# SQL ASSIGNMENT

**NAME: - DILPREET SINGH MAHAL** **SUBJECT: - DATA MINING AND DICOVERY**

**STUDENT ID: - 22014268**

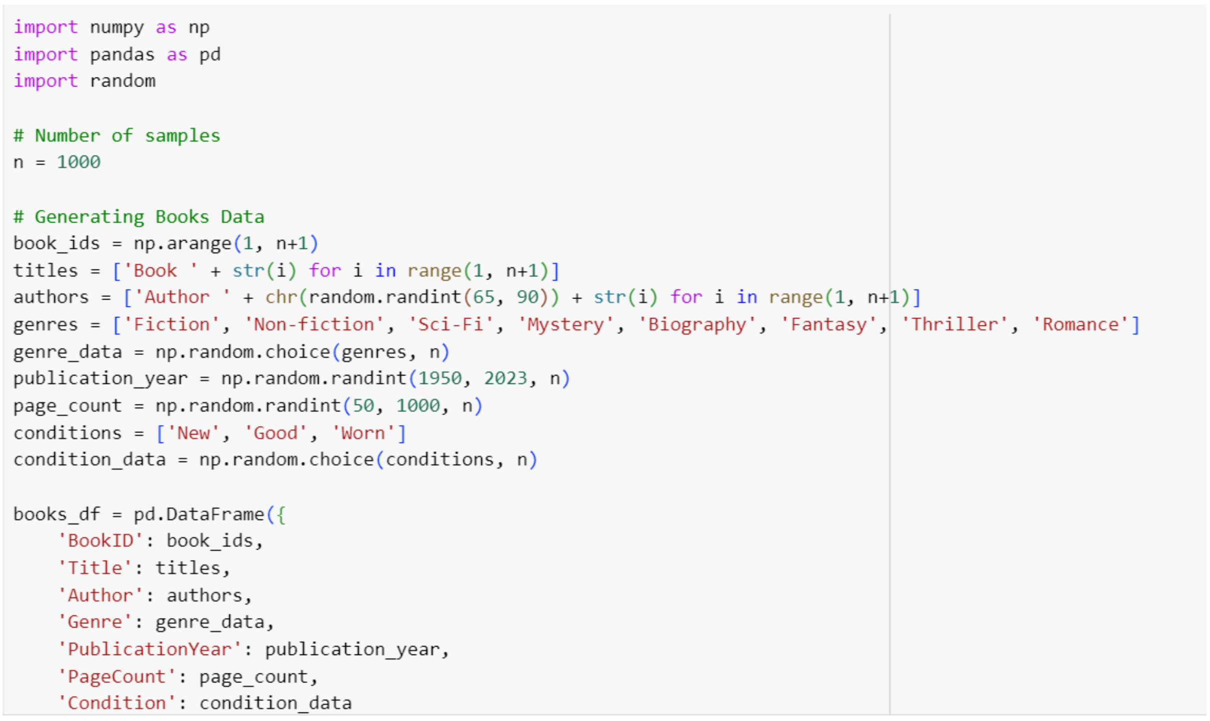
## INTRODUCTION

This report outlines the process of creating a simulated database for a library system. It includes details on generating random data for books, members, and loans, as well as the schema of the database. The database is intended for testing database functionalities.

## DATABASE GENERATION

The Python script used for the generation of database is vital for this report. It makes use of number of libraries such as: NumPy, Pandas, along with Python's built-in **'random'** module, to create structured data.

### Books Data

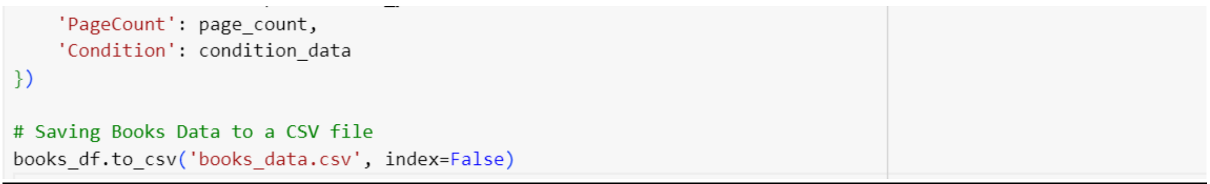


```python
import numpy as np
import pandas as pd
import random

# Number of samples
n = 1000

# Generating Books Data
book_ids = np.arange(1, n+1)
titles = ['Book ' + str(i) for i in range(1, n+1)]
authors = ['Author ' + chr(random.randint(65, 90)) + str(i) for i in range(1, n+1)]
genres = ['Fiction', 'Non-fiction', 'Sci-Fi', 'Mystery', 'Biography', 'Fantasy', 'Thriller', 'Romance']
genre_data = np.random.choice(genres, n)
publication_year = np.random.randint(1950, 2023, n)
page_count = np.random.randint(50, 1000, n)
conditions = ['New', 'Good', 'Worn']
condition_data = np.random.choice(conditions, n)

books_df = pd.DataFrame({
    'BookID': book_ids,
    'Title': titles,
    'Author': authors,
    'Genre': genre_data,
    'PublicationYear': publication_year,
    'PageCount': page_count,
    'Condition': condition_data
```

