Candra Kurniawan Mentoring Week 6 - SQL Query

SQL and Relational Database - Job Preparation Program - Pacmann AI

Report

1. **Project Objectives**

Description:

Create an e-library application that helps people for easy accessibility and centralized all the library into one platform and giving the best experience for exploring, borrowing, and interacting with a wide array of books.

2. **Designing the databases**

- **Mission statement:**

The mission of the E-Library Database Project is to create a robust and efficient system that facilitates the management of multiple libraries and their diverse collections. This application aims to provide users with a seamless experience in exploring, borrowing, and interacting with a wide array of books. The primary focus is on streamlining the processes of book lending, user registration, and ensuring a well-organized categorization of books for easy accessibility.

Features:

Multi-Library Management:

- The system will support the administration of multiple libraries, each serving as a distinct entity with its unique collection of books.

Comprehensive Book Collection:

- The database will store detailed information about each book, including titles, authors, and available quantities.

- Books will be categorized into genres such as self-improvement, biography, fantasy, romance, science fiction, etc., enhancing the user's ability to search and discover books.

User Registration:

- Users will have the ability to register on the e-library platform, allowing them to engage in various activities like borrowing books, placing holds, and managing their accounts.

Loan and Hold System:

- Users can borrow books from any library within the application, with a maximum limit of 2 books at a time.

- The loan period is set at 2 weeks, and users can return books earlier than the due date.

- The system will automatically handle book returns after the due date.

- Users can place holds on books that are currently unavailable, and the system will maintain a hold queue.

- A book becomes available for borrowing by the customer at the front of the queue, and if not borrowed within one week, it is released for others to borrow.

Limitations:

While the E-Library Database Project aims to be comprehensive and user-friendly, there are certain limitations:

- The system does not provide real-time book availability updates, and users may experience delays in accessing popular books.

- The application does not include a feature for user reviews and ratings of books. Detailed user activity analytics are not within the scope of this project.

- **Creating Table Structure**

After outlining the Mission Statement for the E-Library Database Project, the next step is to design the table structure. In this stage, we will identify the objects needed for this database. These objects can eventually become tables. After identifying the tables and their descriptions, specify the fields and keys for each table. Begin by determining candidate keys, then choose the primary key among these candidates.

Tables	Columns	Descriptions
library	library_id serial PK NOT NULL library_name varchar NOT NULL Location_provider varchar NOT NULL	Stores information about each library, with Library_ID as the primary key.
book	book_id int PK NOT NULL library_id FK1 NOT NULL library title varchar NOT NULL quantity_available int NOT NULL category varchar NOT NULL	Contains detailed information about each book, with Book_ID as the primary key. It includes information about the quantity of available books.
users	user_id int PK NOT NULL user_name varchar NOT NULL password varchar NOT NULL email varchar NOT NULL	Contains information about registered user, with User_ID as the primary key.
transaction	transaction_id int PK NOT NULL user_id int FK1 user NOT NULL book_id int FK2 book NOT NULL transaction_type varchar NOT NULL loan_date date NOT NULL due_date date NOT NULL return_date date NOT NULL	Records book lending transactions, with Transaction_ID as the primary key. It also includes information about the transaction_type, loan date, due date, and return date.
reservation	reservation_id int PK NOTNULL user_id int FK1 user NOT NULL book_id int FK2 book NOT NULL queue_position int NOT NULL reservation_date date NOT NULL	Stores information about the reservation book queue, with reservation_id as the primary key. It includes details about the hold date.

- **Determine Table Relationships**

  Now that we have identified the tables for the E-Library Database Project, let's determine the relationships between these tables.

  1. Library and Book:

     - Relationship Type: One-to-Many (1:N)
     - Foreign Key: LibraryID in the Book table (connects to Library_ID in the Library table)
     - Explanation: One library can have many books, but each book belongs to only one library.

