024-04-02');

A screen shot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

**Result**

**SELECT \* FROM Shipping;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description of Shipping table**

ShippingID (Primary Key, INT): This column serves as the unique identifier for each shipping record in the table. It is an integer value that uniquely identifies each shipment.

OrderID (Foreign Key, INT): This column references the OrderID from the Orders table, establishing a relationship between shipments and orders. It indicates which order the shipment belongs to.

TrackingNumber (VARCHAR): This column stores the tracking number associated with the shipment. It is a variable-length string that uniquely identifies the shipment for tracking purposes.

ShippedDate (DATE): This column stores the date when the shipment was sent out. It represents the date without the time component.

ShippingMethod (VARCHAR): This column stores the shipping method used for the shipment, such as "Standard Shipping" or "Express Shipping". It provides information about the method used to deliver the shipment.

ExpectedDeliveryDate (DATE): This column stores the expected delivery date for the shipment. It represents the date without the time component and indicates when the shipment is expected to be delivered to the recipient.

The Shipping table is designed to store information about shipments associated with orders. Each shipment is uniquely identified by a ShippingID and includes details such as the order it belongs to, the tracking number, the date it was shipped, the shipping method used, and the expected delivery date. This table facilitates tracking and management of shipments for orders.

**Payments Table**

**Create Table Statement**

**CREATE TABLE Payments (**

**PaymentID INT PRIMARY KEY,**

**OrderID INT,**

**CustomerID INT,**

**PaymentDate DATE,**

**PaymentMethod VARCHAR(50),**

**Amount DECIMAL(10, 2),**

**Status VARCHAR(20),**

**FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),**

**FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));**

**A white background with black text

Description automatically generated**

**Insert Statement**

**INSERT INTO Payments (PaymentID, OrderID, CustomerID, PaymentDate, PaymentMethod, Amount, Status) VALUES**

(701, 401, 301, '2024-04-24', 'Credit Card', 99.99, 'Completed'),

(702, 402, 302, '2024-04-23', 'PayPal', 199.99, 'Completed'),

(703, 403, 303, '2024-04-22', 'Credit Card', 299.99, 'Completed'),

(704, 404, 304, '2024-04-21', 'Credit Card', 399.99, 'Completed'),

(705, 405, 305, '2024-04-20', 'PayPal', 499.99, 'Completed'),

(706, 406, 306, '2024-04-19', 'Credit Card', 599.99, 'Completed'),

(707, 407, 307, '2024-04-18', 'Credit Card', 699.99, 'Completed'),

(708, 408, 308, '2024-04-17', 'PayPal', 799.99, 'Completed'),

(709, 409, 309, '2024-04-16', 'Credit Card', 899.99, 'Completed'),

(710, 410, 310, '2024-04-15', 'Credit Card', 999.99, 'Completed'),

(711, 411, 311, '2024-04-14', 'PayPal', 1099.99, 'Completed'),

(712, 412, 312, '2024-04-13', 'Credit Card', 1199.99, 'Completed'),

(713, 413, 313, '2024-04-12', 'Credit Card', 1299.99, 'Completed'),

(714, 414, 314, '2024-04-11', 'PayPal', 1399.99, 'Completed'),

(715, 415, 315, '2024-04-10', 'Credit Card', 1499.99, 'Completed'),

(716, 416, 316, '2024-04-09', 'Credit Card', 1599.99, 'Completed'),

(717, 417, 317, '2024-04-08', 'PayPal', 1699.99, 'Completed'),

(718, 418, 318, '2024-04-07', 'Credit Card', 1799.99, 'Completed'),

(719, 419, 319, '2024-04-06', 'Credit Card', 1899.99, 'Completed'),

(720, 420, 320, '2024-04-05', 'PayPal', 1999.99, 'Completed'),

(721, 421, 321, '2024-04-04', 'Credit Card', 2099.99, 'Completed'),

(722, 422, 322, '2024-04-03', 'Credit Card', 2199.99, 'Completed'),

(723, 423, 323, '2024-04-02', 'PayPal', 2299.99, 'Completed'),

(724, 424, 324, '2024-04-01', 'Credit Card', 2399.99, 'Completed'),

(725, 425, 325, '2024-03-31', 'Credit Card', 2499.99, 'Completed'),

(726, 426, 326, '2024-03-30', 'PayPal', 2599.99, 'Completed'),

(727, 427, 327, '2024-03-29', 'Credit Card', 2699.99, 'Completed'),

(728, 428, 328, '2024-03-28', 'Credit Card', 2799.99, 'Completed'),

(729, 429, 329, '2024-03-27', 'PayPal', 2899.99, 'Completed'),

(730, 430, 330, '2024-03-26', 'Credit Card', 2999.99, 'Completed');

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Result**

**SELECT \* FROM Payments;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description of Payments Table**

PaymentID (Primary Key, INT): This column serves as the unique identifier for each payment in the table. It is an integer value that uniquely identifies each payment.