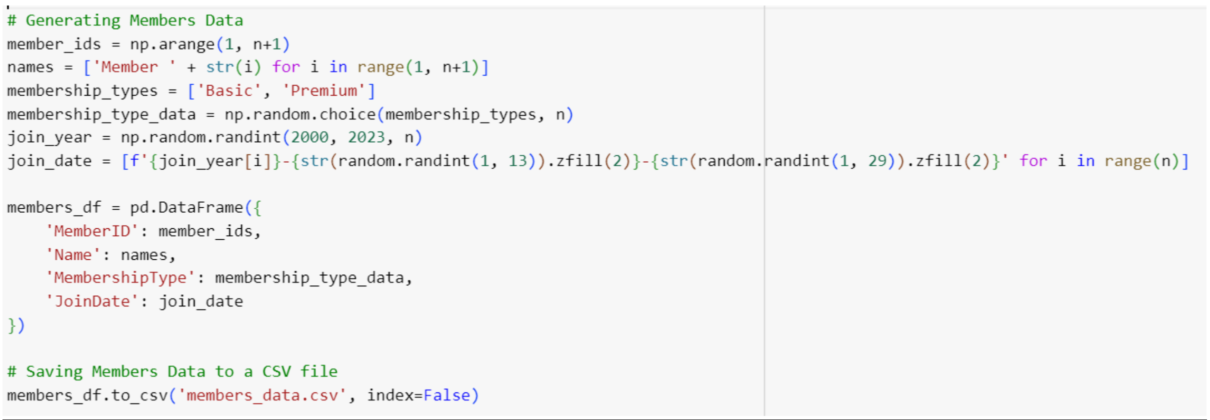


```
    'PageCount': page_count,
    'Condition': condition_data
})

# Saving Books Data to a CSV file
books_df.to_csv('books_data.csv', index=False)
```

- **Objective**: Create a randomized list of 1,000 books.
- **Attributes**:
  - **BookID**: Unique identifier.
  - **Title**: Randomly generated titles.
  - **Author**: Fictional author names.
  - **Genre**: Random selection from genres like Fiction, Non-fiction, etc.
  - **PublicationYear**: Random years ranging from 1950 to 2022.
  - **PageCount**: Randomly assigned page counts.
  - **Condition**: Random conditions (New, Good, Worn).
- **Method**: Utilized Python's random and NumPy libraries for data generation.

## Members Data



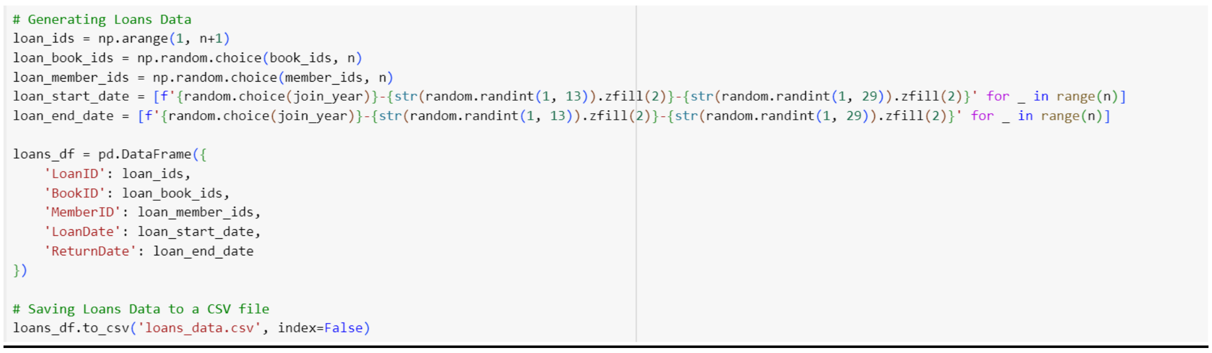
```
# Generating Members Data
member_ids = np.arange(1, n+1)
names = ['Member ' + str(i) for i in range(1, n+1)]
membership_types = ['Basic', 'Premium']
membership_type_data = np.random.choice(membership_types, n)
join_year = np.random.randint(2000, 2023, n)
join_date = [f'{join_year[i]}-{str(random.randint(1, 13)).zfill(2)}-{str(random.randint(1, 29)).zfill(2)}' for i in range(n)]

members_df = pd.DataFrame({
    'MemberID': member_ids,
    'Name': names,
    'MembershipType': membership_type_data,
    'JoinDate': join_date
})

# Saving Members Data to a CSV file
members_df.to_csv('members_data.csv', index=False)
```