  2. Users and Transaction:

     - Relationship Type: One-to-Many (1:N)
     - Foreign Key: UserID in the Transaction table (connects to user_id in the Users table)
     - Explanation: One user can have multiple transactions, but each transaction is associated with only one user.

  3. Book and Transaction:

     - Relationship Type: Many-to-Many (M:N)
       Foreign Keys:
     - book_id in the Transaction table (connects to book_id in the Book table)
     - user_id in the Transaction table (connects to user_id in the Users table)
     - Explanation: Many books can be involved in many transactions. This is achieved through an intermediary table (Transaction) connecting books and users.

  4. User and reservation:

     - Relationship Type: One-to-Many (1:N)
     - Foreign Key: user_id in the reservation table (connects to user_id in the User table)
     - Explanation: One user can have multiple holds, but each hold is associated with only one user.

  5. Book and reservation:

     - Relationship Type: Many-to-Many (M:N)
       Foreign Keys:
     - book_id in the reservation table (connects to book_id in the Book table)
     - user_id in the reservation table (connects to user_id in the User table)

- Explanation: Many books can be in many hold queues. This is achieved through an intermediary table (reservation) connecting books and users.

Additional Notes:

- The one-to-many relationships indicate that one record in the first table can be related to multiple records in the second table, but each record in the second table is related to only one record in the first table.

- The many-to-many relationships are implemented using an intermediary table to connect the related records in the two tables.

These relationships help define how data is shared and connected between the different tables in the database.

- **Determine Business Rules**

After we define the tables and the inheritance now we define the constraint of each column in the tables.

1. library
   library_id = NOT NULL (PK)
   library_name= NOT NULL
   location = NOT NULL

2. book
   book_id = NOT NULL (PK)
   tItle = NOT NULL
   author = NOT NULL
   quantity_available = NOT NULL CHECK (quantity_available >= 0)
   category = NOT NULL

3. users
   user_id = NOT NULL (PK)
   user_name = NOT NULL
   password = NOT NULL
   email = NOT NULL

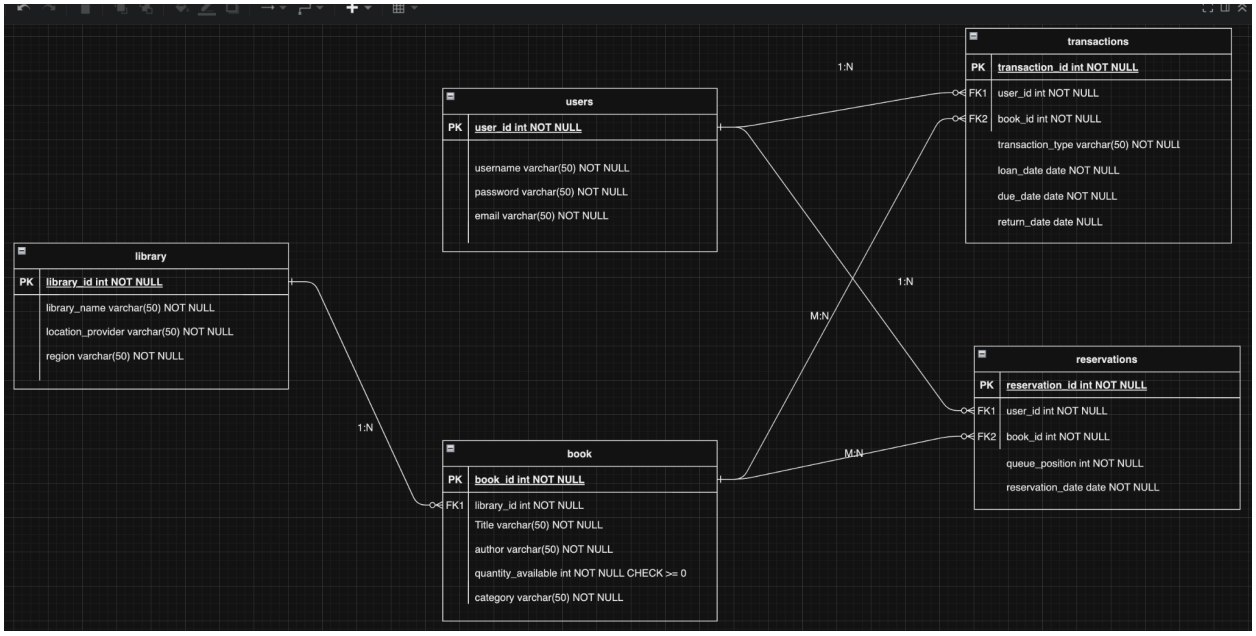
4. transactions
   transaction_id = NOT NULL (PK)
   user_id = NOT NULL (Foreign Key referencing User Table)
   book_id = NOT NULL (Foreign Key referencing Book Table)
   transaction_type = NOT NULL