OrderID (Foreign Key, INT): This column references the OrderID from the Orders table, establishing a relationship between payments and orders. It indicates which order the payment is associated with.

CustomerID (Foreign Key, INT): This column references the CustomerID from the Customers table, establishing a relationship between payments and customers. It indicates which customer made the payment.

PaymentDate (DATE): This column stores the date when the payment was made. It represents the date without the time component.

PaymentMethod (VARCHAR): This column stores the method used for the payment, such as "Credit Card" or "PayPal". It provides information about the payment method used by the customer.

Amount (DECIMAL): This column stores the amount of the payment. It is a decimal value representing the total amount paid by the customer.

Status (VARCHAR): This column stores the status of the payment, such as "Completed" or "Pending". It indicates the current status of the payment transaction.

The Payments table is designed to store information about payments made by customers for their orders. Each payment is uniquely identified by a PaymentID and includes details such as the order it belongs to, the customer who made the payment, the date of the payment, the payment method used, the amount paid, and the status of the payment transaction. This table facilitates tracking and management of payments for orders.

**Data Retrieval and Simple Reports**

In this section we will see 5 data analysis statements that our database can answer.

1. **Calculate the total sales amount generated by products in each product category.**

SELECT pc.Name, SUM(py.Amount) AS TotalSales

FROM ProductCategory pc

LEFT JOIN Products p ON pc.CategoryID = p.CategoryID

LEFT JOIN OrderDetails od ON p.ProductID = od.ProductID

LEFT JOIN Orders o ON od.OrderID = o.OrderID

LEFT JOIN Payments py ON o.OrderID = py.OrderID

GROUP BY pc.Name;

A screenshot of a computer

Description automatically generated

**Description**

**SELECT pc.Name, SUM(py.Amount) AS TotalSales**

SELECT: This part of the query specifies the columns we want to include in the result set. In this case, we are selecting the Name column from the ProductCategory table and calculating the sum of the Amount column from the Payments table, aliasing it as TotalSales to make the output more descriptive.

**FROM ProductCategory pc**

FROM: This specifies the ProductCategory table as the primary table from which to retrieve category names.

**LEFT JOIN Products p ON pc.CategoryID = p.CategoryID**

LEFT JOIN: This part of the query joins the Products table with the ProductCategory table using a LEFT JOIN operation. The condition specifies that the CategoryID in the Products table must match the CategoryID in the ProductCategory table. This implies that each product is assigned to a specific category, and we want to include product information in our analysis.

**LEFT JOIN OrderDetails od ON p.ProductID = od.ProductID**

LEFT JOIN: Another LEFT JOIN operation is used to join the OrderDetails table with the Products table. The condition specifies that the ProductID in the Products table must match the ProductID in the OrderDetails table. This implies that each product may appear in multiple orders, and we want to include order details in our analysis.

**LEFT JOIN Orders o ON od.OrderID = o.OrderID**

LEFT JOIN: This LEFT JOIN operation links the Orders table with the OrderDetails table. The condition specifies that the OrderID in the OrderDetails table must match the OrderID in the Orders table. This implies that each order may contain multiple order details, and we want to include order information in our analysis.

**LEFT JOIN Payments py ON o.OrderID = py.OrderID**

LEFT JOIN: Finally, another LEFT JOIN operation is used to join the Payments table with the Orders table. The condition specifies that the OrderID in the Orders table must match the OrderID in the Payments table. This implies that each order may have a corresponding payment, and we want to include payment information in our analysis.

**GROUP BY pc.Name;**

In this statement, we are grouping the results based on category name so that we get the result for each product category as requested.

