Project Title: Database Design Online Bookstore

Objective:
To highlight my ability to design and manage databases, write SQL queries, move data between Excel and SQL Server, create reports, and automate workflows using Python.

---

1. Database Design and Optimization

Tables Created:

- **Customers**: Stores customer details.
- **Books**: Contains book info such as title, author, genre, and stock.
- **Orders**: Tracks customer orders.
- **Order_Items**: Holds order line items, book quantities, sale price, and profit.

Highlights:

- Used **Primary Keys** and **Foreign Keys** to maintain data integrity and optimize joins.
- Proper normalization to reduce redundancy and improve performance.

SQL Schema:

-- Create Customers Table

```
CREATE TABLE Customers (
    Customer_id INT PRIMARY KEY,
    name VARCHAR (100),
    email VARCHAR (100),
    phone VARCHAR (20),
    city VARCHAR (100),
    age INT
);
```

-- Create Books Table

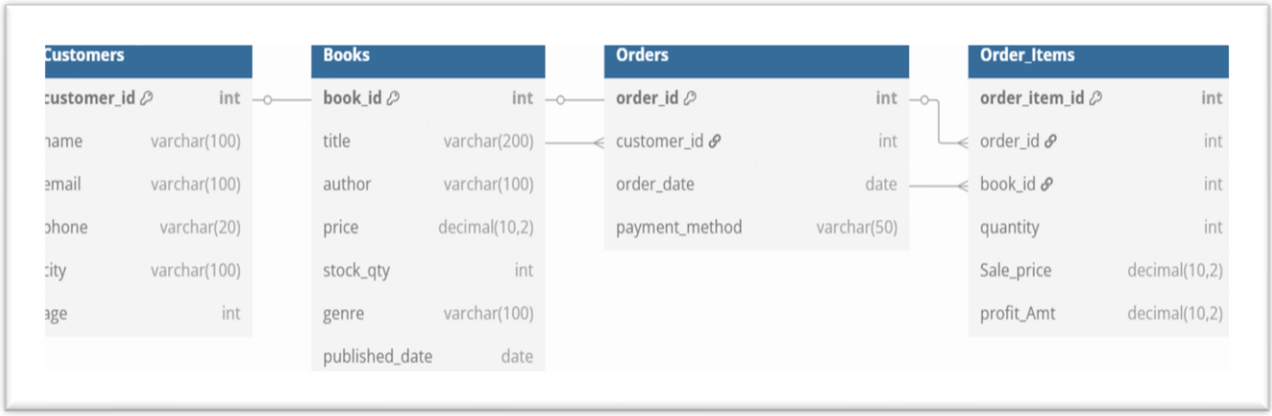
```
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR (200),
    author VARCHAR (100),
    price DECIMAL (10, 2),
    stock_qty INT,
    genre VARCHAR (100),
    published_date DATE
);
```

-- Create Orders Table

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    payment_method VARCHAR (50),
    CONSTRAINT FK_Orders_Customers FOREIGN KEY (customer_id) REFERENCES
Customers(customer_id));
```

-- Create Order_Items Table

```
CREATE TABLE Order_Items (
    order_item_id INT PRIMARY KEY,
    order_id INT,
    book_id INT,
    quantity INT,
    Sale_price DECIMAL (10, 2),
    profit_Amt DECIMAL (10, 2),
    CONSTRAINT FK_OrderItems_Orders FOREIGN KEY (order_id) REFERENCES
Orders(order_id),
    CONSTRAINT FK_OrderItems_Books FOREIGN KEY (book_id) REFERENCES Books(book_id)
);
```



---

2. SQL Queries / Stored Procedures

- Wrote SQL queries to retrieve order summaries, out-of-stock books, and customer purchases. Used joins, aggregations, and filtering to support reporting.

Key Queries:

-- 1. Get all customers

```
SELECT * FROM Customers;
```

-- 2. List all books that are in stock

```
SELECT * FROM Books WHERE stock_qty > 0;
```

-- 3. Get all orders placed after April 1, 2024

```
SELECT * FROM Orders WHERE order_date > '2024-04-01';
```

-- 4. Find total number of orders placed by each customer

```
SELECT customer_id, COUNT (*) AS total_orders
FROM Orders
GROUP BY customer_id;
```

-- 5. Show total sales value for each book

