Library Management Systems

Vasu Krishna Bandike, Vadlamudi Sai Venkata Nimish
University at Buffalo
UBIT: vasukris, svadlamu
UBMail:{vasukris, svadlamu}@buffalo.edu

***Abstract*—Key library operations difficulties, such as determining popular book genres, comprehending customer borrowing patterns, and examining the reasons behind late book returns, are intended to be addressed by the proposed library management system database. Libraries may increase customer happiness through data-driven insights, inventory management, and personalized recommendations by putting in place a strong database system. The system's design, implementation, and testing are described in this document, which also covers the use case, relational structure, creation of fictitious datasets, and query execution.**

I. Introduction

When using spreadsheets or manual methods, libraries often struggle to understand the preferences and behaviors of their users. This results in poor inventory control, inefficient handling of past-due books, and a dearth of customized recommendations. Setting up a database system allows the library to save, organize, and examine data on late returns, book genres, and borrowing trends.

II. DataBase over Excel

We choose databases over Excel sheets because DB provide SQL queries that are quick and simple to work with big amounts of data, while Excel's performance deteriorates as data volume grows. While Excel only has standard filters and a few more sophisticated capabilities, databases contain SQL queries for database operations, which makes it challenging. Although databases allow you control (filters) over the data, Excel files make it tough to relate all the tables. Coditions(joining) are primarily used in databases to join various tables, but they cannot be used in Excel files. Therefore, we chose databases over excel files for the aforementioned benefit.

III. Problem Statement

The suggested database system seeks to address the challenge of identifying which kinds of books are most likely to be borrowed by particular users by analyzing trends in their borrowing behavior. It also looks for the causes of late book returns, which can assist the library in enhancing its offerings and streamlining the loaning procedure.

Librarians can use this approach to forecast popular books, learn about user preferences, and pinpoint the elements (such as subscription types, occupations, or demographics) that influence late returns. The library will be able to optimize the lending process and improve user happiness by making data-driven decisions thanks to this investigation.

IV. Target Users

The following will be the main users of this library management system:

- **Library Employees:** Staff members working in the library will utilize the system to log loan transactions, add new books, manage user subscriptions, and respond to user feedback.
- **Library Users:** They will use the system to look for books, see their borrowing history, give comments, and verify due dates.
- **Administrators:** They will oversee the database, taking care of upkeep and guaranteeing data security and integrity.

V. E/R-Diagram

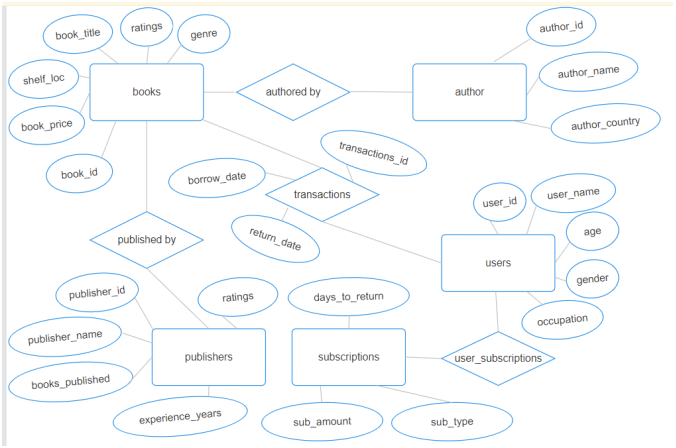


Fig. 1: ER diagram

VI. Tasks

A. Task-1: Use Case Domain

Library System Management:

- This includes managing book lending, monitoring user activity, and examining borrowing patterns and late returns as part of this use case.

Dataset:

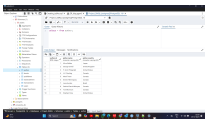
- Programmed generated fake dataset (codes used for creating fake data are provided in Task-5).