- **Objective**: Generate a list of 1,000 library members.
- **Attributes**:
  - **MemberID**: Unique identifier.
  - **Name**: Generated names.
  - **MembershipType**: Types include Basic and Premium, randomly assigned.
  - **JoinDate**: Random dates representing when members joined.
- **Method**: Applied Python's random methods for diverse data.

## Loans Data



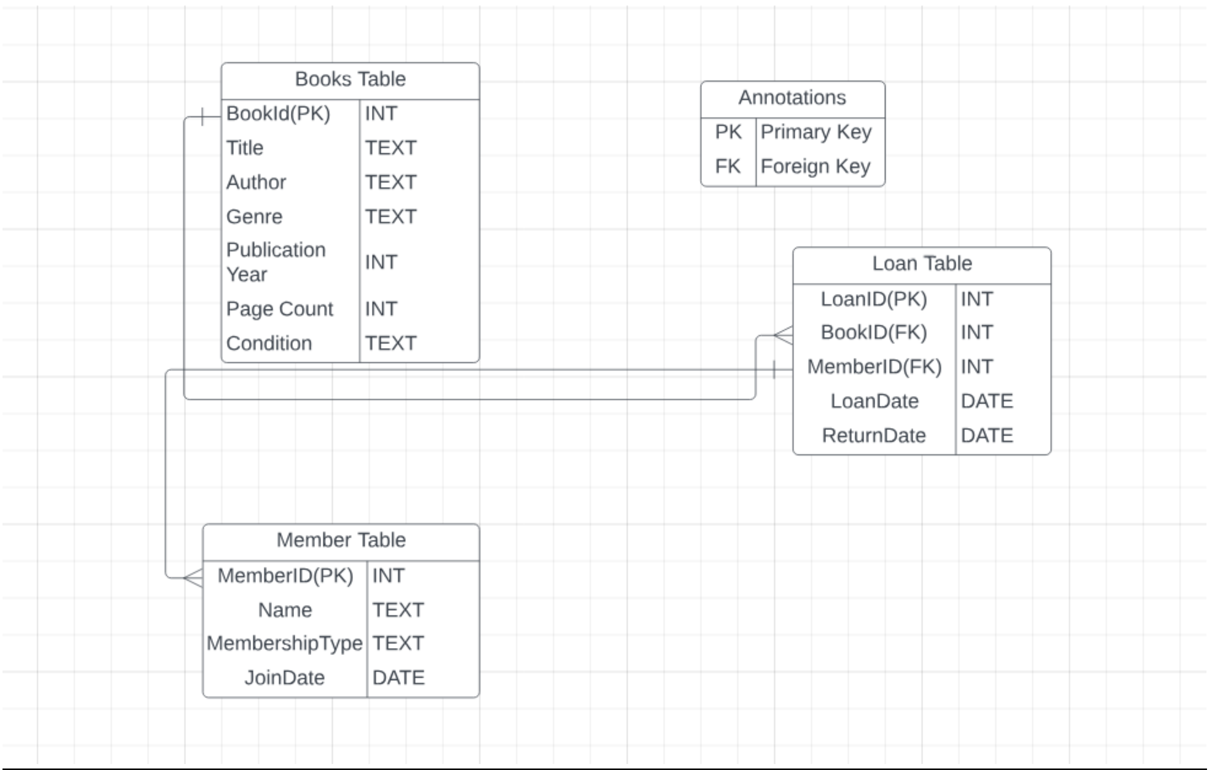
```python
# Generating Loans Data
loan_ids = np.arange(1, n+1)
loan_book_ids = np.random.choice(book_ids, n)
loan_member_ids = np.random.choice(member_ids, n)
loan_start_date = [f'{random.choice(join_year)}-{str(random.randint(1, 13)).zfill(2)}-{str(random.randint(1, 29)).zfill(2)}' for _ in range(n)]
loan_end_date = [f'{random.choice(join_year)}-{str(random.randint(1, 13)).zfill(2)}-{str(random.randint(1, 29)).zfill(2)}' for _ in range(n)]

loans_df = pd.DataFrame({
    'LoanID': loan_ids,
    'BookID': loan_book_ids,
    'MemberID': loan_member_ids,
    'LoanDate': loan_start_date,
    'ReturnDate': loan_end_date
})

# Saving Loans Data to a CSV file
loans_df.to_csv('loans_data.csv', index=False)
```