This statement describes a SQL query that calculates the total sales amount generated by products in each product category. It utilizes multiple tables, including ProductCategory, Products, OrderDetails, Orders, and Payments, to gather relevant information. By joining these tables and grouping the data by product category, the query computes the sum of sales amounts associated with each category. The resulting dataset provides insights into the sales performance of products within different categories, helping analyse and understand the revenue distribution across product categories.

1. **Determine the percentage of orders paid using each payment method.**

SELECT PaymentMethod,

COUNT(\*) AS TotalOrders,

(COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM Orders)) AS PercentageOfOrders

FROM Payments

GROUP BY PaymentMethod;

A screenshot of a computer

Description automatically generated

**Description**

**SELECT:**

This part of the query specifies the columns we want to include in the result set.

PaymentMethod: It selects the payment method from the Payments table.

COUNT(\*) AS TotalOrders: It calculates the total number of orders for each payment method and aliases it as TotalOrders to make the output more descriptive.

(COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM Orders)) AS PercentageOfOrders: It calculates the percentage of orders for each payment method. It first counts the total number of orders in the Orders table and then divides the count of orders for each payment method by the total count of orders. The multiplication by 100.0 ensures that the division yields a floating-point result.

**FROM Payments:**

This specifies the Payments table as the source from which to retrieve data.

**GROUP BY PaymentMethod:**

This part of the query groups the result set by the PaymentMethod column.

It ensures that the aggregate functions (COUNT and calculation of percentage) are applied to each distinct payment method.

Result displays the percentage of orders for each payment method. It's calculated as the proportion of orders for each payment method compared to the total number of orders, expressed as a percentage.

This query provides valuable insights into the distribution of orders across different payment methods. It helps understand the popularity and usage of each payment method among customers. By calculating the percentage of orders for each payment method, businesses can identify which methods are preferred by their customers and tailor their payment processing systems accordingly. Additionally, it enables businesses to evaluate the effectiveness of marketing campaigns or incentives associated with specific payment methods.

1. **Find top five customers who has spent the highest total amount across all orders.**

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

SUM(p.Amount) AS TotalAmountSpent

FROM Payments p

JOIN Customers c ON p.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY TotalAmountSpent DESC

LIMIT 5;

A screenshot of a computer

Description automatically generated

**Description**

**SELECT:**

This part of the query specifies the columns we want to include in the result set.

CustomerID: It selects the customer ID from the Payments table.

FirstName: It selects the first name of the customer from the Customers table.

LastName: It selects the last name of the customer from the Customers table.

SUM(p.Amount) AS TotalAmountSpent: It calculates the total amount spent by each customer across all orders and aliases it as TotalAmountSpent to make the output more descriptive.

**FROM Payments p:**

This specifies the Payments table as the primary table from which to retrieve data.

**JOIN Customers c ON p.CustomerID = c.CustomerID:**

This JOIN operation links the Payments table with the Customers table using the CustomerID column as the common key. It ensures that we can retrieve the first name and last name of the customer associated with each payment.

**GROUP BY c.CustomerID, c.FirstName, c.LastName:**

This part of the query groups the result set by the CustomerID, FirstName, and LastName columns.

It ensures that the aggregate function (SUM) is applied to each distinct customer.

**ORDER BY TotalAmountSpent DESC:**

This orders the result set by the TotalAmountSpent column in descending order.

It ensures that the customer with the highest total amount spent appears first in the result set.

**LIMIT 5:**

This limits the result set to only the top 5 customers by total amount spent.

This query retrieves the top 5 customers who have spent the highest total amount across all orders, along with their customer IDs, first names, last names, and total amount spent.

This query provides valuable insights into the top-spending customers, allowing businesses to identify and prioritize their most valuable customers. By understanding the spending behaviour of these top customers, businesses can tailor their marketing strategies, loyalty programs, and customer service efforts to better serve and retain these high-value customers. It enables businesses to focus their resources on nurturing and maintaining relationships with customers who contribute significantly to their revenue.

1. **Count the number of orders shipped by each shipping method.**

SELECT ShippingMethod, COUNT(\*) AS TotalOrdersShipped

FROM Shipping

GROUP BY ShippingMethod;

A screenshot of a computer

Description automatically generated

**Description**

The purpose of this query is to analyse the distribution of orders based on the shipping methods used. It counts the number of orders shipped by each shipping method to provide insights into the usage and popularity of different shipping options.

**SELECT Clause:**

This part of the query specifies the columns to include in the result set.

ShippingMethod: Represents the different shipping methods available for orders.

