Learner name: Rohaan Pisavadia

Project: eCommerce Simple Database

Purpose: A dedicated database is created to store and manage all the information for the e-commerce system.

Step 1: Create the Database

Command Used:

CREATE DATABASE command creates a new database named 'eCommerceDB'.

USE command selects this database for subsequent operations, and

SELECT DATABASE() confirms the selected database.

```
-- Step 1 Create the database eCommerceDB

CREATE DATABASE eCommerceDB;
-- Select the eCommerceDB database to use for the following operations
SHOW DATABASES;
USE eCommerceDB;
-- Confirm the selected database
SELECT DATABASE();
```

Command SHOW DATABASES to see the CREATE DATABASE Output:

Database
eCommerceDB

Step 2: Create Tables and Define Relationships

The tables are created to organise data about products, customers, orders, and more. Here's what each table does:

2.1 Products Table

- Purpose: Stores details about the products available in the e-commerce store.
- Fields/Columns: productID (Primary Key), name, description, price, stockQuantity.
- Command Used: CREATE TABLE command defines the structure for tbl_products.

```sql
CREATE TABLE tbl_products (
    productID INT AUTO_INCREMENT PRIMARY KEY,   -- Unique identifier for each product
    name VARCHAR(100) NOT NULL,                  -- Product name
    description TEXT,                            -- Description of the product
    price DECIMAL(10, 2) NOT NULL,               -- Price of the product
    stockQuantity INT NOT NULL                  -- Quantity of the product in stock
);
```

Command DESC 'tableName' for table structure - Output:

```sql
-- See table structure
DESC tbl_products;
```

100% 19:25

Result Grid Filter Rows: Search Export:

Field	Type	Null	Key	Default	Extra
productID	int	NO	PRI	NULL	auto_increment
name	varchar(100)	NO		NULL	
description	text	YES		NULL	
price	decimal(10,2)	NO		NULL	
stockQuantity	int	NO		NULL	

2.2 Customers Table

Purpose: Stores information about the customers of the e-commerce store.

Fields/Columns: customerID (Primary Key), firstName, lastName, email, phoneNumber, address.

Command Used: CREATE TABLE command defines the structure for tbl_customers.

```sql
CREATE TABLE tbl_customers (
    customerID INT AUTO_INCREMENT PRIMARY KEY,  -- Unique identifier for each customer
    firstName VARCHAR(50) NOT NULL,             -- Customer's first name
    lastName VARCHAR(50) NOT NULL,              -- Customer's last name
    email VARCHAR(100) NOT NULL,                -- Customer's email address
    phoneNumber VARCHAR(15),                    -- Customer's phone number
    address TEXT                                -- Customer's address
);
```

Command DESC 'tableName' for table structure - Output:

```
40 •  DESC tbl_customers;
41
```

100% 20:40

Result Grid Filter Rows: Search Export:

Field	Type	Null	Key	Default	Extra
customerID	int	NO	PRI	NULL	auto_increment
firstName	varchar(50)	NO		NULL	
lastName	varchar(50)	NO		NULL	
email	varchar(100)	NO		NULL	
phoneNumber	varchar(15)	YES		NULL	
address	text	YES		NULL	

2.3 Orders Table

Purpose: Manages customer orders and links them to products and customers.

Fields/Columns: orderID (Primary Key), customerID (Foreign Key), orderDate, totalAmount.

Command Used: CREATE TABLE command defines the structure for tbl_orders.

```
46 • CREATE TABLE tbl_orders (
47       orderID INT AUTO_INCREMENT PRIMARY KEY,
48       -- Unique identifier for each order
49       customerID INT,
50       -- Foreign key to identify the customer who placed the order
51       orderDate DATETIME DEFAULT
52       CURRENT_TIMESTAMP,  -- Date and time when the order was placed
53       status ENUM
54       ('Pending', 'Shipped', 'Delivered', 'Cancelled')
55       DEFAULT 'Pending',  -- Status of the order
56       FOREIGN KEY (customerID)
57       REFERENCES tbl_customers (customerID)
58       ON DELETE CASCADE   -- Delete orders if the customer is deleted
59   );
```

Command DESC 'tableName' for table structure - Output:

```
60 •  DESC tbl_orders;
61
```

100% 17:60

Result Grid Filter Rows: Search Export:

Field	Type	Null	Key	Default	Extra
orderID	int	NO	PRI	NULL	auto_increment
customerID	int	YES	MUL	NULL	
orderDate	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
status	enum('Pending','Shipped','Delivered','Cancelled')	YES		Pending	

2.4 Order Items Table

Purpose: Provides a breakdown of items in each order.

Fields/Columns: orderItemID (Primary Key), orderID (Foreign Key), productID (Foreign Key), quantity, price.