Updates and Queries:

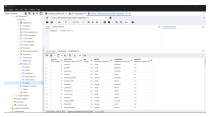
- Analyzing borrowing trends (such as the most often borrowed genres) and the causes of late returns are two possible areas of inquiry.
- There will be updates for tasks like maintaining user subscriptions, adding new books, and borrowing or returning books.

B. Task-2: Data Import

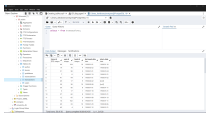
- A small amount of data, say 10 rows, has been first imported to the database.
- The screenshots of those queries with outputs are attached below.



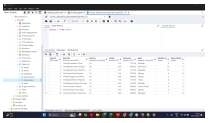
(a) Author



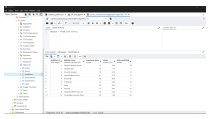
(b) Users



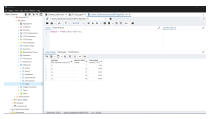
(c) Transactions



(d) Books



(e) Publishers



(f) Subscriptions

Issues Encountered During Loading:

1) We encountered problems importing the relations in the correct sequence when importing the data from fake generated data's CSV files (smaller versions, say 10 rows).
2) We were unable to import the files into the relations in a mindless manner because of several foreign key limitations.
3) The stage in which we resolved issues and completed an import order was as follows:
   - Author
   - Publishers
   - Users
   - Books
   - Transactions (Direct values were inserted for subscriptions)
4) Run the Insert statement Into subscriptions in create.sql,only after importing all the csv files for Author,Publishers, Users,Books,Transactions. If these values are already inside the subscriptions table Then due to foreign key constraints these data will be store inside the Users data which will cause error while importing CSV files for users.

C. Task-3: E/R Diagram and Relational Schema

E/R Diagram:

- Simplified version of the E/R diagram.

Fig. 3: Er diagram

Relational Schema:
Books Table:

```
Create table Books(
    Book_id INT PRIMARY KEY,
    Book_Title VARCHAR(55),
    Shelf_loc VARCHAR(50),
    Ratings DECIMAL(3, 2),
    Book_price DECIMAL(10, 2),
    Genre VARCHAR(50),
    Publisher_id INT,
    Book_authorid INT,
    FOREIGN KEY (Publisher_id)
     REFERENCES Publishers(Publisher_id));
Primary Key: Book_id
Foreign Keys: Publisher_id, Book_authorid
```

Users Table:

```
Create table Users (
    user_id INT PRIMARY KEY,
    user_name VARCHAR(55),
    Age INT,
    Gender VARCHAR(10),
    Occupation VARCHAR(50),
    Sub_type VARCHAR(10),
    FOREIGN KEY (Sub_type)
     REFERENCES Subscriptions(Sub_type));
Primary Key: user_id
Foreign Keys: Sub_type
```

Author Table:

```
Create table Author(
    Author_id INT PRIMARY KEY,
    Author_name VARCHAR(50),
    Author_country VARCHAR(50));
Primary Key: Author_id
Foreign Keys: No foreign keys
```

Transactions Table:

```
Create table Transactions (
    Trans_id INT PRIMARY KEY,
    User_id INT,
    Book_id INT,
    Borrowed_date DATE,
    Return_date DATE,
    FOREIGN KEY (User_id)
     REFERENCES Users(user_id),
```

```
    FOREIGN KEY (Book_id)
     REFERENCES Books(Book_id));
Primary Key: Trans_id
Foreign Keys: User_id, Book_id
```

Publishers Table:

```
Create table Publishers(
    Publisher_id INT PRIMARY KEY,
    Publisher_name VARCHAR(100) UNIQUE,
    Experience_years INT,
    Ratings DECIMAL(3, 2),
    Books_published INT);
Primary Key: Publisher_id
Foreign Keys: No foreign keys
```

Subscriptions Table:

```
Create table Subscriptions(
    Sub_type VARCHAR(10) PRIMARY KEY,
    days_to_return INT,
    Sub_amount DECIMAL(10, 2));
Primary Key: Sub_type
Foreign Keys: No foreign keys
```

Relationships Between Tables

Different parts of the library's data are stored in a number of interconnected tables that make up the library management system. These connections guarantee that information is appropriately arranged and available for a range of uses, including user subscriptions and book transactions. The following explains the connections between the tables:

Publishers and Books: The Publisher_id foreign key connects the Books table to the Publishers table. This connection shows that every book in the library has a publisher attached to it. Although a single publisher may publish several books, only one publisher is associated with each book.