COUNT(\*) AS TotalOrdersShipped: Calculates the total number of orders shipped using each shipping method and aliases it as TotalOrdersShipped to make the output more descriptive.

**FROM Clause:**

Specifies the primary table from which to retrieve data, which is the Shipping table.

**GROUP BY Clause:**

Groups the result set by the ShippingMethod column.

Ensures that the aggregate function (COUNT) is applied to each distinct shipping method.

The result set will contain one row for each distinct shipping method, with the ShippingMethod column indicating the method and the TotalOrdersShipped column indicating the corresponding count of orders shipped using that method.

1. **Find all customers who have placed an order in the month of March.**

SELECT DISTINCT Customers.CustomerID, Customers.FirstName,

Customers.EmailAddress, Orders.OrderID FROM Customers

JOIN Orders ON Customers.CustomerID = Orders.CustomerID

WHERE MONTH(Orders.OrderDate) = 3;

A screenshot of a computer

Description automatically generated

**Description**

**SELECT DISTINCT Customers.CustomerID, Customers.FirstName, Customers.EmailAddress, Orders.OrderID:**

This part of the query specifies the columns to be retrieved in the result set.

Customers.CustomerID, Customers.FirstName, Customers.EmailAddress: These columns are selected from the Customers table and represent the unique identifier, first name, and email address of customers, respectively.

Orders.OrderID: This column is selected from the Orders table and represents the unique identifier of orders.

**FROM Customers JOIN Orders ON Customers.CustomerID = Orders.CustomerID:**

This part of the query specifies the tables from which to retrieve data and the condition to join them.

FROM Customers: Specifies that the primary table from which to retrieve data is the Customers table.

JOIN Orders ON Customers.CustomerID = Orders.CustomerID: Joins the Orders table with the Customers table using the CustomerID column as the common key. This ensures that only orders associated with existing customers are included in the result set.

**WHERE MONTH(Orders.OrderDate) = 3:**

This part of the query filters the rows based on a specified condition.

MONTH(Orders.OrderDate) = 3: Filters the rows to include only those where the month component of the OrderDate column in the Orders table is equal to 3 (March).

The query retrieves a distinct list of customer IDs, first names, email addresses, and order IDs for orders placed in the month of March. It joins the Customers table with the Orders table based on the CustomerID column and applies a filter to include only orders placed in March. The DISTINCT keyword ensures that only unique combinations of customer details and order IDs are returned in the result set, avoiding duplicates. This query is useful for analysing customer activity and order patterns specifically for the month of March.

1. **Retrieve reviews with ratings above 4.**

SELECT \*

FROM Reviews

WHERE Rating > 4;

A screenshot of a computer

Description automatically generated

**Description**

**SELECT \*:**

This part of the query specifies the columns to be retrieved in the result set.

The asterisk (\*) is a wildcard character that selects all columns from the Reviews table.

**FROM Reviews:**

This part of the query specifies the table from which to retrieve data, which is the Reviews table.

**WHERE Rating > 4:**

This part of the query filters the rows based on a specified condition.

Rating > 4: Specifies that only rows where the value in the Rating column is greater than 4 will be included in the result set.

This condition ensures that only reviews with ratings higher than 4 (typically indicating high satisfaction or positive feedback) are included in the result set.

The query retrieves all columns from the Reviews table but filters the rows to include only those where the rating is greater than 4. This means that only reviews with high ratings are selected, excluding reviews with lower ratings. This query is useful for identifying positive reviews or reviews that indicate high levels of satisfaction with the products or services.

1. **Retrieve orders and customer names where the total value of items is greater than $70.**

SELECT o.OrderID, c.FirstName, c.LastName, SUM(od.Quantity \* od.UnitPrice) AS TotalValue

FROM OrderDetails od

JOIN Orders o ON od.OrderID = o.OrderID

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY o.OrderID, c.FirstName, c.LastName

HAVING TotalValue > 70;

A screenshot of a computer program

Description automatically generated

**Description**

**SELECT:** Specifies the columns to be retrieved in the result set.

o.OrderID: Represents the unique identifier of each order.

c.FirstName: Represents the first name of the customer who made the order.

c.LastName: Represents the last name of the customer who made the order.

SUM(od.Quantity \* od.UnitPrice) AS TotalValue: Calculates the total value of each order (quantity \* unit price) and aliases it as TotalValue.