- **Objective**: Simulate 1,000 loan records.
- **Attributes**:
  - **LoanID**: Unique identifier.
  - **BookID**: References the **BookID** from Books table.
  - **MemberID**: References the **MemberID** from Members table.
  - **LoanDate** and **ReturnDate**: Random dates indicating the duration of each loan.
- **Method**: Random dates generated to reflect various loan periods.

## DATABASE SCHEMA



| Books Table | |
|---|---|
| BookId(PK) | INT |
| Title | TEXT |
| Author | TEXT |
| Genre | TEXT |
| Publication Year | INT |
| Page Count | INT |
| Condition | TEXT |

| Annotations | |
|---|---|
| PK | Primary Key |
| FK | Foreign Key |

| Loan Table | |
|---|---|
| LoanID(PK) | INT |
| BookID(FK) | INT |
| MemberID(FK) | INT |
| LoanDate | DATE |
| ReturnDate | DATE |

| Member Table | |
|---|---|
| MemberID(PK) | INT |
| Name | TEXT |
| MembershipType | TEXT |
| JoinDate | DATE |

In this section, we will explore the structure of library system database. Database consists of three key tables: **Books**, **Members**, and **Loans**. Each of these tables uses various data types – nominal, ordinal, interval and ratio data - to model real-world characteristics effectively.

- **Nominal Data:**
  - Genre (in Books table) classifies books into various genres like Fiction, Non-fiction, etc.
  - MembershipType (in Members table) differentiates members by types such as Basic, Premium, etc.
- **Ordinal Data:**
  - Condition (in Books table) represents the condition of books, categorized as New, Good, or Worn.
- **Interval Data:**
  - PublicationYear (in Books table) represents the year a book was published. We can measure the difference between years, but there is no absolute zero point in this scale.
- **Ratio Data :**
  - PageCount (in Books table) indicates the number of pages in a book, which is a quantifiable figure and can be compared proportionally.

This schema effectively explains the important features of library management, such as inventory (books), users (members), and transactions (loans). The nature of the database, with **BookID** and **MemberID** acting as foreign keys in the Loans table, allows for efficient querying and data integrity.

## JUSTIFICATION FOR DIFFERENT TABLES

**Books Table**:

- **Purpose**: Stores data about each book, including title, writer, class, and so on.
- **Justification**: Having a dedicated table for books allows easy management of the library's books inventory, separate from membership information or loan transactions.

**Members Table**:

- **Purpose**: Contains personal and membership details of each library member.
- **Justification**: Data integrity and privacy are guaranteed by separating member data. It makes it easier to update membership details without interfering with loan or book records.

**Loans Table**:

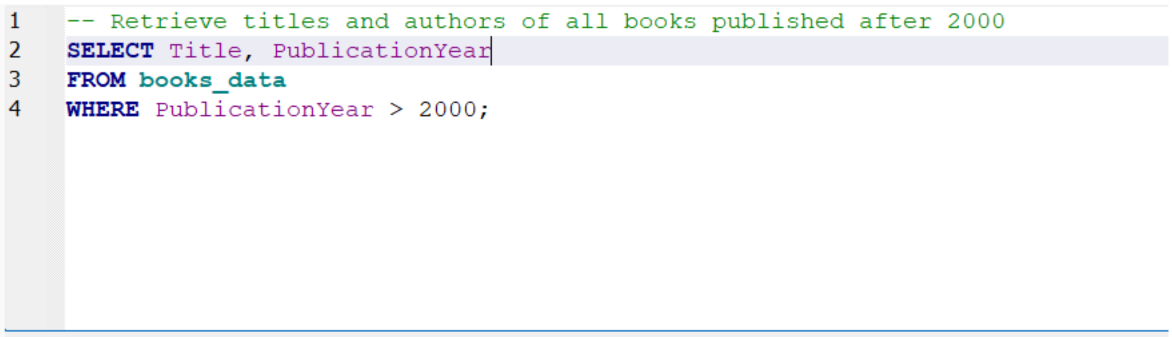
- **Purpose**: Monitors the borrowing details of books by members.
- **Justification**: We can keep track of which member has borrowed which book and when thanks to this table, which serves as a bridge between books and members. It makes sense to keep it apart from the Books and Members tables to avoid duplication and preserve clarity.