loan_date = NOT NULL
due_date = NOT NULL
return_date = (Nullable, as books might not have been returned yet)

5. reservation
   reservation_id = NOT NULL (PK)
   user_id = NOT NULL (Foreign Key referencing User table)
   book_id: NOT NULL (Foreign Key referencing Book table)
   queue_position = NOT NULL
   reservation_date = NOT NULL

- **Implementing The Design**

  ERD



Picture 0.1 ERD e-library app

Preparing the dummy data with Python Faker library

1. **First, we initialize all the library and package that will be needed for create dummy data**

```
import pandas as pd
import numpy as np
import faker_commerce
import csv
import random

from faker import Faker
from sqlalchemy import create_engine
from tabulate import tabulate
from datetime import datetime

# set localization in Indonesia
FAKER = Faker('id_ID')

fake = Faker()
nama = fake.name()

print(nama)
```

Picture 0.2 Initialize library and packages

2. **Second, for easy visualize the dummy data that we create. We use library tabulate for create a table inside the python.**

```
def show_data(table):
    """
    Fungsi untuk menampilkan data
    arg:
        - table (dict) : data dictionary yang ingin ditampilkan
    return:
        None
    """
    tab = tabulate(tabular_data = table,
                   headers = table.keys(),
                   tablefmt = "psql",
                   numalign = "center")
    print(tab)
```

Picture 0.3 Function tabulate for table

3. **Next, we create a python function for creating dummy data.**

```
def table_library(n_dummy_library, is_print):
    '''
    Fungsi untuk membuat dummy data pada table library untuk e-library project
    Pacmann
    header:
        - library_id (SERIAL) PK
        - library_name (varchar(50)) NOT NULL
        - location_provider (varchar(50)) NOT NULL
        - region (varchar(50)) NOT NULL

    arg:
        - n_dummy_library (int) : Jumlah perpustakaan dummy yang ingin dibuat
        - is_print (bool) : Jika True akan menampilkan hasil data.
    return:
        - table (list) :
    '''
    # Buat table
    table = {}
    table["library_id"] = [i+1 for i in range(n_dummy_library)]
    table["library_name"] = [f'library-{str(i + 1).zfill(3)}' for i in range(n_dummy_library)]
    table["location_provider"] = [fake.random_element(elements=('GCP', 'AWS', 'Azure', 'Aliyun')) for i
in range(n_dummy_library)]
    table["region"] = [fake.random_element(elements=('Asia', 'US', 'Europe', 'China')) for i in
range(n_dummy_library)]

    # Print table
    if is_print:
        show_data(table)
    return table

# membuat data table library
table_library = table_library(n_dummy_library=100,
                              is_print=True)
```