ON DELETE CASCADE constraint for orders and orderItems relationship has been implemented

Command Used: CREATE TABLE command defines the structure for tbl_orderItems.

```sql
CREATE TABLE tbl_orderItems (
    orderItemID INT AUTO_INCREMENT PRIMARY KEY,
    -- Unique identifier for each order item
    orderID INT,
    -- Foreign key linking to the Orders table
    productID INT,
    -- Foreign key linking to the Products table
    quantity INT NOT NULL,
    -- Quantity of the product ordered
    price DECIMAL(10, 2) NOT NULL,
    -- Price of the product at the time of order
    FOREIGN KEY (orderID)
    REFERENCES tbl_orders (orderID)
    -- Creating relationship with primaryKey in tbl_orders
    ON DELETE CASCADE, -- Delete order items if the order is deleted
    FOREIGN KEY (productID)
    REFERENCES tbl_products (productID)
    -- -- Creating relationship with primaryKey in tbl_products
);
```

Command DESC 'tableName' for table structure - Output:

```sql
DESC tbl_orderItems;
```

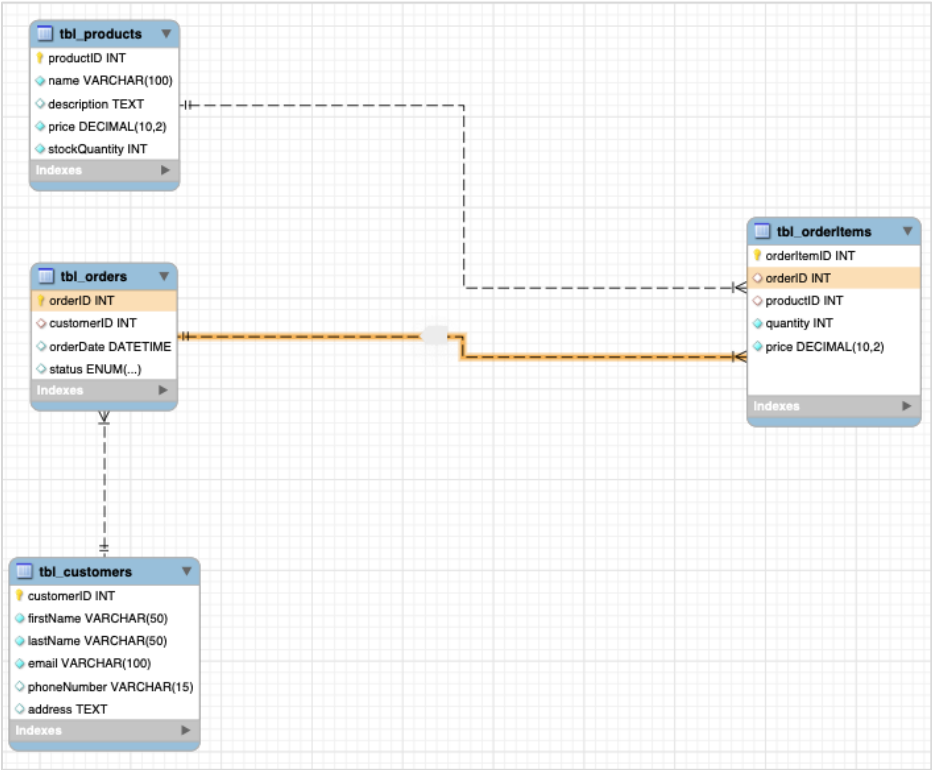
100% 13:84

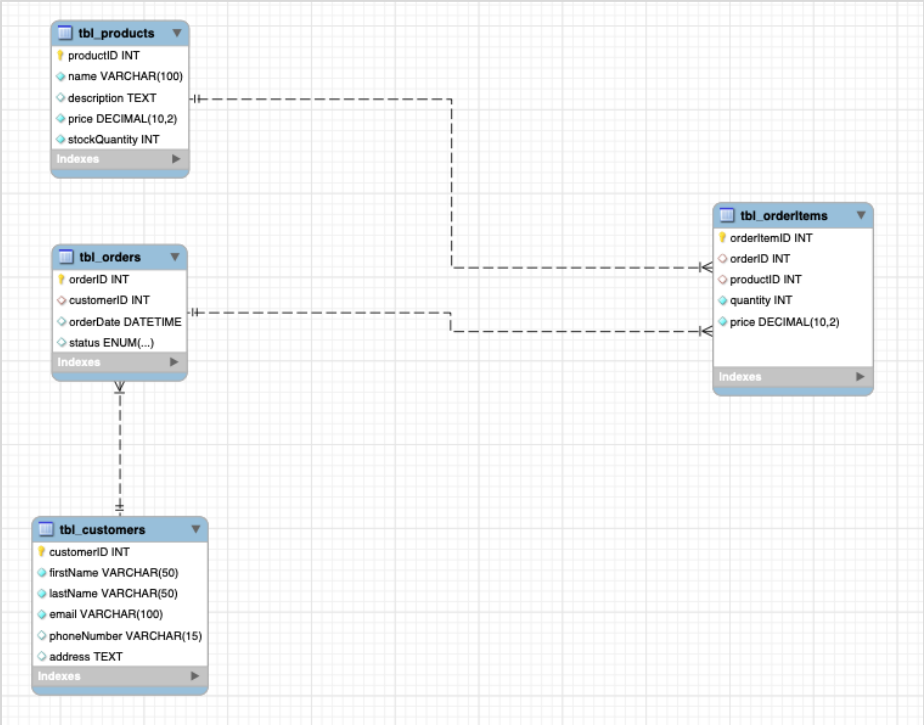
Result Grid Filter Rows: Search Export:

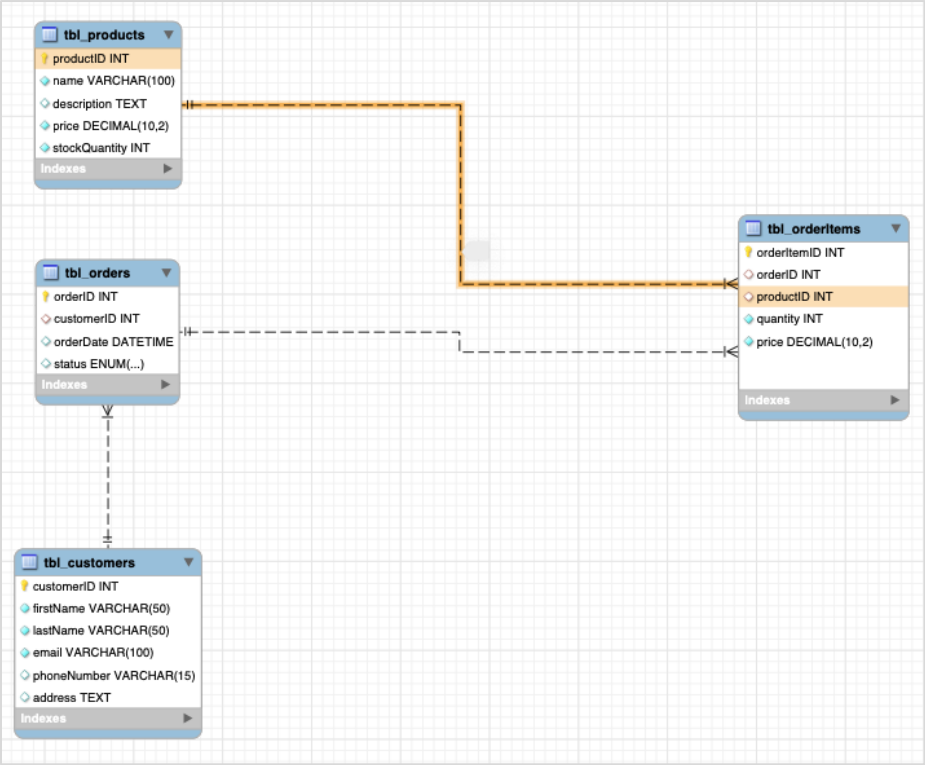
Field	Type	Null	Key	Default	Extra
orderItemID	int	NO	PRI	NULL	auto_increment
orderID	int	YES	MUL	NULL	
productID	int	YES	MUL	NULL	
quantity	int	NO		NULL	
price	decimal(10,2)	NO		NULL	