**FROM:** Specifies the primary table from which to retrieve data, which is the OrderDetails table.

JOIN: Joins the OrderDetails table with the Orders table based on the common OrderID column, and then joins the Orders table with the Customers table based on the common CustomerID column.

**JOIN Orders o ON od.OrderID = o.OrderID:** Joins the OrderDetails table with the Orders table.

**JOIN Customers c ON o.CustomerID = c.CustomerID:** Joins the Orders table with the Customers table.

**GROUP BY:** Groups the result set by the OrderID, FirstName, and LastName columns.

Ensures that the aggregate function (SUM) is applied to each distinct order.

**HAVING:** Filters the groups based on a specified condition (TotalValue > 70).

This ensures that only orders with a total value exceeding $70 are included in the result set.

The query retrieves the order ID, customer's first name, and last name for each order where the total value of the order exceeds $70. It joins the OrderDetails table with the Orders table to get the order details and then further joins the Orders table with the Customers table to get the customer details associated with each order. The SUM function calculates the total value of each order, and the HAVING clause filters the result to include only orders with a total value greater than $70. This query is useful for identifying high-value orders and analysing customer spending patterns.

1. **Identify top five products with highest quantity sold.**

SELECT p.ProductID, p.ProductName, SUM(od.Quantity) AS TotalQuantitySold

FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY p.ProductID, p.ProductName

ORDER BY TotalQuantitySold DESC

LIMIT 5;

A screenshot of a computer

Description automatically generated

**Description**

**SELECT:** Specifies the columns to be retrieved in the result set.

p.ProductID: Represents the unique identifier of each product.

p.ProductName: Represents the name of each product.

SUM(od.Quantity) AS TotalQuantitySold: Calculates the total quantity sold for each product and aliases it as TotalQuantitySold.

**FROM:** Specifies the primary table from which to retrieve data, which is the OrderDetails table.

**JOIN:** Joins the OrderDetails table with the Products table based on the common ProductID column.

JOIN Products p ON od.ProductID = p.ProductID: Joins the OrderDetails table with the Products table.

**GROUP BY:** Groups the result set by the ProductID and ProductName columns.

Ensures that the aggregate function (SUM) is applied to each distinct product.

**ORDER BY:** Orders the result set by the TotalQuantitySold column in descending order.

This ensures that products with the highest total quantity sold appear first in the result set.

**LIMIT:** Limits the result set to only the top 5 products by total quantity sold.

The query retrieves the product ID, product name, and total quantity sold for each product, focusing on the top 5 products with the highest total quantity sold. It joins the OrderDetails table with the Products table to get product details and then groups the result by product ID and name. The SUM function calculates the total quantity sold for each product, and the ORDER BY clause sorts the result by total quantity sold in descending order, ensuring that the top-selling products appear at the top of the list. Finally, the LIMIT clause limits the result to only the top 5 products. This query is useful for identifying popular or best-selling products in the dataset.

**Conclusion**

In this project, we used SQL queries on an e-commerce database to carry out a variety of data analysis activities. We asked questions about a wide range of topics, including order processing, customer insights, sales data, product performance, and review evaluation. Below is a synopsis of our main discoveries for each query:   
  
**Total Sales by Product Category:** To better understand how revenue is distributed throughout categories, we evaluated sales quantities by product category.

**Order Payment Method Percentage:** We were able to learn more about client payment preferences by measuring the utilisation of different payment methods.

**Top Customers by Total Amount Spent:** Using data on the five customers who spent the most overall, we were able to develop marketing tactics that were specifically targeted at them.

**Orders dispatched by Shipping option:** To get insight into shipping preferences, we tallied the number of orders dispatched by each delivery option.

**Consumers with March Orders:** We were able to analyse temporal customer patterns by identifying consumers who placed orders in March.

**Favourable Reviews with Ratings Above 4:** We found reviews that had high ratings, which enables companies to highlight favourable client comments.

**Orders Over $70:** We emphasised orders with a value more than $70 to expose high-value deals and consumer purchasing trends.

**Top Products by Quantity Sold:** To help draw attention to popular things, we determined the top five products based on quantity sold.

We were able to make well-informed decisions to improve tactics, increase customer happiness, and increase income through this project's invaluable insights into e-commerce operations. SQL queries demonstrated how well they could extract insights that could be put into practice, highlighting the significance of data-driven decision-making in contemporary business.