Picture 0.4 Function dummy data library

4. **Next, we create a python function for creating book dummy data.**

```
def table_book(tables_library, n_dummy_book, is_print):
    '''
    Fungsi untuk membuat dummy data pada table book untuk e-library project
    Pacmann
    header:
        - book_id (int) NOT NULL PK
        - library_id (int) NOT NULL FK1 from table library
        - title (varchar(50)) NOT NULL
        - author (varchar(50)) NOT NULL
        - quantity_available INT NOT NULL CHECK >= 0
        - category (varchar(50)) NOT NULL

    arg:
        - n_dummy_book (int) : Jumlah data book dummy yang ingin dibuat
        - is_print (bool) : Jika True akan menampilkan hasil data.
    return:
        - table (list) :
    '''
    # Buat table
    table = {}
    table["book_id"] = [i+1 for i in range(n_dummy_book)]
    table["library_id"] = [fake.random_element(tables_library['library_id']) for i in
range(n_dummy_book)]
    table["title"] = [fake.catch_phrase() for _ in range(n_dummy_book)]
    table["author"] = [fake.name() for i in range(n_dummy_book)]
    table["quantity_available"] = [fake.random_int(min = 0, max = 100) for i in range(n_dummy_book)]
    table["category"] = [fake.random_element(elements=('Fiction', 'Non-Fiction', 'Science', 'History',
'Fantasy', 'Mystery', 'Romance', 'Historical Fiction')) for i in range(n_dummy_book)]

    # Print table
    if is_print:
        show_data(table)
    return table

# membuat data table library
table_book = table_book(tables_library = table_library, n_dummy_book=200,
                            is_print=True)
```

Picture 0.5 Function dummy data book

5. **Next, we create a python function for creating dummy data users.**

```python
def table_user(num_dummy_user, is_print):
    '''
    Fungsi untuk membuat dummy data pada table book untuk e-library project
    Pacmann
    header:
        - user_id (SERIAL) NOT NULL PK
        - username (varchar(50)) NOT NULL
        - password (varchar(50)) NOT NULL
        - email (varchar(50)) NOT NULL

    arg:
        - n_dummy_user (int) : Jumlah data user dummy yang ingin dibuat
        - is_print (bool) : Jika True akan menampilkan hasil data.
    return:
        - table (list) :
    '''
    # Buat table
    table = {}
    table["user_id"] = [1000 + i for i in range(num_dummy_user)]
    table["username"] = [fake.user_name() for index in range(num_dummy_user)]
    table["password"] = [fake.password() for index in range(num_dummy_user)]
    table["email"] = [fake.email() for index in range(num_dummy_user)]
    # Print table
    if is_print:
        show_data(table)
    return table

# membuat data dummy table user
table_user = table_user(num_dummy_user=100, is_print=True)
```

Picture 0.6 Function dummy data user

6. **Next, we create a python function for creating transaction dummy data.**

```python
def table_transaction(tables_user, tables_book, num_dummy_transaction, is_print):
    '''
    Fungsi untuk membuat dummy data pada table user untuk e-library project
    Pacmann
    header:
        - transaction_id (SERIAL) NOT NULL PK
        - user_id (SERIAL) FK From table user
        - book_id (SERIAL) FK From table book
        - transaction_type (VARCHAR(50)) NOT NULL
        - loan_date (Date) NOT NULL
        - due_date (DATE) NOT NULL
        - return_date (DATE) (NULL)

    arg:
        - num_dummy_transaction (int) : Jumlah data user transaction yang ingin dibuat
        - is_print (bool) : Jika True akan menampilkan hasil data.
    return:
        - table (list) :
    '''
```

```
    # Buat table
    table = {}
    table["transaction_id"] = [20230 + i for i in range(num_dummy_transaction)]
    table["user_id"] = [fake.random_element(tables_user['user_id']) for i in
range(num_dummy_transaction)]
    table["book_id"] = [fake.random_element(tables_book['book_id']) for i in
range(num_dummy_transaction)]
    table["transaction_type"] = [fake.random_element(elements = ('Borrow', 'Return')) for i in
range(num_dummy_transaction)]
    table['loan_date'] = [fake.date_between(start_date='-30d', end_date='today').strftime('%Y-%m-%d')
for i in range(num_dummy_transaction)]
    table['due_date'] = [fake.date_between(start_date='today', end_date='+14d').strftime('%Y-%m-%d') for
i in range(num_dummy_transaction)]
    table['return_date'] = [fake.date_between(start_date='-30d', end_date='today').strftime('%Y-%m-%d')
if np.random.choice([True, False]) else None for i in range(num_dummy_transaction)]

    # Print table
    if is_print:
        show_data(table)
    return table

# membuat data dummy table transaction
table_transaction = table_transaction(tables_user = table_user,
                                      tables_book = table_book,
                                      num_dummy_transaction = 500,
                                      is_print = True)
```