Books and Authors: The Book_authorid foreign key connects the Books table to the Author table as well. This connection demonstrates that there is an author assigned to every book in the library. One author may write more than one book, but only one author may write a single book.

Users and Transactions: The User_id foreign key connects the Transactions table to the Users table. This relationship keeps track of users' borrowing activities. A user can have more than one transaction, but each transaction is associated with a single user.

Books and Transactions: The Book_id foreign key connects the Transactions database to the Books table as well. This displays the book that was checked out for each transaction. Although a book may be borrowed more than once, each transaction relates to a particular book.

Subscriptions and Users: The Sub_type foreign key establishes a connection between the Users and Subscriptions tables. This shows that every user has a subscription type, which establishes the user's borrowing rights, including the length of time they can check out books and the related subscription costs. There can be more than one user for each type of subscription.

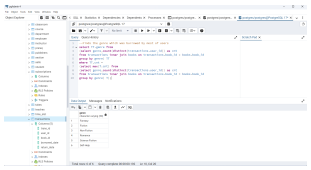
Foreign Key Restrictions: Maintaining referential integrity between the tables is aided by foreign keys. For example, if a user's subscription type is changed or removed, the Users table's foreign key makes sure that the associated data reflects the change, avoiding any invalid or orphaned records. In a similar manner, modifications made to the Books or Publishers table would be reflected in the associated Transactions records, guaranteeing that the borrowing history is consistent.

D. Task-4: Dataset Acquisition and Import

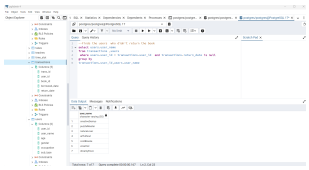
1) Acquired a large dataset by generating the fake dataset using scripts in Python.
2) Our dataset perfectly fits with the schema of the created database.
3) Imported the script-generated data to the SQL servers. Here are the screenshots after importing the large data to the SQL servers.
4) The fake dataset has many foreign keys which allow the dataset to support advanced queries.

E. Task-5: Query Execution and Codes

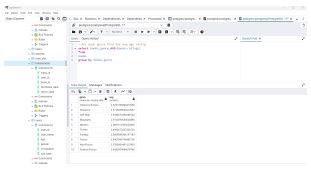
- We have executed the queries which are combined with: selection, subqueries, `GROUP_BY`, and joins.
- Below are the screenshots of those queries executed on PgAdmin:



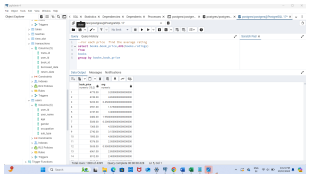
(a) Query 1



(b) Query 2



(a) Query 3



(b) Query 4

- **Codes Used:** Link to codes

F. Task-6: Boyce-Codd Normal Form (BCNF) Analysis

To ensure all relations are in Boyce-Codd Normal Form (BCNF), we first list all the functional dependencies for each table/relation:

1) 1) Books Table: **Schema:**

Books(Book_id, Book_Title, Shelf_loc, Ratings, Book_price, Genre, Publisher_id, Book_authorid)

Functional Dependencies:

1) Book_id $\rightarrow$ {Book_Title, Shelf_loc, Ratings, Book_price, Genre, Publisher_id, Book_authorid}
   - Explanation: Book_id is the superkey, and this is a non-trivial FD because the right-hand side attributes do not contain Book_id.

