# Database Design and Dataset Selection

To demonstrate the solution, we created a mock e-commerce database to simulate a real-world scenario. This database is representative of many industry applications that involve large volumes of customer and transactional data.

The database contains two primary tables:

1. Customers Table:  
- Stores information about customers, including their name, email, country, and account creation date.  
- Designed to simulate customer data for an online retail platform.

2. Orders Table:  
- Stores transaction details, including the associated customer, order amount, and order date.  
- Mimics real-world order data in an e-commerce system.

This structure allows us to test query performance for retrieving orders based on country and date ranges, operations common in analytics and reporting for businesses. Without indexing, such queries typically result in full table scans, leading to slow execution times. By implementing single-column and composite indexes, we were able to demonstrate significant improvements in query performance.

## SQL Implementation

To set up the database, we used the following script. It creates the database, defines the schema, and populates it with sample data to simulate realistic scenarios.

-- Step 1: Create the Database if it does not exist  
IF NOT EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name = 'EcommerceDB')  
BEGIN  
 CREATE DATABASE EcommerceDB;  
END;  
GO  
USE EcommerceDB;  
GO  
  
-- Step 2: Create the Customers Table  
CREATE TABLE Customers (  
 CustomerID INT IDENTITY(1,1) PRIMARY KEY, -- Auto-increment for unique Customer IDs  
 Name VARCHAR(50),  
 Email VARCHAR(50),  
 Country VARCHAR(50),  
 CreatedAt DATETIME  
);  
  
-- Step 3: Create the Orders Table  
CREATE TABLE Orders (  
 OrderID INT IDENTITY(1,1) PRIMARY KEY, -- Auto-increment for unique Order IDs  
 CustomerID INT,  
 OrderAmount DECIMAL(10, 2),  
 OrderDate DATETIME,  
 FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);  
  
-- Step 4: Populate the Customers Table with 1,000 rows  
SET NOCOUNT ON;  
  
DECLARE @i INT = 1;  
  
WHILE @i <= 1000  
BEGIN  
 INSERT INTO Customers (Name, Email, Country, CreatedAt)  
 VALUES (  
 CONCAT('Customer', @i),   
 CONCAT('customer', @i, '@example.com'),  
 CASE   
 WHEN @i % 5 = 0 THEN 'USA'  
 WHEN @i % 5 = 1 THEN 'Canada'  
 WHEN @i % 5 = 2 THEN 'UK'  
 WHEN @i % 5 = 3 THEN 'Australia'  
 ELSE 'Germany'  
 END,  
 DATEADD(DAY, @i, '2020-01-01')  
 );  
 SET @i = @i + 1;  
END;  
  
-- Step 5: Populate the Orders Table with 10,000 rows  
SET @i = 1;  
  
WHILE @i <= 10000  
BEGIN  
 INSERT INTO Orders (CustomerID, OrderAmount, OrderDate)  
 VALUES (  
 FLOOR(1 + (RAND() \* 1000)), -- Random CustomerID between 1 and 1000  
 ROUND(10 + (RAND() \* 490), 2), -- Random OrderAmount between 10 and 500  
 DATEADD(DAY, FLOOR(RAND() \* 1000), '2022-01-01') -- Random OrderDate  
 );  
 SET @i = @i + 1;  
END;

## Why This Database Design?

We chose this database design for the following reasons:

1. Real-World Relevance:  
- The schema reflects a typical e-commerce scenario with customers and their orders. Businesses often run queries like retrieving orders by country, date ranges, or specific customer details, making it a meaningful use case for query optimization.

2. Manageable Data Size:  
- The dataset includes 1,000 customers and 10,000 orders, which provides enough complexity to demonstrate the performance improvements from indexing without overloading the server or exceeding storage limits.

3. Focus on Query Optimization:  
- The structure allows us to simulate performance bottlenecks commonly encountered in industry applications, such as filtering, joins, and sorting.

4. Flexibility:  
- This mock dataset is easily customizable, allowing us to experiment with different query types and indexing strategies.

By using this database, we demonstrate the tangible benefits of indexing in a controlled environment, providing insights into its real-world applications.