Activity: Writing Basic SQL Queries with Microsoft Copilot

```
SELECT
    ProductName,
    Category,
    Price,
    StockLevel
FROM
    Products;


SELECT
    ProductName,
    Category,
    Price,
    StockLevel
FROM
    Products
WHERE
    Category = 'Electronics'  -- Replace 'Electronics' with the desired category
ORDER BY
    Price ASC;  -- Sorting by Price in ascending order


SELECT
    ProductName,
    Category,
    Price,
    StockLevel
FROM
    Products
WHERE
```

```
   StockLevel <= 10  -- Adjust threshold as needed
ORDER BY
   Price ASC;  -- Sorting by Price in ascending order
```

Activity 2: Creating Complex SQL Queries with Microsoft Copilot

```
SELECT
   p.ProductName,
   s.SaleDate,
   st.StoreLocation,
   s.UnitsSold
FROM
   Sales s
JOIN
   Products p ON s.ProductID = p.ProductID
JOIN
   Stores st ON s.StoreID = st.StoreID
ORDER BY
   s.SaleDate DESC; -- Sorting by most recent sales first


SELECT
   p.ProductName,
   SUM(s.UnitsSold) AS TotalUnitsSold
FROM
   Sales s
JOIN
   Products p ON s.ProductID = p.ProductID
GROUP BY
   p.ProductName
ORDER BY
   TotalUnitsSold DESC; -- Sorting to show top-selling products first
```

```
SELECT
   sp.SupplierName,
   COUNT(*) AS DelayedDeliveries
FROM
   Deliveries d
JOIN
   Suppliers sp ON d.SupplierID = sp.SupplierID
WHERE
   d.DeliveryDate > d.ExpectedDeliveryDate  -- Finding delayed deliveries
GROUP BY
   sp.SupplierName
ORDER BY
   DelayedDeliveries DESC; -- Sorting by the highest delays first
```

Activity 3: Debugging and Optimizing SQL Queries with Microsoft Copilot

```
SELECT
   p.ProductName,
   s.SaleDate,
   st.StoreLocation,
   s.UnitsSold
FROM
   Sales s
LEFT JOIN
   Products p ON s.ProductID = p.ProductID
LEFT JOIN
   Stores st ON s.StoreID = st.StoreID
ORDER BY
   s.SaleDate DESC;
```

```
SELECT
   p.ProductName,
   SUM(s.UnitsSold) AS TotalUnitsSold
FROM
   Products p
JOIN
   Sales s ON p.ProductID = s.ProductID
GROUP BY
   p.ProductName
ORDER BY
   TotalUnitsSold DESC;
-- Index for filtering and joining Sales table
CREATE INDEX idx_sales_productid ON Sales (ProductID);
CREATE INDEX idx_sales_storeid ON Sales (StoreID);
CREATE INDEX idx_sales_saledate ON Sales (SaleDate);

-- Index for filtering by category in Products
CREATE INDEX idx_products_category ON Products (Category);

-- Index for joining Deliveries with Suppliers
CREATE INDEX idx_deliveries_supplierid ON Deliveries (SupplierID);

-- Index for checking delayed deliveries
CREATE INDEX idx_deliveries_dates ON Deliveries (DeliveryDate, ExpectedDeliveryDate);
SELECT
   p.ProductName,
   COALESCE(SUM(s.UnitsSold), 0) AS TotalUnitsSold
FROM
   Products p
```

```
LEFT JOIN
    Sales s ON p.ProductID = s.ProductID
GROUP BY
    p.ProductName
ORDER BY
    TotalUnitsSold DESC;
```

====================================

Here's a brief summary of how ChatGPT assisted in each step of developing the **SmartShop Inventory System**, based on your document:

Activity 1: Writing Basic SQL Queries with Microsoft Copilot

ChatGPT helped construct foundational queries to retrieve and filter inventory data from the Products table. It assisted in:

- Listing product details like name, category, price, and stock level.
- Filtering products by category and low stock levels.
- Sorting results by price for better visibility of cost-effective inventory.

Activity 2: Creating Complex SQL Queries with Microsoft Copilot

ChatGPT guided the creation of advanced queries to analyze sales trends and supplier performance. Specifically, it:

- Joined multiple tables (Sales, Products, Stores) to track product sales across locations.
- Aggregated sales data to identify top-selling products.
- Generated queries to analyze delayed deliveries by suppliers, offering insights into supply chain reliability.

Activity 3: Debugging and Optimizing SQL Queries with Microsoft Copilot

In this phase, ChatGPT helped refine and optimize SQL queries to enhance performance. This included:

- Modifying joins for data completeness (using LEFT JOIN for optional matches).
- Improving query reliability with functions like COALESCE to handle nulls.
- Creating appropriate indexes on frequently filtered and joined columns to improve query speed and scalability.
