CLEAR, CONCRETE EXAMPLES USING **PRODUCTS, CATEGORIES, AND SALES**, and you specifically want to see **multiple (composite) primary keys** plus foreign keys. Let's walk through it cleanly like you'd see in a real database

---

1. Categories table (simple primary key)

Each category is unique.

```
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(100)
);
```

Primary Key

- `CategoryID`

---

2. Products table (foreign key + primary key)

Each product belongs to one category.

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    CategoryID INT,
    Price DECIMAL(10,2),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

Primary Key

- `ProductID`

Foreign Key

- `CategoryID → Categories(CategoryID)`

---

3. Sales table (COMPOSITE primary key)

Here's where multiple primary keys come in.

A sale is uniquely identified by:

- `SaleID`
- `ProductID`

Together, they form a **composite primary key**.

```
CREATE TABLE Sales (
    SaleID INT,
    ProductID INT,
    SaleDate DATE,
    Quantity INT,
    TotalAmount DECIMAL(10,2),
    PRIMARY KEY (SaleID, ProductID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

Composite Primary Key

- `(SaleID, ProductID)`

Foreign Key

- `ProductID → Products(ProductID)`

Why composite?

- One sale (`SaleID`) can contain **multiple products**
- The same product can appear in **many sales**
- The combination guarantees uniqueness

---

4. Example data (to make it click)

Categories

CategoryID	CategoryName
1	Electronics
2	Clothing

Products

ProductID	ProductName	CategoryID	Price
101	Laptop	1	1000
102	T-Shirt	2	25

Sales

SaleID	ProductID	SaleDate	Quantity	TotalAmount
1	101	2025-02-01	1	1000
1	102	2025-02-01	2	50
2	101	2025-02-02	1	1000

Notice:

- `SaleID = 1` appears **twice**
- `ProductID = 101` appears **twice**
- But `(SaleID, ProductID)` is **never duplicated**

---

5. Key takeaway (exam-friendly)

- **Primary Key** → uniquely identifies a row
- **Foreign Key** → links tables together
- **Composite Primary Key** → multiple columns together ensure uniqueness (very common in sales/order tables)

Categories → Products → Sales setup. I'll include **SELECT, JOIN, GROUP BY, and INSERT examples** that cover real-world scenarios.

---

1 Insert sample data

```
-- Insert Categories
INSERT INTO Categories (CategoryID, CategoryName) VALUES
(1, 'Electronics'),
(2, 'Clothing');

-- Insert Products
INSERT INTO Products (ProductID, ProductName, CategoryID, Price) VALUES
(101, 'Laptop', 1, 1000),
(102, 'Smartphone', 1, 700),
(103, 'T-Shirt', 2, 25),
(104, 'Jeans', 2, 50);

-- Insert Sales
INSERT INTO Sales (SaleID, ProductID, SaleDate, Quantity, TotalAmount) VALUES
(1, 101, '2026-02-01', 1, 1000),
(1, 103, '2026-02-01', 2, 50),
(2, 102, '2026-02-02', 1, 700),
(2, 104, '2026-02-02', 3, 150);
```

---

2 Basic SELECT queries

```
-- Get all products
SELECT * FROM Products;

-- Get all sales
SELECT * FROM Sales;

-- Get all categories
SELECT * FROM Categories;
```

---

3 JOIN queries (link tables)

```
-- Get sales with product names
SELECT s.SaleID, p.ProductName, s.Quantity, s.TotalAmount
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID;

-- Get sales with category names
SELECT s.SaleID, p.ProductName, c.CategoryName, s.Quantity, s.TotalAmount
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID;
```

---

4 Aggregation queries

```
-- Total sales amount per product
SELECT p.ProductName, SUM(s.TotalAmount) AS TotalSales
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
GROUP BY p.ProductName;

-- Total quantity sold per category
SELECT c.CategoryName, SUM(s.Quantity) AS TotalQuantity
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID
GROUP BY c.CategoryName;

-- Number of products sold in each sale
SELECT SaleID, COUNT(ProductID) AS ProductsInSale
FROM Sales
GROUP BY SaleID;
```

---

5 Advanced queries

```
-- Find highest selling product by revenue
SELECT p.ProductName, SUM(s.TotalAmount) AS Revenue
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY Revenue DESC
LIMIT 1;