Picture 0.7 Function dummy data transaction

7. **Next, we create a python function for creating reservations table dummy data.**

```
def table_reservation(tables_user, tables_book, num_dummy_reservation, is_print):
    '''
    Fungsi untuk membuat dummy data pada table user untuk e-library project
    Pacmann
    header:
        - reservation_id (int) NOT NULL PK
        - user_id (int) NOT NULL FK1 From table user
        - book_id (int) NOT NULL FK2 From table book
        - queue_position (int) NOT NULL
        - reservation_date (date) NOT NULL

    arg:
        - num_dummy_reservation (int) : Jumlah data dummy reservasi yang ingin dibuat
        - is_print (bool) : Jika True akan menampilkan hasil data.
    return:
        - table (list) :
    '''
    # Buat table
    table = {}
    table["reservation_id"] = [20231 + i for i in range(num_dummy_reservation)]
    table["user_id"] = [fake.random_element(tables_user['user_id']) for i in
range(num_dummy_reservation)]
    table["book_id"] = [fake.random_element(tables_book['book_id']) for i in
range(num_dummy_reservation)]
    table["queue_position"] = [np.random.randint(1, 20) for _ in range(num_dummy_reservation)]
    table["reservation_date"] = [fake.date_between(start_date='-30d',
```

```
end_date='today').strftime('%Y-%m-%d') for _ in range(num_dummy_reservation)]
    # Print table
    if is_print:
        show_data(table)
    return table

# membuat data dummy table reservation
table_reservation = table_reservation(tables_user = table_user,
                                      tables_book = table_book,
                                      num_dummy_reservation = 200,
                                      is_print = True)
```

Picture 0.8 Function dummy data reservations

8. **Lastly, after we create all the dummy data for the tables. We need to save the data into CSV file that later we import into the PostgreSQL Database.**