## ETHICAL DISCUSSION

- **Data Privacy**: Make sure the database doesn't use actual personal information. Anonymization and appropriate consent should be used if actual data is needed.
- **Data Security**: Talk about how crucial it is to protect the database from illegal access, particularly in light of the personal information held by members.
- **Accuracy and Integrity**: It's critical that users benefit from accurate and current information kept in a library system.
- **Responsible Use**: Stress the need of utilizing the database in an ethical manner, protecting user privacy, and making sure the information isn't exploited for immoral ends.

## SQL QUERIES

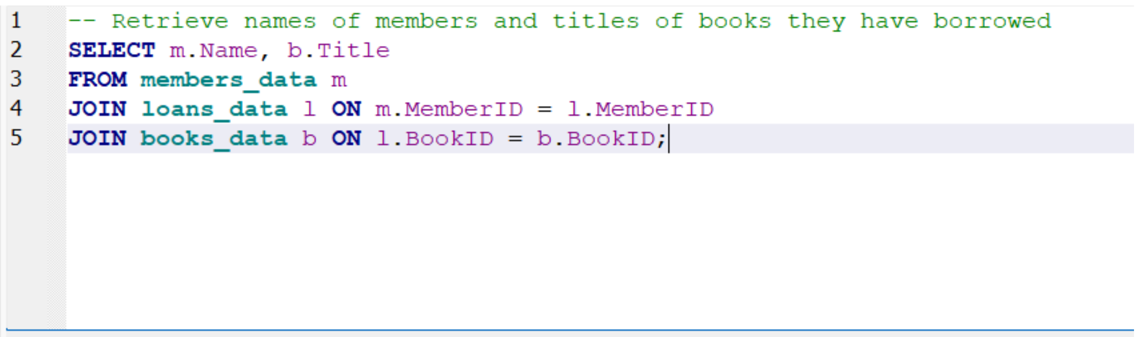
- **SELECTING SPECIFIC COLUMNS**



```sql
-- Retrieve titles and authors of all books published after 2000
SELECT Title, PublicationYear
FROM books_data
WHERE PublicationYear > 2000;
```

| | Title | PublicationYear |
|---|---|---|
| 1 | Book 3 | 2005 |
| 2 | Book 4 | 2009 |
| 3 | Book 10 | 2002 |
| 4 | Book 14 | 2005 |

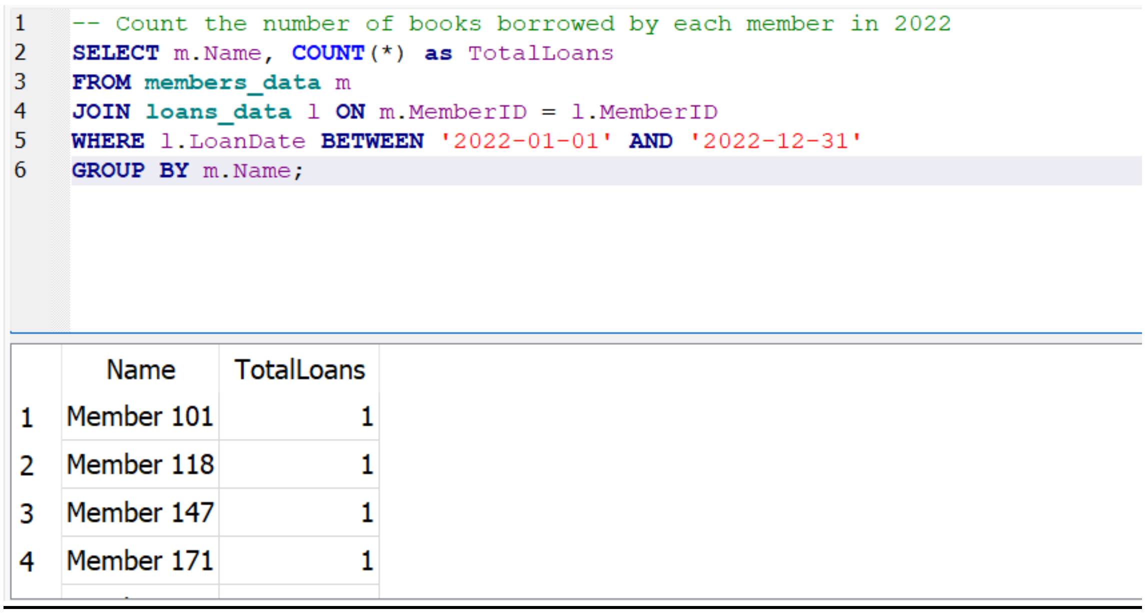
- **JOINING COLUMNS**



```sql
-- Retrieve names of members and titles of books they have borrowed
SELECT m.Name, b.Title
FROM members_data m
JOIN loans_data l ON m.MemberID = l.MemberID
JOIN books_data b ON l.BookID = b.BookID;
```

