# Library Database Management System

### Individual Project Report

**Submitted by:** Blessing James
**Course:** AI Engineering

## 1. Introduction

The **Library Database Management System** is a structured database solution developed to manage library operations efficiently. It handles information about **authors, books, members, staff, departments, borrow history, and book orders**.

The project integrates **PostgreSQL** for database management, **Python (Pandas & SQLAlchemy)** for data querying and analysis, and **Matplotlib/Seaborn** for data visualization.

This report documents the database design process, query implementation, insights derived, and challenges encountered.

## 2. Database Description

The `library_db` database consists of **seven interrelated tables**:

1. **Authors** – Stores author details such as name, number of books written, and country.
2. **Departments** – Contains information about library departments.
3. **LibraryStaff** – Keeps staff data, their department, and contact information.
4. **Members** – Records member information, membership type, and join date.
5. **Books** – Stores book data such as title, author, genre, publication year, and copies.
6. **BorrowHistory** – Tracks books borrowed, due dates, and return dates.
7. **BookOrders** – Maintains supplier orders, costs, and fulfillment status.

Each table is linked using **primary and foreign keys** to maintain referential integrity. The database supports **CRUD** operations, complex queries, and analytical reporting.

# 3. Methodology

## 3.1 Database Creation

The database was created using PostgreSQL with the command:

```
CREATE DATABASE library_db;
```

- Tables were defined using SQL scripts with appropriate **data types, primary keys, and foreign keys**.
- Constraints such as `NOT NULL` were applied to ensure data validity.

## 3.2 Data Loading

- Sample data was manually inserted into each table using the `INSERT INTO` statements.
- At least 50 records were added to each table to ensure realistic data representation.

## 3.3 Query Implementation

- SQL queries were executed to extract information for both basic and advanced analysis.
- Queries covered book listings, borrowing records, membership data, staff analysis, and revenue summaries.

## 3.4 Python Integration

- A PostgreSQL connection was established using **SQLAlchemy**.
- Data from the database was fetched into **Pandas DataFrames** for analysis.
- SQL queries were translated into Pandas equivalents for direct computation in Python.

## 3.5 Visualization

- Data visualization was performed using **Matplotlib** and **Seaborn**.
- Charts were created to represent data insights visually and support decision-making.

# 4. SQL Query Summary

Below is a summary of key SQL queries implemented:

| Query No. | Description | Type |
|---|---|---|
| Q1 | List books published after 2015 with author names | Basic |
| Q3 | Total number of books per author | Basic |
| Q5 | List staff working in the Circulation department | Basic |
| Q7 | Top 5 most borrowed books | Intermediate |
| Q8 | Members who have never borrowed a book | Intermediate |
| Q10 | Overdue books borrowed more than 30 days ago | Intermediate |
| Q11 | Staff count and average tenure by department | Advanced |
| Q12 | Monthly borrowing trends for the past year | Advanced |
| Q13 | Authors whose books were borrowed more than 10 times | Advanced |
| Q15 | Inactive Premium members (no borrow in last 6 months) | Advanced |

---

# 5. Python and Pandas Integration

## Connection Setup

```
# importing necessary libraries
import pandas as pd
from sqlalchemy import create_engine
from urllib.parse import quote_plus
import matplotlib.pyplot as plt

# Configuring connection to `library_db` database
USER = "postgres"
PASS = quote_plus("@user56")
HOST = "localhost"
PORT = "5432"
DB = "library_db"

engine = create_engine(f'postgresql://{USER}:{PASS}@{HOST}:{PORT}/{DB}')
```

### Example Pandas Query (Book Count per Author)

```
books = pd.read_sql("SELECT * FROM Books", engine)
authors = pd.read_sql("SELECT * FROM Authors", engine)

# selecting the columns to be displayed as a table
author_books = authors[["author_id","author_name","number_of_books_written"]]

# sorting by number_of_books_written in descending order
no_of_books_written = author_books.sort_values(by="number_of_books_written",
ascending=False)
no_of_books_written
```

This approach demonstrates how SQL queries can be translated into Pandas operations for flexible data analysis within Python.

# 6. Data Visualization

The following visualizations were created:

1. **Bar Chart – Top 10 Authors by Books Written**
   Shows authors with the highest number of publications.

2. **Pie Chart – Member Distribution by Membership Type**
   Illustrates how many members fall into Basic, Premium, Student, and Senior categories.

3. **Line Chart – Monthly Borrowing Trends**
   Displays changes in the number of borrowed books over a 12-month period.

4. **Horizontal Bar Chart – Available Copies by Genre**
   Visualizes the availability of books across different genres.

5. **Stacked Bar Chart/Heatmap – Orders by Fulfillment Status and Supplier**
   Compares the order fulfillment rates across suppliers.

Each chart was properly labeled with titles, legends, and color distinctions for clarity.

# 7. Insights and Findings

- **Premium members** represent the majority of active borrowers.
- Certain **authors consistently produce high-demand books**.
- Borrowing activity peaks in specific months (possibly academic seasons).
- A few **suppliers account for most of the total order revenue**.

These insights demonstrate how structured database design and SQL/Pandas analysis can inform better management decisions.

# 8. Challenges and Solutions

| Challenge | Solution |
|---|---|
| Establishing database connection from VSCode | Installed PostgreSQL extension and configured host credentials correctly. |
| Maintaining foreign key relationships | Ensured data insertion order and integrity using `REFERENCES` constraints. |
| Translating SQL to Pandas queries | Used `groupby()`, `merge()`, and filtering to replicate SQL logic. |
| Generating accurate charts | Verified data consistency before plotting and added proper labels for readability. |

# 9. SQL vs Pandas Comparison

| Aspect | SQL | Pandas |
|---|---|---|
| Syntax | Declarative, uses SELECT statements | Procedural, uses functions and chaining |
| Data Source | Operates directly on database | Works on in-memory DataFrames |
| Speed | Faster for large datasets in databases | Faster for smaller datasets in memory |
| Flexibility | Great for joins and filters | Ideal for complex calculations and visualizations |

| | | |
|---|---|---|
| Visualization | Requires external tools | Direct integration with Matplotlib and Seaborn |

Both tools complement each other—SQL for structured data retrieval and Pandas for in-depth analysis and visualization.

# 10. Conclusion

The Library Database Management System successfully demonstrates:

- Proper relational database design.
- Efficient data management using SQL.
- Seamless database connectivity and analysis in Python.
- Visual storytelling through meaningful data visualizations.

This project reinforced key database programming concepts and showcased how SQL and Python can be integrated to produce actionable insights from data.

# 11. References


- PostgreSQL Documentation
- Pandas Documentation
- SQLAlchemy Documentation
- Matplotlib Gallery