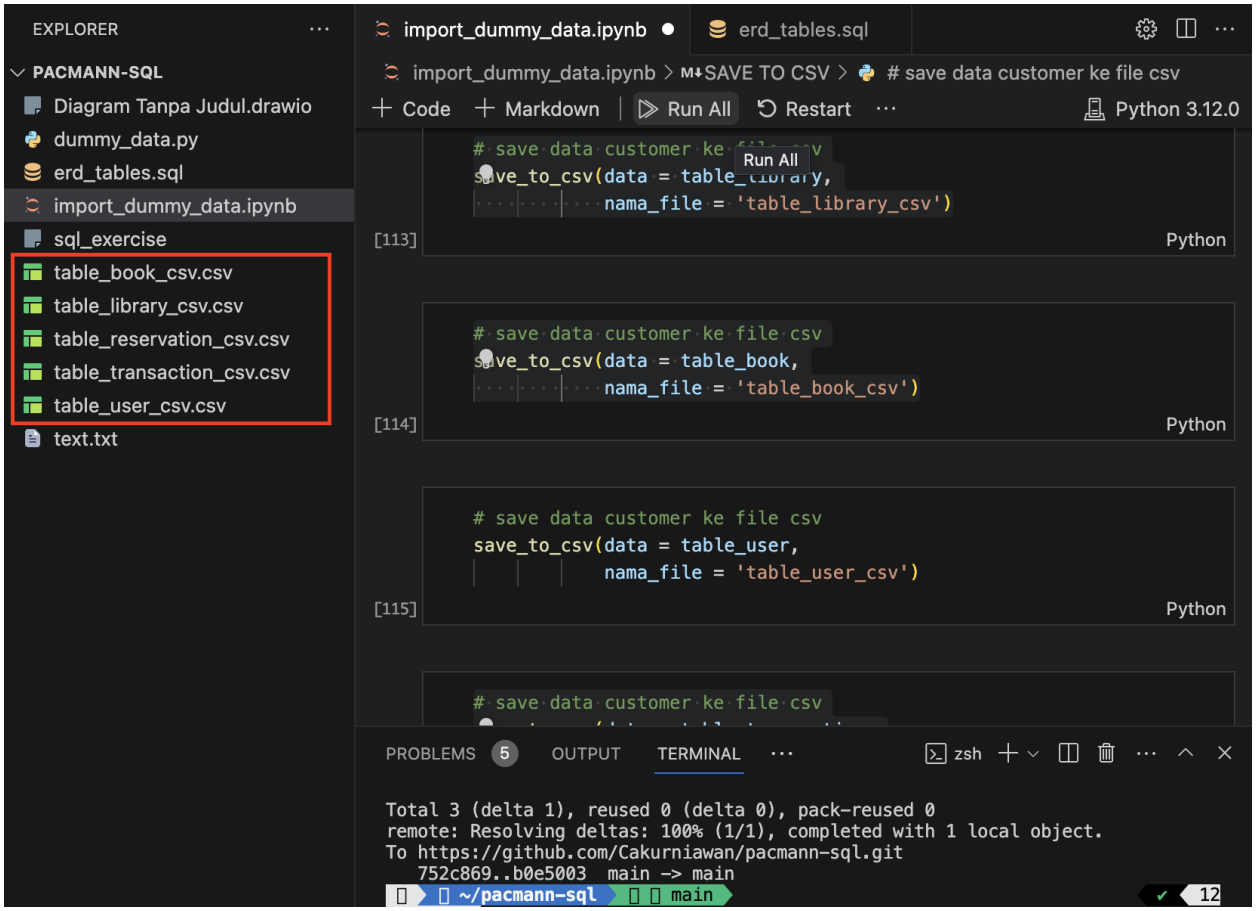
```
def save_to_csv(data, nama_file):
    '''
    Fungsi untuk menyimpan data dummy ke csv
    args:
        - data (list)     : list of dictionary data yang akan dijadikan csv
        - nama_file (str) : nama untuk file csv

    return:
        - None
    '''

    # Membuat file csv
    with open(file = f"{nama_file}.csv", mode = 'w', newline = '') as csv_file:
        # Membuat writer csv
        writer = csv.writer(csv_file)
        # write header csv
        writer.writerow(list(data.keys()))

        # mengetahui panjang data
        len_data = len(list(data.items())[0][1])

        # write data ke file csv
        for i in range(len_data):
            row = []
            for key in data.keys():
                row.append(data[key][i])
            writer.writerow(row)
```

Picture 0.9 Function export to CSV

```
# save data customer ke file csv
save_to_csv(data = table_library,
            nama_file = 'table_library_csv')

# save data customer ke file csv
save_to_csv(data = table_book,
            nama_file = 'table_book_csv')

# save data customer ke file csv
save_to_csv(data = table_user,
```

```
            nama_file = 'table_user_csv')

# save data customer ke file csv
save_to_csv(data = table_transaction,
            nama_file = 'table_transaction_csv')

# save data customer ke file csv
save_to_csv(data = table_reservation,
            nama_file = 'table_reservation_csv')
```

Here are the CSV files that are successfully exported from the tabulate table and functions.



Picture 1.0 Function export to CSV 2

Create tables and Import the dummy data into PostgreSQL Database.

After we successfully create the dummy data for all the tables that we needed for the e-library application. Next, we need to create the table for storing the dummy data.