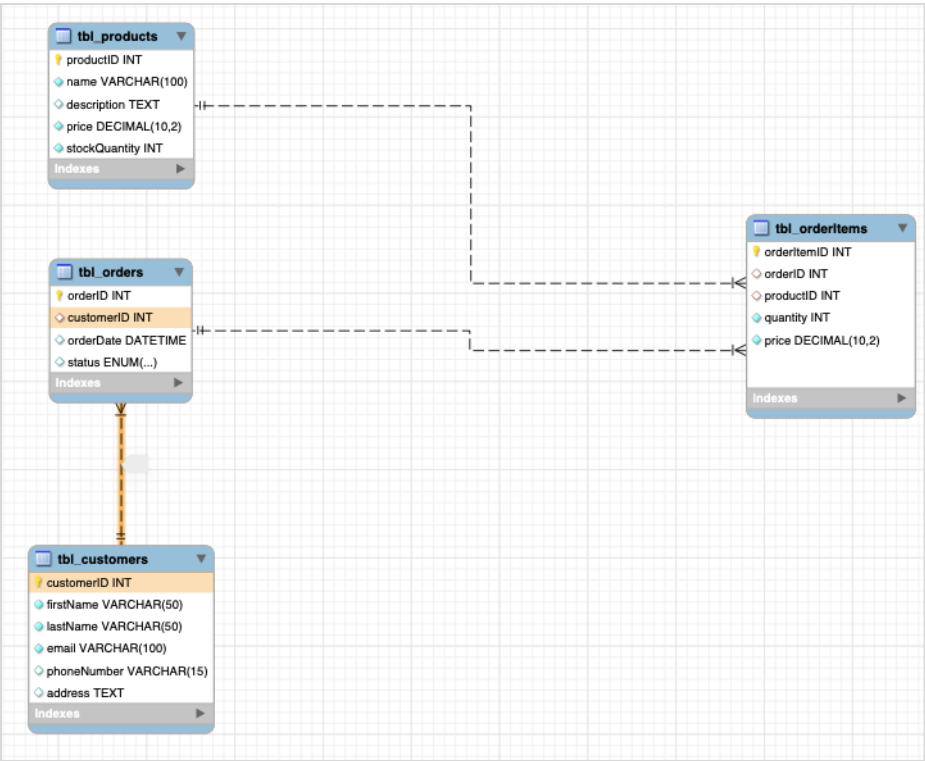
EER diagrams (Enhanced Entity-Relationship diagram)

An **EER diagram** is a visual representation of a database's structure.






tbl_products
productID INT
name VARCHAR(100)
description TEXT
price DECIMAL(10,2)
stockQuantity INT
Indexes
tbl_orderItems
orderItemID INT
orderID INT
productID INT
quantity INT
price DECIMAL(10,2)
Indexes
tbl_orders
orderID INT
customerID INT
orderDate DATETIME
status ENUM(...)
Indexes
tbl_customers
customerID INT
firstName VARCHAR(50)
lastName VARCHAR(50)
email VARCHAR(100)
phoneNumber VARCHAR(15)
address TEXT
Indexes


tbl_products
productID INT
name VARCHAR(100)
description TEXT
price DECIMAL(10,2)
stockQuantity INT
Indexes
tbl_orderItems
orderItemID INT
orderID INT
productID INT
quantity INT
price DECIMAL(10,2)
Indexes
tbl_orders
orderID INT
customerID INT
orderDate DATETIME
status ENUM(...)
Indexes
tbl_customers
customerID INT
firstName VARCHAR(50)
lastName VARCHAR(50)
email VARCHAR(100)
phoneNumber VARCHAR(15)
address TEXT
Indexes

Step 3: Insert Data into Tables

Purpose: Sample data is added to the tables to simulate real-world e-commerce scenarios, including products (5 products), customers (5 customers), and their orders (10 orders).

```sql
INSERT INTO tbl_products (name, description, price, stockQuantity)
VALUES
('Laptop', 'High performance laptop', 1200.00, 10),
('Headphones', 'Noise cancelling headphones', 150.00, 30),
('Mouse', 'Wireless mouse', 25.00, 50),
('Keyboard', 'Mechanical keyboard', 75.00, 20),
('Monitor', '27-inch 4K display', 300.00, 15);
```

```sql
INSERT INTO tbl_customers (firstName, lastName, email, phoneNumber, address)
VALUES
('Cob', 'Medwell', 'cmedwell0@npr.org', '07555 987654', 'Liverpool'),
('Leonie', 'Masham', 'lmasham1@gmail.com', '07444 222111', 'Birmingham'),
('Bastien', 'Springle', 'bspringle2@hotmail.co.uk', '07911 123456', 'SW1A 1AA'),
('Kipp', 'Velasquez', 'kvelasquez3@marketwatch.com', '07822 555444', '222 Imaginary Drive'),
('Evangeline', 'Nevett', 'enevett4@msn.com', '07822 555444', 'Manchester');
```

```sql
INSERT INTO tbl_orders (customerID, orderDate, status)
VALUES
(1, '2024-07-20 10:30:00', 'Pending'),
(2, '2024-07-21 14:15:00', 'Shipped'),
(3, '2024-07-22 09:45:00', 'Delivered'),
(4, '2024-07-23 16:30:00', 'Delivered'),
(5, '2024-07-24 11:20:00', 'Cancelled'),
(1, '2024-07-25 12:50:00', 'Shipped'),
(2, '2024-07-26 15:00:00', 'Delivered'),
(3, '2024-07-27 08:00:00', 'Pending'),
(5, '2024-07-28 17:30:00', 'Shipped'),
(5, '2024-07-29 13:45:00', 'Pending');
```