-- List sales where quantity > 1
SELECT s.SaleID, p.ProductName, s.Quantity, s.TotalAmount
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
WHERE s.Quantity > 1;

-- Insert a new sale with multiple products
```

```
INSERT INTO Sales (SaleID, ProductID, SaleDate, Quantity, TotalAmount) VALUES
(3, 101, '2026-02-03', 2, 2000),
(3, 104, '2026-02-03', 1, 50);
```

A **mini SQL practice lab** for your **Categories → Products → Sales** database. This will cover **INSERT, SELECT, JOIN, UPDATE, DELETE, aggregation, and composite key exercises**. I'll give each one as a **challenge with the solution**, so you can try it and check yourself.

Mini SQL Practice Lab: Products, Categories, Sales

Setup Tables & Sample Data

```
-- Categories
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(100)
);

-- Products
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    CategoryID INT,
    Price DECIMAL(10,2),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

-- Sales (Composite Key: SaleID + ProductID)
CREATE TABLE Sales (
    SaleID INT,
    ProductID INT,
    SaleDate DATE,
    Quantity INT,
    TotalAmount DECIMAL(10,2),
    PRIMARY KEY (SaleID, ProductID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

-- Insert sample data
INSERT INTO Categories (CategoryID, CategoryName) VALUES
(1, 'Electronics'),
(2, 'Clothing');
```

```
INSERT INTO Products (ProductID, ProductName, CategoryID, Price) VALUES
(101, 'Laptop', 1, 1000),
(102, 'Smartphone', 1, 700),
(103, 'T-Shirt', 2, 25),
(104, 'Jeans', 2, 50);

INSERT INTO Sales (SaleID, ProductID, SaleDate, Quantity, TotalAmount) VALUES
(1, 101, '2026-02-01', 1, 1000),
(1, 103, '2026-02-01', 2, 50),
(2, 102, '2026-02-02', 1, 700),
(2, 104, '2026-02-02', 3, 150);
```

Practice Exercises

1 Simple SELECT

Challenge: List all products in the Electronics category.
Solution:

```
SELECT *
FROM Products
WHERE CategoryID = 1;
```

---

2 JOIN

Challenge: List all sales with product names and category names.
Solution:

```
SELECT s.SaleID, p.ProductName, c.CategoryName, s.Quantity, s.TotalAmount
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID;
```

---

3 Aggregation

Challenge: Find total revenue per category.
Solution:

```
SELECT c.CategoryName, SUM(s.TotalAmount) AS TotalRevenue
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID
GROUP BY c.CategoryName;
```

---

4 Composite Key Check

Challenge: Insert a new sale with multiple products (SaleID = 3).
Solution:

```
INSERT INTO Sales (SaleID, ProductID, SaleDate, Quantity, TotalAmount) VALUES
(3, 101, '2026-02-03', 2, 2000),
(3, 104, '2026-02-03', 1, 50);
```

---

5 UPDATE

Challenge: Increase the price of all clothing products by 10%.
Solution:

```
UPDATE Products
SET Price = Price * 1.10
WHERE CategoryID = 2;
```

---

6 DELETE

Challenge: Delete all sales of product 'T-Shirt'.
Solution:

```
DELETE FROM Sales
WHERE ProductID = 103;
```

---

7 Top-Selling Product

Challenge: Find the product with the highest total sales amount.
Solution:

```
SELECT p.ProductName, SUM(s.TotalAmount) AS Revenue
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY Revenue DESC
LIMIT 1;
```

---

8 Count Sales per SaleID

Challenge: Count how many different products are in each sale.
Solution:

```
SELECT SaleID, COUNT(ProductID) AS ProductCount
```

```
FROM Sales
GROUP BY SaleID;
```

---

9 LEFT JOIN Practice

Challenge: List all products and their sales (show products even if never sold).
Solution:

```
SELECT p.ProductName, s.SaleID, s.Quantity, s.TotalAmount
FROM Products p
LEFT JOIN Sales s ON p.ProductID = s.ProductID;
```

Complex Query

Challenge: Find the total quantity sold for each category along with total revenue, only for categories that sold more than 2 items.
Solution:

```
SELECT c.CategoryName, SUM(s.Quantity) AS TotalQuantity, SUM(s.TotalAmount)
AS TotalRevenue
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID
GROUP BY c.CategoryName
HAVING SUM(s.Quantity) > 2;
```