**Assignment 1 Part C -** SE 02 Big Data & Analytics B

**Presented by:**

Kabuniang Buhawi Monjardin

Ezra Runnath

Fatih Selim Salihoglu

Shukurolloh Abdulboqiyev

**To:**

Instructor Ali Vaisifard

## Prerequisites:

The assignment is contained in a GitHub repository with the following link:
https://github.com/Dowiw/UE_library.git

This allows for more thorough team collaboration and version control of the assignment. It also allowed access throughout systems as an open repository.

**To run:**

- One must have **Git** to clone the repository.
- Also have **postgres** and **python3** to have the python script installed into your server and database.
- One must consider postgres user, postgres password and postgres initial database before running the python code.
- Recommended but not required: one must try to have **pgAdmin 4** to have a **better visualization of the data** and **querying into it.**

**Assignment 1 Part C -** SE 02 Big Data & Analytics B

## Part 1 - Creation Script

The creation script code is housed in the builder python file. This has initial variables that are defined by the user to access the currently available database and postgres user.

```python
import csv
import psycopg2
from psycopg2 import sql

# Configuration
HOST = "localhost"
USER = "postgres"
PASSWORD = "somerandompassword"
PORT = "5432"
initial_db = "postgres" # Initial Database
new_db_name = "library" # Database to create

# Connect to initial database first ('postgres')
print("Connecting to initial database...")
connection = psycopg2.connect(
    dbname = initial_db,
    user = USER,
    password = PASSWORD,
    host = HOST,
    port = PORT
)
# Enable for database creations without transactions
connection.autocommit = True
```

The user must input the **correct host, user, password, port** and **initial_db** to ensure that no errors will occur.

Next, the script creates a new database called “library” from new_db_name.

```python
# First sql module to create database
print(f"Creating new database '{new_db_name}'")
cursor.execute(sql.SQL("CREATE DATABASE {}").format(sql.Identifier(new_db_name)))
print(f"Database '{new_db_name}' created successfully.")

# Initially connect to new db using root
print("Connecting to new database...")
new_connection = psycopg2.connect(
    dbname = new_db_name,
    user = USER,
    password = PASSWORD,
    host = HOST,
    port = PORT
)
new_connection.autocommit = True
cursor = new_connection.cursor()
print("Connected!")
```

Next, tables are created using the execution method within the sql import which executes a query based on the string passed.

```
46  # CREATE ALL TABLES
47  print("Creating tables... :")
48
49  # ID Type table
50  print("Creating id_type_code table... ")
51  cursor.execute("""
52  CREATE TABLE id_type_code (
53      id_type_code INT PRIMARY KEY,
54      id_type_name TEXT NOT NULL
55  )
56  """)
57
```

Then, a query to fill up the tables is executed.

```
# FILL THE TABLES UP
print("Filling tables up with relevant data from csv files... ")

# First filling up tables without foreign keys
print("Filling tables with no foreign keys...")

# Shift table
print("Filling shift table (independent)...")
cursor.execute("""
INSERT INTO shift (shift_id, shift_name, shift_start, shift_end)
    VALUES (1, 'Morning', '08:00:00', '12:00:00'),
    (2, 'Afternoon', '12:00:00', '16:00:00'),
    (3, 'Evening', '16:00:00', '20:00:00'),
    (4, 'Night', '18:00:00', '22:00:00')
""")
```

These are initial values that are tables with no foreign keys.

```
187 print("Filling tables with foreign keys...")
188
189 # Staff table
190 print("Filling staff table...")
191 with open('staff.csv', 'r') as f:
192     reader = csv.reader(f)
193     next(reader)
194     for row in reader:
195         cursor.execute(
196             """INSERT INTO staff (staff_id, staff_name, shift_id)
197             VALUES (%s, %s, %s)
198             ON CONFLICT (staff_id) DO UPDATE
199             SET staff_name = EXCLUDED.staff_name,
200             shift_id = EXCLUDED.shift_id;""",
201             row
202         )
203
```

The code above uses open to read through the csv files and execute them as strings.

```
314     ....)
315
316 print("Done filling tables!")
317 print("Builder script done! Use the library database for queries!")
```

Finally, the finishing statement is printed out showing successful running.