Step 4: Query the Data

4.1 Listing All Products, All Customers, and All Orders

Command SELECT * FROM tbl_products; to retrieve all products

```
102 •  SELECT * FROM tbl_products;
103
```

100% 28:102

Result Grid Filter Rows: Search Edit: Export/Import:

productID	name	description	price	stockQuanti...
1	Laptop	High performance laptop	1200.00	13
2	Headphones	Noise cancelling headphones	150.00	30
3	Mouse	Wireless mouse	25.00	50
4	Keyboard	Mechanical keyboard	75.00	30
5	Monitor	27-inch 4K display	300.00	15
NULL	NULL	NULL	NULL	NULL

Command SELECT * FROM tbl_customers; to retrieve all customers

```
SELECT * FROM tbl_orders;
```

100% 39:129

Result Grid Filter Rows: Search Edit: Export/Import:

orderID	customerID	orderDate	status
1	1	2024-07-20 10:30:00	Pending
2	2	2024-07-21 14:15:00	Shipped
3	3	2024-07-22 09:45:00	Delivered
4	4	2024-07-23 16:30:00	Delivered
5	5	2024-07-24 11:20:00	Cancelled
6	1	2024-07-25 12:50:00	Shipped
7	2	2024-07-26 15:00:00	Delivered
8	3	2024-07-27 08:00:00	Pending
9	5	2024-07-28 17:30:00	Shipped
10	5	2024-07-29 13:45:00	Pending
NULL	NULL	NULL	NULL

Command SELECT * FROM tbl_orders; to retrieve customer orders

```
SELECT * FROM tbl_customers;
```

100% 67:111

Result Grid Filter Rows: Search Edit: Export/Import:

customerID	firstName	lastName	email	phoneNumber	address
1	Cob	Medwell	cmedwell0@npr.org	07555 987654	Liverpool
2	Leonie	Masham	lmasham1@gmail.com	07444 222111	Birmingham
3	Bastien	Springle	bspringle2@hotmail.co.uk	07911 123456	SW1A 1AA
4	Kipp	Velasquez	kvelasquez3@marketwatch.com	07822 555444	222 Imaginary Drive
5	Evangeline	Nevett	enevett4@msn.com	07822 555444	Manchester
NULL	NULL	NULL	NULL	NULL	NULL

4.2 Listing All Orders with Customer Details

Purpose: Retrieve a comprehensive list of all orders along with their corresponding customer details.

Fields/Columns Used: orderID, orderDate, status, firstName, lastName, email.

Command Used: A query joins the tbl_orders and tbl_customers tables to link orders with customer details, sorted by orderDate.

```
SELECT tbl_orders.orderID, tbl_orders.orderDate,
tbl_orders.status, tbl_customers.firstName,
tbl_customers.lastName, tbl_customers.email
FROM tbl_orders
INNER JOIN tbl_customers ON tbl_orders.customerID = tbl_customers.customerID
ORDER BY tbl_orders.orderDate;
```

Output:

orderID	orderDate	status	firstName	lastName	email
1	2024-07-20 10:30:00	Pending	Cob	Medwell	cmedwell0@npr.org
2	2024-07-21 14:15:00	Shipped	Leonie	Masham	lmasham1@gmail.com
3	2024-07-22 09:45:00	Delivered	Bastien	Springle	bspringle2@hotmail.co.uk
4	2024-07-23 16:30:00	Delivered	Kipp	Velasquez	kvelasquez3@marketwatch.com
5	2024-07-24 11:20:00	Cancelled	Evangeline	Nevett	enevett4@msn.com
6	2024-07-25 12:50:00	Shipped	Cob	Medwell	cmedwell0@npr.org
7	2024-07-26 15:00:00	Delivered	Leonie	Masham	lmasham1@gmail.com
8	2024-07-27 08:00:00	Pending	Bastien	Springle	bspringle2@hotmail.co.uk
9	2024-07-28 17:30:00	Shipped	Evangeline	Nevett	enevett4@msn.com
10	2024-07-29 13:45:00	Pending	Evangeline	Nevett	enevett4@msn.com

4.3 Query to Calculate Days Since a Customer Placed an Order

Purpose: Calculate the number of days elapsed since each customer placed an order.

Fields/Columns Used: customerID, fullName, orderID, orderDate, daysSinceOrder.