2) Publisher_id → {Publisher_name, Experience_years, Ratings, Books_published} (Foreign Key relation)
   - Explanation: Publisher_id is not a superkey within the Books table, but this FD exists due to the foreign key dependency with Publishers.
3) Book_authorid → {Author_name, Author_country} (Foreign Key relation)
   - Explanation: Book_authorid is not a superkey within the Books table, but this FD exists due to the foreign key dependency with Authors.

BCNF Results:

The Books table is already in BCNF.

2) 2) Users Table: **Schema:**

Users(user_id, user_name, Age, Gender, Occupation, Sub_type)

Functional Dependencies:

1) user_id → {user_name, Age, Gender, Occupation, Sub_type}
   - Explanation: user_id is the superkey, and this is a non-trivial FD because the right-hand side attributes do not contain user_id.
2) Sub_type → {days_to_return, Sub_amount} (Foreign Key relation)
   - Explanation: Sub_type is not a superkey within the Users table, but this FD exists due to the foreign key dependency with Subscriptions.

BCNF Results:

The Users table is already in BCNF.

3) 3) Transactions Table: **Schema:**

Transactions(Trans_id, User_id, Book_id, Borrowed_date, Return_date)

Functional Dependencies:

1) Trans_id → {User_id, Book_id, Borrowed_date, Return_date}
   - Explanation: Trans_id is the superkey, and this is a non-trivial FD because the right-hand side attributes do not contain Trans_id.
2) {User_id, Book_id} → Trans_id
   - Explanation: {User_id, Book_id} is a composite key (candidate key), and this is a non-trivial FD because the right-hand side does not contain {User_id, Book_id}.

BCNF Results:

The Transactions table is already in BCNF.

4) 4) Authors Table: **Schema:**

Author(Author_id, Author_name, Author_country)

Functional Dependencies:

1) Author_id → {Author_name, Author_country}
   - Explanation: Author_id is the superkey, and this is a non-trivial FD because the right-hand side attributes do not contain Author_id.

BCNF Results:

The Authors table is already in BCNF.

5) 5) Publishers Table: **Schema:**

Publishers(Publisher_id, Publisher_name, Experience_years, Ratings, Books_published)

Functional Dependencies:

1) Publisher_id → {Publisher_name, Experience_years, Ratings, Books_published}
   - Explanation: Publisher_id is the superkey, and this is a non-trivial FD because the right-hand side attributes do not contain Publisher_id.
2) Publisher_name → Publisher_id
   - Explanation: Publisher_name is a unique attribute, making it a candidate key. This is a non-trivial FD because the right-hand side does not contain Publisher_name.

BCNF Results:

The Publishers table is already in BCNF.

6) 6) Subscriptions Table: **Schema:**

Subscriptions(Sub_type, days_to_return, Sub_amount)

Functional Dependencies:

1) Sub_type → {days_to_return, Sub_amount}
   - Explanation: Sub_type is the superkey, and this is a non-trivial FD because the right-hand side attributes do not contain Sub_type.

BCNF Results:

The Subscriptions table is already in BCNF.

7) Final BCNF Observation:

- All tables are already in BCNF as the functional dependencies are non-trivial, and the left-hand sides are always superkeys.
- No further decomposition is required.
- Optimized schema.
- Redundancy free.
- Resistant to updating anomalies.

G. Task-7: Finalized E/R Diagram and Constraints

Finalized E/R Diagram:

- The finalized E/R diagram represents the optimized schema of the library management system.

Fig. 6: ER diagram

Constraints:

- For every table, we have a primary key.
- We have a unique constraint for `publisher_name` in the `Publishers` relation.
- All functional dependencies that exist do not violate BCNF, so there is no redundancy.