**Part 2 - Sample Data**

The sample data is clonable within the GitHub repository inside the directory “csv_data”.
https://github.com/Dowiw/UE_library.git

**Part 3 - SQL Queries and Results**

**Query 1:**

Query | Query History

```sql
-- 1. Count of books per Author that authored three books and more, and their loan count by copy (COUNT, GROUP BY, HAVING)
SELECT a.author_name, COUNT(DISTINCT b.book_id) AS book_count, COUNT(l.loan_id) AS loan_count
FROM author a
JOIN book b ON a.author_id = b.author_id
JOIN "copy" c ON b.book_id = c.book_id
LEFT JOIN loan l ON c.copy_id = l.copy_id -- This LEFT JOIN for NULL books (no loans)
GROUP BY a.author_name
HAVING COUNT(DISTINCT b.book_id) > 3 -- Select only authors with books more than 3
ORDER BY book_count;
```

Data Output | Messages | Notifications

Showing rows: 1 to 8 | Page No: 1 of 1

| | author_name text | book_count bigint | loan_count bigint |
|---|---|---|---|
| 1 | Tyler Hernandez | 4 | 2 |
| 2 | Jessica Taylor | 4 | 12 |
| 3 | Madison Brooks | 4 | 8 |
| 4 | Sophia Morris | 4 | 8 |
| 5 | Alex Garcia | 7 | 12 |
| 6 | Hannah Moore | 7 | 21 |
| 7 | Chris Perez | 8 | 20 |
| 8 | Rachel Evans | 9 | 26 |

The query shows the number of books by an author along with the number of loans of their copies which is cut down to those with 3 or more books in the library. This is important in our business context because **we want to be aware of the most constant contributors.** If an author is active and has books that are frequent within the loaning timeframe (January 2024 to May 2025), we may give **more focus to that author’s works.**

### Query 2:

```
Query  Query History
10
11
12  -- 2. Find All Books Loaned in a Range ordered by date (Filtering with WHERE/BETWEEN)
13  SELECT l.borrow_date, b.book_title, c.copy_id, u.user_name
14  FROM loan l
15  JOIN "copy" c ON l.copy_id = c.copy_id
16  JOIN book b ON c.book_id = b.book_id
17  JOIN "user" u ON l.user_id = u.user_id
18  WHERE l.borrow_date BETWEEN '2025-01-01' AND '2025-03-31' -- Ensure this is the time frame wanted
19  ORDER BY borrow_date;
20
21
```

Data Output | Messages | Notifications

Showing rows: 1 to 40 | Page No: 1 of 1

| | borrow_date date | book_title text | copy_id integer | user_name text |
|---|---|---|---|---|
| 1 | 2025-01-01 | The Glass Castle | 123 | Shae Calbreath |
| 2 | 2025-01-04 | The Whispering Wind | 61 | Brew Guild |
| 3 | 2025-01-04 | The Shadow's Edge | 86 | Theodosia Bindin |
| 4 | 2025-01-08 | The Enchanted Forest | 52 | Aurilia Lokier |
| 5 | 2025-01-09 | The Empty Throne | 79 | Benji Ciccoloi |
| 6 | 2025-01-13 | Darkwood Tales | 70 | Aurilia Lokier |
| 7 | 2025-01-14 | The Serpent's Coil | 64 | Rafferty Goodyer |
| 8 | 2025-01-14 | Seven Moons | 34 | Haskell Hanhart |
| 9 | 2025-01-15 | Map of Dreams | 27 | Liz Andrat |

File saved succe

This query shows the loans that occurred within a timeframe. This is very important for our business context because it is showing **a traceback** of the data **ensuring that when something occurs (i.e. book is lost in library but returned in data)** an investigation can occur.