| | Name | Title |
|---|---|---|
| 1 | Member 3 | Book 428 |
| 2 | Member 4 | Book 207 |
| 3 | Member 4 | Book 681 |
| 4 | Member 5 | Book 169 |

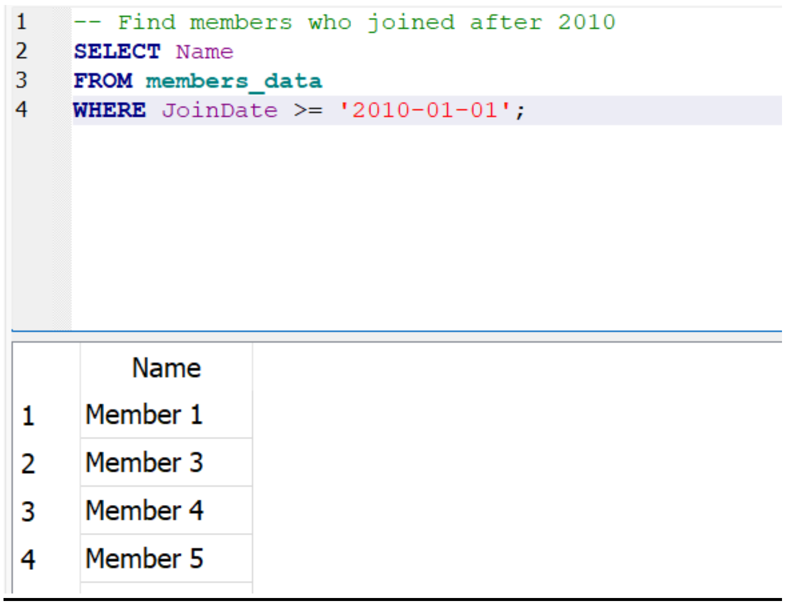
- **FILTERING AND AGGREGATING DATA**



```sql
-- Count the number of books borrowed by each member in 2022
SELECT m.Name, COUNT(*) as TotalLoans
FROM members_data m
JOIN loans_data l ON m.MemberID = l.MemberID
WHERE l.LoanDate BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY m.Name;
```

|   | Name | TotalLoans |
|---|---|---|
| 1 | Member 101 | 1 |
| 2 | Member 118 | 1 |
| 3 | Member 147 | 1 |
| 4 | Member 171 | 1 |

- **DATE RANGE FILTERING**



```sql
-- Find members who joined after 2010
SELECT Name
FROM members_data
WHERE JoinDate >= '2010-01-01';
```

|   | Name |
|---|---|
| 1 | Member 1 |
| 2 | Member 3 |
| 3 | Member 4 |
| 4 | Member 5 |

These queries show how to use a variety of methods, including inner joins, conditional filtering, COUNT() data aggregation, and working with a variety of data types, including strings (Name, Title, Genre), integers (PageCount), and dates (JoinDate, LoanDate). The purpose of every query is to retrieve significant data from the relational database that is made up of the books_data, members_data, and loans_data tables.

## CONCLUSION

This report discusses the design and implementation of a LIBRARY database, emphasizing data generation, schema design, ethical considerations, and sample queries, among other important topics. The artificial data was created to fill the database, putting data privacy first and making sure the data's origin was not external. Tables pertaining to books, memberships, and loans were included in the schema. To establish relationships, columns were assigned the appropriate data types and keys. Privacy and data protection were prioritized, and data anonymization was used. strategies used. Examples of SQL queries were used to show the database's features for organizing the books, book loans and memberships in the library.