```
SELECT
book_id,
SUM (sale_price) AS total_sales
FROM Order_Items
GROUP BY book_id;
```

Stored Procedure:

-- Stored Procedure: Get all orders for a specific customer

```
CREATE PROCEDURE GetCustomerOrders
    @cust_id INT
AS
BEGIN
    SELECT
        Orders.customer_id,
        Orders.order_id,
        Orders.order_date,
        Books.title,
        Order_Items.quantity,
        Order_Items.sale_price
    FROM Orders
    JOIN Order_Items ON Orders.order_id = Order_Items.order_id
    JOIN Books ON Order_Items.book_id = Books.book_id
    WHERE Orders.customer_id = @cust_id;
END;
```

-- Execute example

```
EXEC GetCustomerOrders @cust_id = 1;
```

---

3. Python Scripts for Data Migration, ETL & Automation

- Used Python scripts to move and update data between Excel and SQL Server.
- One script reads Excel, cleans the data, and loads it into the database.
- Another script reads the database and updates the Excel file.
- This setup keeps both Excel and SQL Server in sync automatically

Script 1: Excel to SQL Server

- Reads Excel data with 4 sheets
- Cleans data (removes duplicates)
- Loads data into SQL Server using to_sql()

```
import pandas as pd
from sqlalchemy import create_engine
import urllib

params = urllib.parse.quote_plus(
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=DESKTOP-509Q3SC;"
    "DATABASE=Database_Design_Online_Bookstore;"
    "Trusted_Connection=yes;"
)
engine = create_engine("mssql+pyodbc:///?odbc_connect=%s" % params)
```

```
excel_path = 'bookstore_data.xlsx'
customers_df = pd.read_excel(excel_path, sheet_name='Customers')
books_df = pd.read_excel(excel_path, sheet_name='Books')
orders_df = pd.read_excel(excel_path, sheet_name='Orders')
order_items_df = pd.read_excel(excel_path, sheet_name='Order_Items')

customers_df = customers_df.drop_duplicates(subset=['customer_id'])
books_df = books_df.drop_duplicates(subset=['book_id'])
orders_df = orders_df.drop_duplicates(subset=['order_id'])
order_items_df = order_items_df.drop_duplicates(subset=['order_item_id'])

customers_df.to_sql('Customers', con=engine, if_exists='replace',
index=False)
books_df.to_sql('Books', con=engine, if_exists='replace', index=False)
orders_df.to_sql('Orders', con=engine, if_exists='replace', index=False)
order_items_df.to_sql('Order_Items', con=engine, if_exists='replace',
index=False)

print("✅ Data successfully inserted into SQL Server!")
```

Script 2: SQL Server to Excel

- Reads data from SQL tables
- Writes data into an Excel file with 4 sheets

```
import pandas as pd
from sqlalchemy import create_engine
import urllib

params = urllib.parse.quote_plus(
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=DESKTOP-509Q3SC;"
    "DATABASE=Database_Design_Online_Bookstore;"
    "Trusted_Connection=yes;"
)
engine = create_engine("mssql+pyodbc:///?odbc_connect=%s" % params)

customers_df = pd.read_sql("SELECT * FROM Customers", engine)
books_df = pd.read_sql("SELECT * FROM Books", engine)
orders_df = pd.read_sql("SELECT * FROM Orders", engine)
order_items_df = pd.read_sql("SELECT * FROM Order_Items", engine)

with pd.ExcelWriter("bookstore_data.xlsx", engine='openpyxl', mode='w') as
writer:
    customers_df.to_excel(writer, sheet_name="Customers", index=False)
    books_df.to_excel(writer, sheet_name="Books", index=False)
    orders_df.to_excel(writer, sheet_name="Orders", index=False)
    order_items_df.to_excel(writer, sheet_name="Order_Items", index=False)