H. Task-8: Problems Encountered and Optimizations

Problems While Handling: Yes, managing a bigger dataset presented certain difficulties for us. The specifics are as follows:

Problems Faced:

1) **Degradation of Performance with More Data:**
   - Because subqueries were executed repeatedly and there was no indexing, some queries' runtimes increased noticeably.
   - Particularly in inquiries that calculated aggregate numbers, subqueries in `CASE` statements were being called again and redundantly.
2) **Primary Keys That Are Not Indexed:**
   - Due to the lack of indexes on primary keys, queries that involved joins across many tables (such as `Books`, `Transactions`, `Users`, etc.) encountered performance bottlenecks.
   - The use of sequential scans lengthened the query execution time.
3) **Bringing in Data in the Right Order:**
   - We encountered problems with foreign key restrictions when loading the dataset since it resulted in violations when data was inserted into child tables before parent tables.
   - For instance, `Transactions` refers to both `Books` and `Users`, whereas `Books` refers to `Publishers`. The data import would fail if there was no set order.

Optimizations of Problems:

1) **Optimization with CTEs:**
   - Queries with repeated subqueries, like computing `MAX(Borrowed_date)`, are optimized by avoiding the use of subqueries and instead replacing these with **CTEs**.
   - This eliminates redundant calculations, contributing significantly to query runtime improvement, going as high as 152ms for one query.
2) **Indexing for Primary Keys:**
   - To solve the problem caused by a lack of indexing, we have created indexes for the primary keys of each table with the sql queries
   - This speeded up several queries (examples are mentioned in Task-10).
3) **Data Import Sequence:**
   - To avoid issues with the dependency of foreign keys, we followed the strict order below:
     - Import Order: `Author`, `Publishers`, `Users`, `Books`, `Transactions` (direct values were inserted for `Subscriptions`).
   - By ensuring parent tables were populated before child tables, we avoided foreign key violations.

I. Task-9: Queries

Query 1:

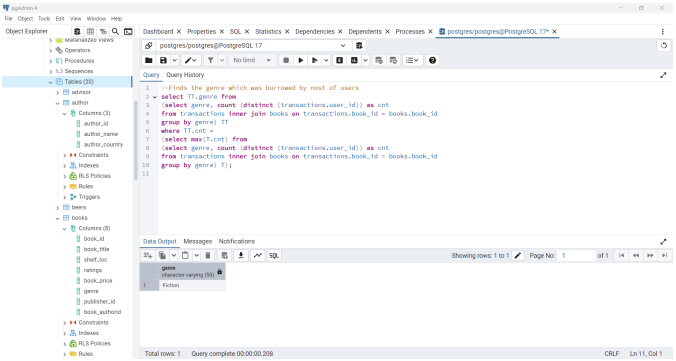


Fig. 7: Query 1

Query 2:

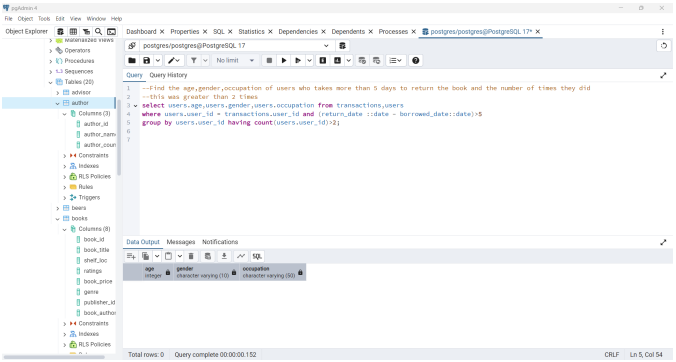


Fig. 8: Query 2

Query 3:

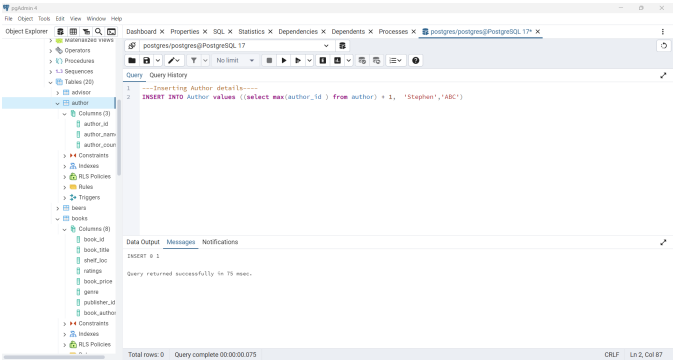


Fig. 9: Query 3

Query 4:

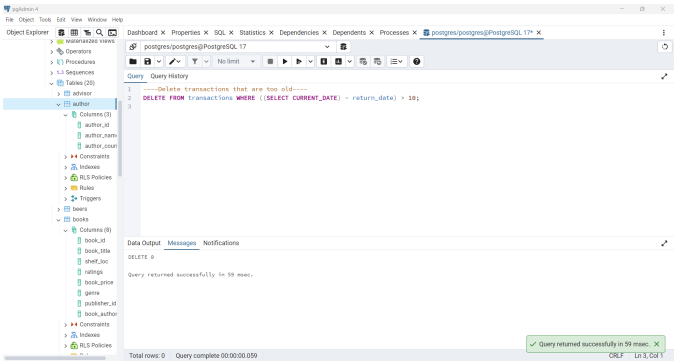


Fig. 10: Query 4

Query 5:

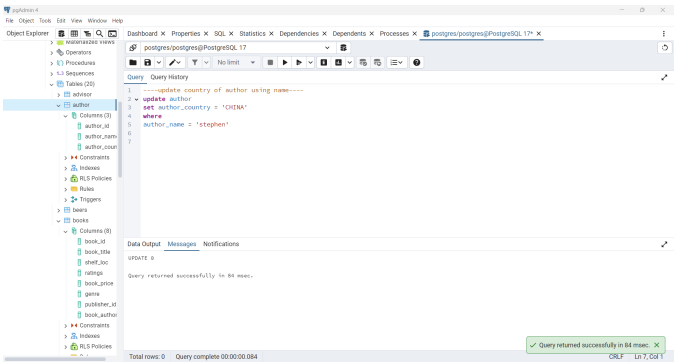


Fig. 11: Query 5

Query 6:

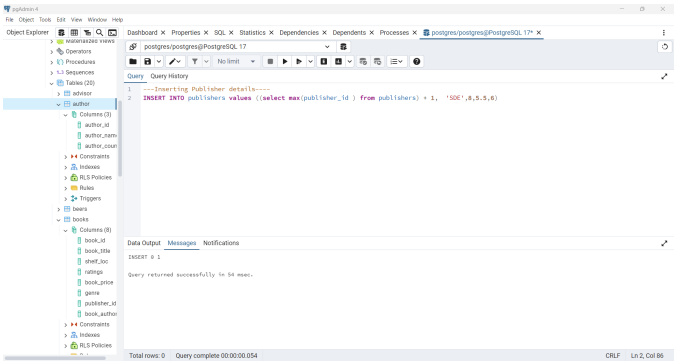


Fig. 12: Query 6

Query 7:

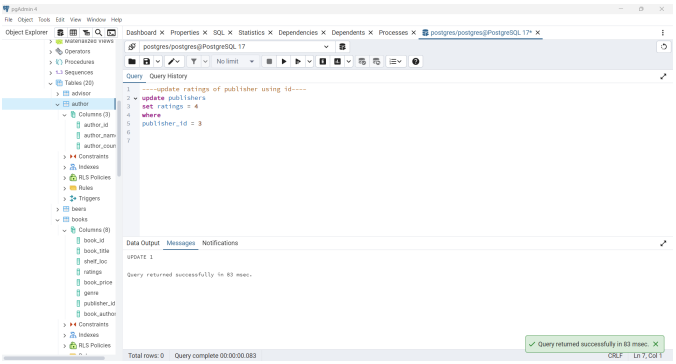


Fig. 13: Query 7

Query 8:

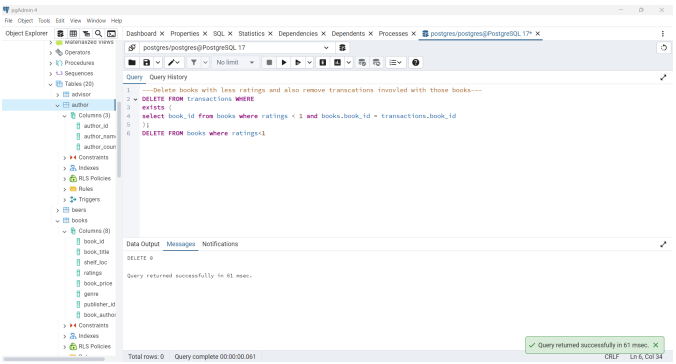


Fig. 14: Query 8

Query 9:

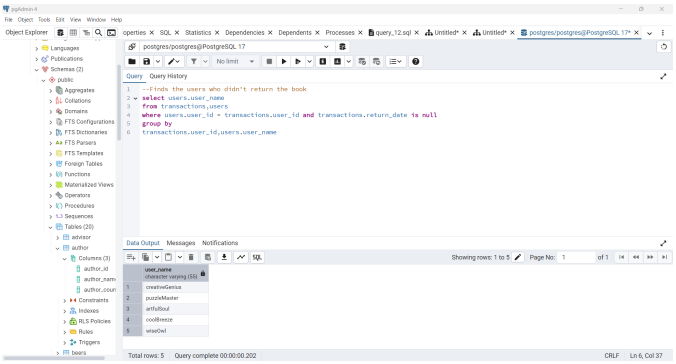


Fig. 15: Query 9

Query 10:

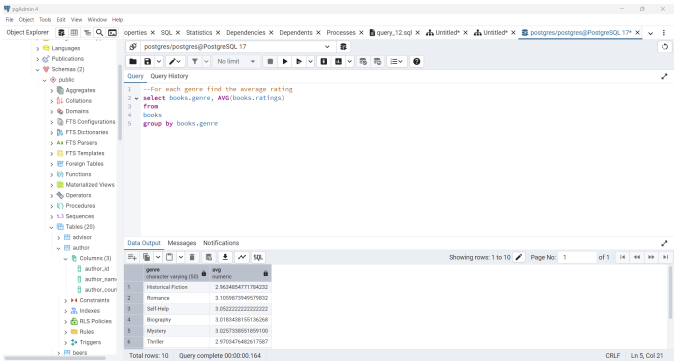


Fig. 16: Query 10

Query 11:

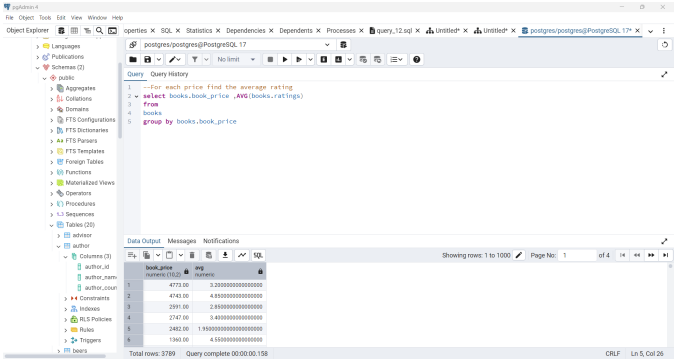


Fig. 17: Query 11

Query 12:

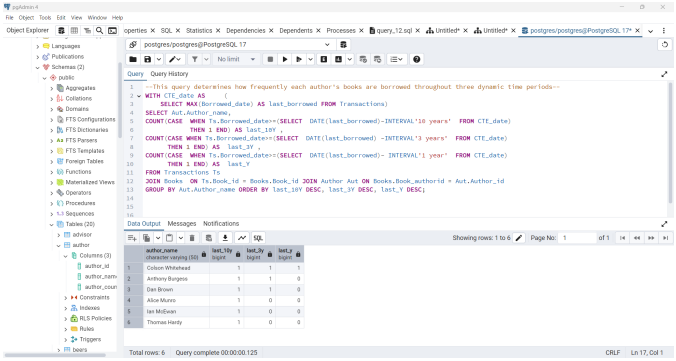


Fig. 18: Query 12

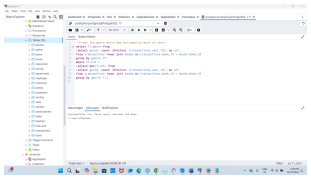
J. Task-10: Query Execution Analysis

Problem for a Frequent Author Books Query:

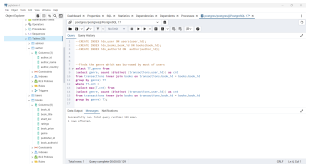
- **Screenshot of query using subqueries:**
  - Total query runtime: 239 msec
- **Screenshot of query using CTE (Common Table Expression):**
  - Total query runtime: 87 msec
- **Execution Plan:**
  - The problem in the above query is that subqueries are being called thrice in every `CASE` statement.
  - This problem is optimized by creating a CTE.
  - Resulted in a 152 msec reduction in total query runtime.

Problem for Almost Every Query: (3 examples are listed below)

1) **Query-1:**



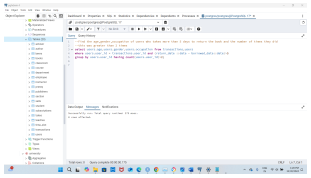
(a) Query 1(with out indexing)



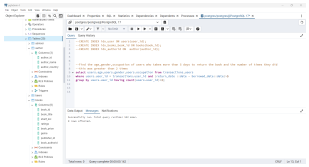
(b) Query 1(with indexing)

- **Before Indexing:** Total query runtime: 141 msec
- **After Indexing:** Total query runtime: 129 msec
- **Difference:** 12 msec

2) **Query-2:**



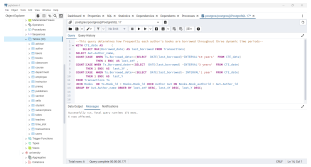
(a) Query 2(with out indexing)



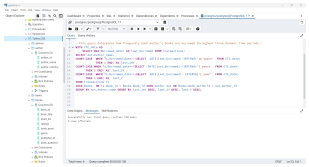
(b) Query 2(with indexing)

- **Before Indexing:** Total query runtime: 175 msec
- **After Indexing:** Total query runtime: 142 msec
- **Difference:** 33 msec

3) **Query-3:**



(a) Query 3(with out indexing)



(b) Query 3(with indexing)

- **Before Indexing:** Total query runtime: 171 msec
- **After Indexing:** Total query runtime: 138 msec
- **Difference:** 33 msec

Execution Plan:

- The problem in the above queries is the lack of indexing.
- This problem is optimized by creating new indexes for the primary keys of all tables using the following commands:

```
CREATE INDEX idx_books_book_id
ON Books(Book_id);
CREATE INDEX idx_users_user_id
ON Users(user_id);
CREATE INDEX idx_transactions_trans_id
ON Transactions(Trans_id);
```

```
CREATE INDEX idx_authors_author_id
ON Author(Author_id);
CREATE INDEX idx_publishers_publisher_id
ON Publishers(Publisher_id);
CREATE INDEX idx_subscriptions_sub_type
ON Subscriptions(Sub_type);
```

- Resulted in reductions of 12 msec, 33 msec, and 33 msec in the total query runtime for Query-1, Query-2, and Query-3, respectively.
- This indexing also reduced runtime for all other queries; however, three examples are explained for demonstration.

VII. Contribution

A. *Vasu*

Worked on Data Creation,building website, Task - 6,7

B. *Nimish*

Worked on Data Creation,building website, Task - 8,10