Command Used: A query joins the tbl_customers and tbl_orders tables and uses the DATEDIFF function to calculate elapsed days.

```
SELECT
tbl_customers.customerID, CONCAT(tbl_customers.firstName,
' ', tbl_customers.lastName) AS fullName,
tbl_orders.orderID, tbl_orders.orderDate, DATEDIFF(CURDATE(), tbl_orders.orderDate)
AS daysSinceOrder FROM tbl_customers
INNER JOIN tbl_orders ON tbl_customers.customerID = tbl_orders.customerID
ORDER BY daysSinceOrder DESC;
```

Output:

customerID	fullName	orderID	orderDate	daysSinceOrder
1	Cob Medwell	1	2024-07-20 10:30:00	118
2	Leonie Masham	2	2024-07-21 14:15:00	117
3	Bastien Springle	3	2024-07-22 09:45:00	116
4	Kipp Velasquez	4	2024-07-23 16:30:00	115
5	Evangeline Nevett	5	2024-07-24 11:20:00	114
1	Cob Medwell	6	2024-07-25 12:50:00	113
2	Leonie Masham	7	2024-07-26 15:00:00	112
3	Bastien Springle	8	2024-07-27 08:00:00	111
5	Evangeline Nevett	9	2024-07-28 17:30:00	110
5	Evangeline Nevett	10	2024-07-29 13:45:00	109

Step 5: Update Product Stock After an Order

Purpose: Adjust stock levels for products automatically after an order is placed, ensuring accurate inventory management.

Fields Affected: productID, stockQuantity.

Command Used: SELECT statement used first to see initial stock quantity for example customer orderID = 1.

```
-- Check initial stock quantities for products in orderID = 1
SELECT productID, stockQuantity
FROM tbl_products
WHERE productID IN (SELECT productID FROM tbl_orderItems WHERE orderID = 1);
```

Output:

productID	stockQuanti...
1	10
2	30

The UPDATE statement adjusts the stockQuantity field in the tbl_products table by subtracting quantities sold.

```
SELECT productID, stockQuantity FROM tbl_products
WHERE productID IN (SELECT productID FROM tbl_orderItems WHERE orderID = 1);
UPDATE tbl_products
SET stockQuantity = stockQuantity - (SELECT quantity FROM tbl_orderItems
WHERE productID = tbl_products.productID
AND orderID = 1)
WHERE productID IN (SELECT productID
FROM tbl_orderItems
WHERE orderID = 1);
```

```
-- Check updated stock quantities for products in orderID = 1
SELECT productID, stockQuantity
FROM tbl_products
WHERE productID IN (SELECT productID FROM tbl_orderItems WHERE orderID = 1);
```

Output:

productID	stockQuanti...
1	9
2	29

Step 6: Generating Reports

6.1 Total Sales by Product

Purpose: Generate a report to show the revenue generated by each product.

Command:

```
-- 6.1 Total sales by product using the sum up
-- quantity * price for each productID in table orderItems
-- and join with table products to get the product names
SELECT tbl_products.productID,
tbl_products.name,
SUM(tbl_orderItems.quantity * tbl_orderItems.price)
AS total_sales
FROM tbl_orderItems
INNER JOIN tbl_products ON tbl_orderItems.productID = tbl_products.productID
GROUP BY tbl_products.productID
ORDER BY total_sales DESC;
```

Output:

productID	name	total_sales
1	Laptop	4800.00
2	Headphones	600.00
5	Monitor	600.00
4	Keyboard	225.00
3	Mouse	150.00

6.2 Total Orders by Customer

Purpose: Display the number of orders placed by each customer.

Fields/Columns shown: customerID, firstName, lastName, total_orders.

Command:

```
-- 6.2 Total orders by customer, by counting the number of
-- orderIDs in the orders table grouped by customerID and
-- join with customers table to get customer details
SELECT tbl_customers.customerID, tbl_customers.firstName, tbl_customers.lastName,
COUNT(tbl_orders.orderID) AS total_orders FROM tbl_orders
INNER JOIN tbl_customers ON tbl_orders.customerID = tbl_customers.customerID
GROUP BY tbl_customers.customerID
ORDER BY total_orders DESC;
```

Output:

customerID	firstName	lastName	total_orders
5	Evangeline	Nevett	3
1	Cob	Medwell	2
2	Leonie	Masham	2
3	Bastien	Springle	2
4	Kipp	Velasquez	1

6.3 Total Order Value Per Customer

Purpose: Calculate the total value of orders made by each customer.

Fields/Columns shown: customerID, fullName, totalOrderValue.