### Query 3:

```
Query  Query History
21
22  -- 3. Top 5 Most Loaned Books (JOIN, COUNT, ORDER BY)
23  SELECT b.book_title, a.author_name, COUNT(l.copy_id) AS loan_count
24  FROM loan l
25  JOIN "copy" c ON l.copy_id = c.copy_id
26  JOIN book b ON c.book_id = b.book_id
27  JOIN author a ON b.author_id = a.author_id
28  GROUP BY b.book_title, a.author_name
29  ORDER BY loan_count DESC
30  -- Limit 5 can be modifiable to have an extensive ranking
31  LIMIT 5;
```

Data Output | Messages | Notifications

| | book_title text | author_name text | loan_count bigint |
|---|---|---|---|
| 1 | The Broken Arrow | Rachel Evans | 8 |
| 2 | The Secret Library | Jared Morgan | 6 |
| 3 | The Sapphire Ri... | Jessica Taylor | 6 |
| 4 | Labyrinth of Lies | Nicole Hall | 5 |
| 5 | Darkwood Tales | Chris Perez | 5 |

This query shows the top 5 most loaned books along with author name. This is relevant to our business context because **we want to know which books are the most popular throughout the business**. Ensuring that these **books are served to customers now and beyond.**

**Assignment 1 Part C -** SE 02 Big Data & Analytics B

### Query 4:

```
Query  Query History
31  LIMIT 5;
32
33
34  -- 4. Users Who Have Never Borrowed a Book (Subquery)
35  SELECT u.user_name, u.user_id
36  FROM "user" u
37  WHERE u.user_id NOT IN (
38      SELECT DISTINCT l.user_id FROM loan l
39  )
40
41
```

Data Output  Messages  Notifications

| | user_name text | user_id [PK] integer |
|---|---|---|
| 1 | Kennith Hedon | 1 |
| 2 | Malorie Petri | 10 |
| 3 | Alicia Jagels | 30 |
| 4 | Rosco MacCallum | 34 |
| 5 | Katha Satch | 37 |

This query is simple but powerful, it shows the users that have never borrowed a book at all. The data is **very important for the business** to assure that **data processing (which is quite costly) is reduced and a reach-out to inactive users** is implemented.

### Query 5:

```
-- 5. Average Loan Duration Per User (CTE, Window Function) for bonus :- )
WITH loan_duration AS (
  SELECT
    l.user_id,
    (r.return_date - l.borrow_date) AS duration
  FROM loan l
  JOIN "return" r ON l.loan_id = r.loan_id
  WHERE r.return_date IS NOT NULL
)
SELECT
  u.user_name,
  ROUND(AVG(l.duration), 2) AS avg_loan_days
FROM "user" u
JOIN loan_duration l ON u.user_id = l.user_id
GROUP BY u.user_name
ORDER BY avg_loan_days DESC;
```

| | user_name text | avg_loan_days numeric |
|---|---|---|
| 1 | Adi Sacher | 22.00 |
| 2 | Madeleine Giacomi | 21.00 |
| 3 | Anetta Tuvey | 16.00 |
| 4 | Ezra Sheber | 15.50 |
| 5 | Marti Ablott | 15.50 |
| 6 | Theodosia Bindin | 15.00 |
| 7 | Omar Songist | 14.75 |
| 8 | Amos Nenci | 14.67 |
| 9 | Brigg Surtees | 14.00 |
| 10 | Gaynor McCamish | 14.00 |
| 11 | Dwight Delong | 14.00 |
| 12 | Jaquith Garbert | 14.00 |

This query shows the average loan duration of a user.
This is important to our business context so that **we can determine which users have the tendency to return books very late (either on time or not)**. This is also helpful to determine **which users hog a book for a long time on average**.

**Assignment 1 Part C -** SE 02 Big Data & Analytics B

**Part 4 - Challenges**

The most challenging part was **generating relevant data**. The **developer (KB)** had to specify terms within mockaroo in Rust (a language that he never learned), other than that, data schemas for the csv files were also checked with an **LLM** to diversify results which included:

- Having users that have not borrowed or had overdue books
- Matching staff shift time to loan time and return time
- Having books currently borrowed (recent time and labelled as borrowed in copies table)
- Having a distribution of random durations between loan date and return date for data diversification

And all the little details to ensure that the queries produce results that are different.

Another challenge was **querying**. So many syntax errors had to occur before releasing a final product.

The more “easier” part of the assignment was the python builder which the developer just referred to slides given in the teams. The logic for creating, filling and ensuring correctness is following this because **the developer is not a python coder.** However, the results were smooth and the builder is functioning.

To sum it up, the assignment was a fun one. This just goes to show how much effort is required to create mock data to test databases, thinking hard about working queries and ensuring that these are relevant to a certain topic.