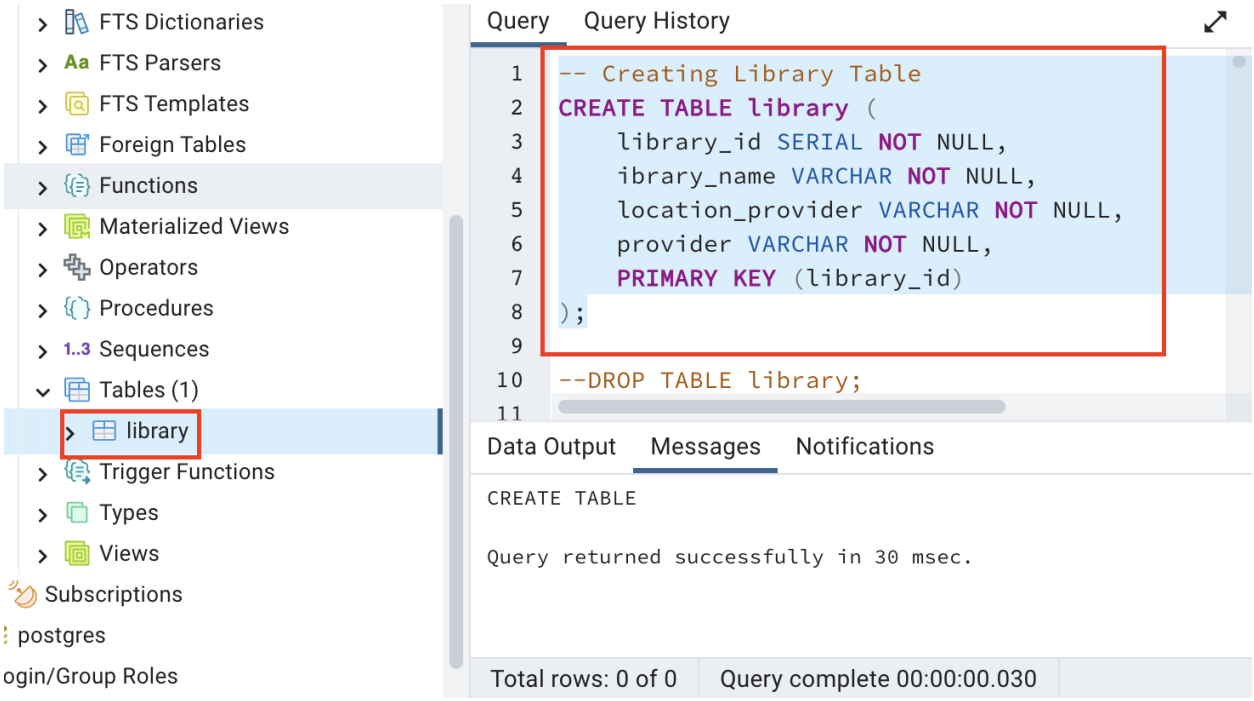
Here are the sql DDL syntax for creating tables and define constraint for the e-library application.

Create Library Table

First, we create a table called library for storing the information about each library that will be included in the application, there will be **100 total library** in the application that will provided a books.

Here is the syntax for create library table with the primary key in the library_id, and the other columns.

```
-- Creating Library Table
CREATE TABLE library (
    library_id SERIAL NOT NULL,
    ibrary_name VARCHAR NOT NULL,
    location_provider VARCHAR NOT NULL,
    provider VARCHAR NOT NULL,
    PRIMARY KEY (library_id)
);
-- DROP TABLE library;
```