print("✅ Data exported from SQL Server to Excel!")
```

Summary:

- These Python scripts automate ETL, data migration, and sync between Excel and SQL Server.
- Script 1 loads cleaned Excel data into the database.
- Script 2 exports database data back into Excel.
- If Excel is updated ➔ run Script 1 to sync with SQL and If SQL is updated ➔ run Script 2 to sync with Excel

---

4. Reporting & Visualization (Power BI)

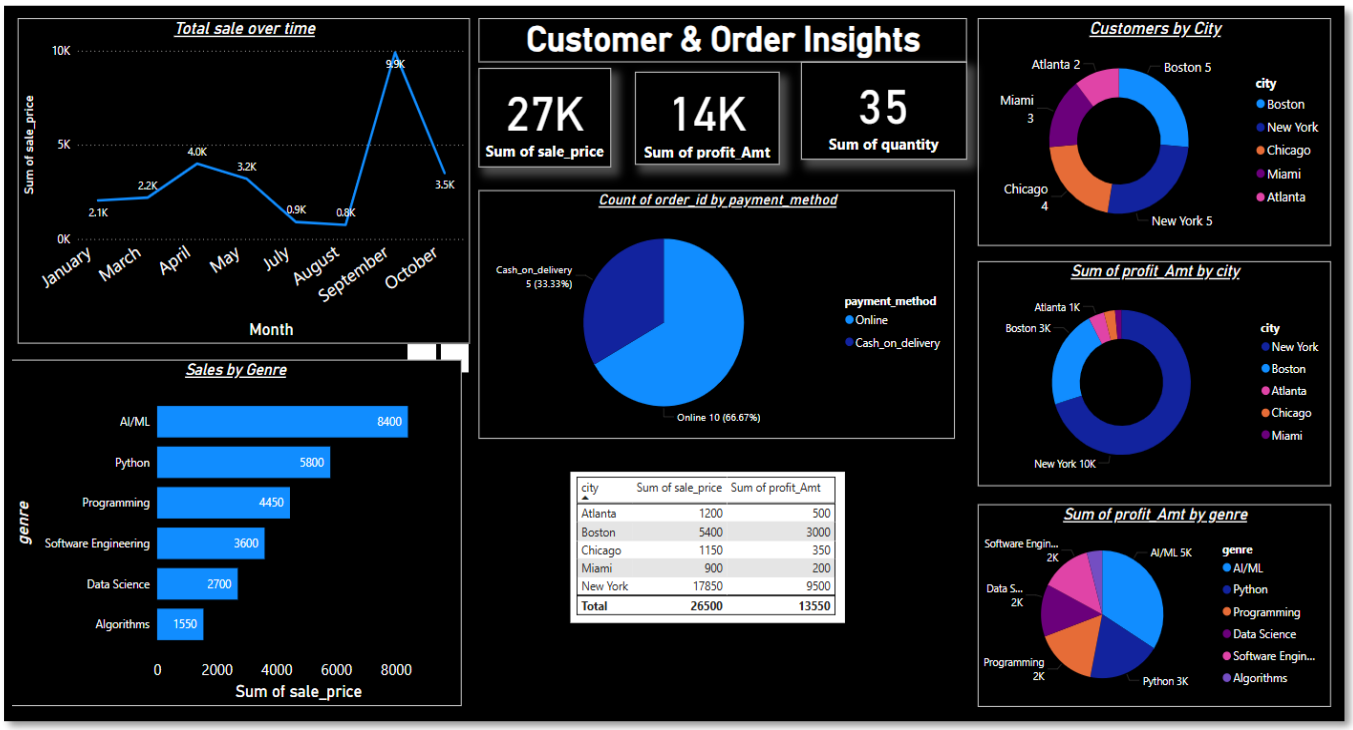
Dashboard Includes:

- Pie chart: Book stock by genre
- Line chart: Sales over time
- Table: Orders and profit
- Card: Total customers
- Donut: Orders by payment method

Features:

- Connected to SQL Server using On-Premises Gateway
- Enabled **Scheduled Refresh** to auto-update reports when database changes

Screenshot:



city	Sum of sale_price	Sum of profit_Amt
Atlanta	1200	500
Boston	5400	3000
Chicago	1150	350
Miami	900	200
New York	17850	9500
Total	**26500**	**13550**

✅ Project Summary

- This project demonstrates a complete data-driven solution, starting from relational database design and development, to building automated data pipelines and visualizing insights with Power BI. By integrating SQL Server, Python, Excel, and Power BI, the system supports efficient data management, automated ETL, and interactive reporting. This hands-on project helped reinforce real-world database concepts and workflow automation practices.