Command:

```
-- 6.3 Total order value per customer
-- Obtaining customer firstName and lastName as 'fullName'
-- Calculating the SUM total number of ordered items * price
-- to give a total order value
-- join tables customers, orders and orderItems
SELECT
tbl_customers.customerID,CONCAT(tbl_customers.firstName, ' ', tbl_customers.lastName)
AS fullName,
SUM(tbl_orderItems.price * tbl_orderItems.quantity) AS totalOrderValue
FROM tbl_customers INNER JOIN tbl_orders ON tbl_customers.customerID
= tbl_orders.customerID
INNER JOIN tbl_orderItems ON tbl_orders.orderID = tbl_orderItems.orderID
GROUP BY tbl_customers.customerID ORDER BY totalOrderValue DESC;
```

Output:

customerID	fullName	totalOrderVal...
5	Evangeline Nevett	3900.00
1	Cob Medwell	1800.00
4	Kipp Velasquez	300.00
3	Bastien Springle	225.00
2	Leonie Masham	150.00

Step 7: Implementing ON DELETE CASCADE

Purpose: Automatically delete dependent records in the tbl_orderItems table when an associated order is removed, ensuring referential integrity.

Command Used: A foreign key constraint is added to the tbl_orderItems table with the ON DELETE CASCADE option. (see screenshot example in Step 2.4)

Verification Steps: Add a new order and its associated items, delete the order, and confirm that related items in tbl_orderItems are also removed.

Command: INSERT

```sql
-- inserting a new order
INSERT INTO tbl_orders (customerID, orderDate, status)
VALUES
(2, '2024-08-13 13:40:00', 'Pending');

-- inserting data related to new orderID = 12
INSERT INTO tbl_orderItems (orderID, productID, quantity, price)
VALUES
(12, 1, 3, 1200.00);    -- Order 12: 3 Laptops
```

Output:

```sql
-- new order has been created with orderID = 12
SELECT * FROM tbl_orders;
```

12	2	2024-08-13 13:40:00	Pending
NULL	NULL	NULL	NULL

```sql
-- new orderItems record created with orderItems = 15
SELECT * FROM tbl_orderItems;
```

15	12	1	3	1200.00
NULL	NULL	NULL	NULL	NULL

Command: DELETE

```sql
-- delete record with orderID = 12
-- on doing so this will then delete the related record
-- in orderItems table
DELETE FROM tbl_orders WHERE orderID = 12;
```

Output: Shows the record has been removed from both tables.

```
SELECT * FROM tbl_orders;
```

8	3	2024-07-27 08:00:00	Pending
9	5	2024-07-28 17:30:00	Shipped
10	5	2024-07-29 13:45:00	Pending
NULL	NULL	NULL	NULL

```
SELECT * FROM tbl_orderItems;
```

10	9	5	1	300.00
11	10	1	2	1200.00
NULL	NULL	NULL	NULL	NULL

Additional Step: Stored Procedures and CASE Updates

Stored Procedure 1: Retrieve Customer Order Details

Purpose: Provide a detailed report of customer orders, including products purchased and their details.

Fields/Columns Shown: customerID (Filters results for a specific customer), fullName email, address, orderID, productID, productName, description, productPrice, quantity and totalProductValue.

Command Used: The PROCEDURE and CALL retrieves detailed order information for a specific customer (or all

```
-- Procedure 1
-- Using a stored procedure for when all or specific customer
-- detailed record wants to be viewed
-- providing customer details and products ordered.
DELIMITER $$
CREATE PROCEDURE GetCustomerOrderDetails( IN p_customerID INT)
BEGIN
SELECT tbl_customers.customerID, CONCAT(tbl_customers.firstName, ' ', tbl_customers.lastName) AS fullName,
tbl_customers.email,tbl_customers.address,tbl_orders.orderID,tbl_products.productID,
tbl_products.name AS productName,tbl_products.description,tbl_orderItems.price AS productPrice,
tbl_orderItems.quantity, FORMAT(tbl_orderItems.price * tbl_orderItems.quantity, 2) AS totalProductValue
FROM tbl_customers INNER JOIN tbl_orders ON tbl_customers.customerID = tbl_orders.customerID
INNER JOIN tbl_orderItems ON tbl_orders.orderID = tbl_orderItems.orderID
INNER JOIN tbl_products ON tbl_orderItems.productID = tbl_products.productID
WHERE p_customerID IS NULL OR tbl_customers.customerID = p_customerID
ORDER BY tbl_customers.customerID, tbl_orders.orderID, tbl_products.productID;
END $$

DELIMITER $$
```

customers if no customer ID is provided).

```
-- To call the procedure and generate the report
-- retrieves the detailed order report for the customer/s
-- by using NULL (to get all customers
-- by using customerID = 3 for example for a specific customer report
CALL GetCustomerOrderDetails(NULL);
CALL GetCustomerOrderDetails(3);
```

Output:

For all customer information

customerID	fullName	email	address	orderID	productID	productName	description	productPri...	quantity	totalProductVal...
1	Cob Medwell	cmedwell0@npr.org	Liverpool	1	1	Laptop	High performance laptop	1200.00	1	1,200.00
1	Cob Medwell	cmedwell0@npr.org	Liverpool	1	2	Headphones	Noise cancelling headphones	150.00	1	150.00
1	Cob Medwell	cmedwell0@npr.org	Liverpool	6	2	Headphones	Noise cancelling headphones	150.00	3	450.00
2	Leonie Masham	lmasham1@gmail.com	Birmingham	2	3	Mouse	Wireless mouse	25.00	2	50.00
2	Leonie Masham	lmasham1@gmail.com	Birmingham	7	3	Mouse	Wireless mouse	25.00	4	100.00
3	Bastien Springle	bspringle2@hotmail.co.uk	SW1A 1AA	3	4	Keyboard	Mechanical keyboard	75.00	1	75.00
3	Bastien Springle	bspringle2@hotmail.co.uk	SW1A 1AA	8	4	Keyboard	Mechanical keyboard	75.00	2	150.00
4	Kipp Velasquez	kvelasquez3@marketwatch.com	222 Imaginary Drive	4	5	Monitor	27-inch 4K display	300.00	1	300.00
5	Evangeline Nevett	enevett4@msn.com	Manchester	5	1	Laptop	High performance laptop	1200.00	1	1,200.00
5	Evangeline Nevett	enevett4@msn.com	Manchester	9	5	Monitor	27-inch 4K display	300.00	1	300.00
5	Evangeline Nevett	enevett4@msn.com	Manchester	10	1	Laptop	High performance laptop	1200.00	2	2,400.00

For specific customer report – customerID = 3

customerID	fullName	email	address	orderID	productID	productName	description	productPri...	quantity	totalProductVal...
3	Bastien Springle	bspringle2@hotmail.co.uk	SW1A 1AA	3	4	Keyboard	Mechanical keyboard	75.00	1	75.00
3	Bastien Springle	bspringle2@hotmail.co.uk	SW1A 1AA	8	4	Keyboard	Mechanical keyboard	75.00	2	150.00

Stored Procedure 2: Generate Picking List for Pending Orders

Purpose: Create a report listing the products and quantities needed for orders with a 'Pending' status.

Fields/Columns Shown: productID, productName, totalQty.

Command Used: The PROCEDURE aggregates data from the tbl_orderItems table for orders with the 'Pending' status. CALL to run the function and retrieve the picking list report.

```
-- Procedure 2 - Retrieve a pickingList for pending orders
-- for when an eCommerce company needs to
-- pick items from their stored location
DELIMITER $$

CREATE PROCEDURE GetPickingList()
BEGIN
SELECT tbl_products.productID,tbl_products.name AS productName,
SUM(tbl_orderItems.quantity) AS totalQty FROM tbl_orderItems
INNER JOIN tbl_orders ON tbl_orderItems.orderID = tbl_orders.orderID
INNER JOIN tbl_products ON tbl_orderItems.productID = tbl_products.productID
WHERE tbl_orders.status = 'Pending' GROUP BY tbl_products.productID, tbl_products.name
ORDER BY totalQty DESC;
END $$

DELIMITER $$
```

```
-- Retrieve pickingList by calling the procedure
CALL GetPickingList();
```

Output:

productID	productName	totalQty
1	Laptop	3
4	Keyboard	2
2	Headphones	1

CASE-Based Stock Update

Purpose: Conditionally update stock levels for specific products.

Fields Affected: productID, stockQuantity.

Command Used: The CASE statement applies different stock adjustments based on the product ID.

Current stock levels:

productID	name	description	price	stockQuanti...
1	Laptop	High performance laptop	1200.00	9
2	Headphones	Noise cancelling headphones	150.00	29
3	Mouse	Wireless mouse	25.00	50
4	Keyboard	Mechanical keyboard	75.00	30
5	Monitor	27-inch 4K display	300.00	15
NULL	NULL	NULL	NULL	NULL

```
UPDATE tbl_products
SET stockQuantity =
(CASE productID
    WHEN 1 THEN stockQuantity + 5  -- Add by 5 for productID 1
    WHEN 2 THEN stockQuantity + 2  -- Add by 2 for productID 2
    WHEN 4 THEN stockQuantity + 10 -- Add by 10 for productID 4
    ELSE stockQuantity                          -- No change for other productIDs
END)
WHERE productID IN (1, 2, 4);
```

Output: Stock levels after update:

productID	name	description	price	stockQuanti...
1	Laptop	High performance laptop	1200.00	14
2	Headphones	Noise cancelling headphones	150.00	31
3	Mouse	Wireless mouse	25.00	50
4	Keyboard	Mechanical keyboard	75.00	40
5	Monitor	27-inch 4K display	300.00	15
NULL	NULL	NULL	NULL	NULL