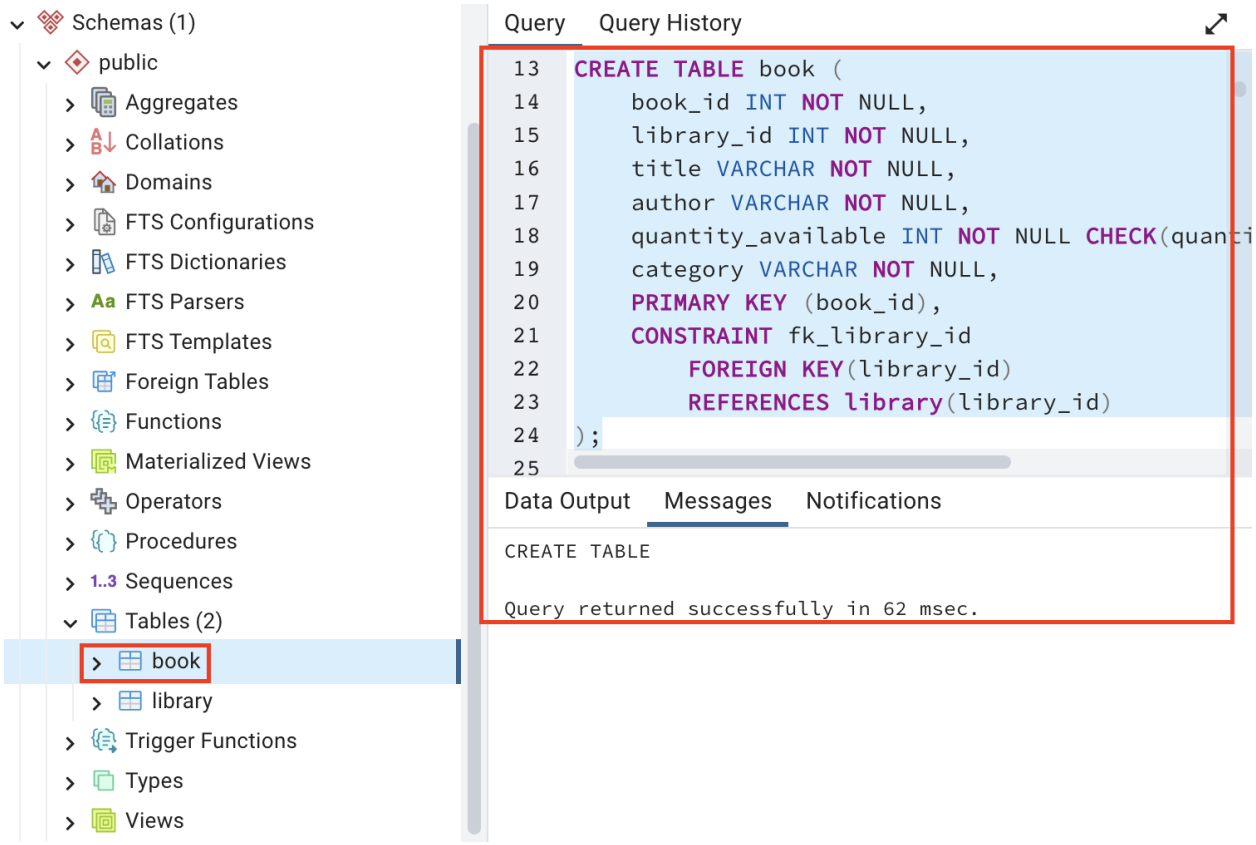
picture 1.0 Create library table

Create Book Table

Second, we create a table called book for storing the book that will be included in the application, there will be **200 total books** in the application.

Here is the syntax for create book table with the primary key in the book_id, and the other columns. And on this table there will be a **1 foreign key** that will be using a column **library_id** from library table.

```
-- Creating Book Table
CREATE TABLE book (
    book_id INT NOT NULL,
    library_id INT NOT NULL,
    title VARCHAR NOT NULL,
    author VARCHAR NOT NULL,
    quantity_available INT NOT NULL CHECK(quantity_available >= 0),
    category VARCHAR NOT NULL,
    PRIMARY KEY (book_id),
    CONSTRAINT fk_library_id
        FOREIGN KEY(library_id)
        REFERENCES library(library_id)
);
-- DROP TABLE book;
```



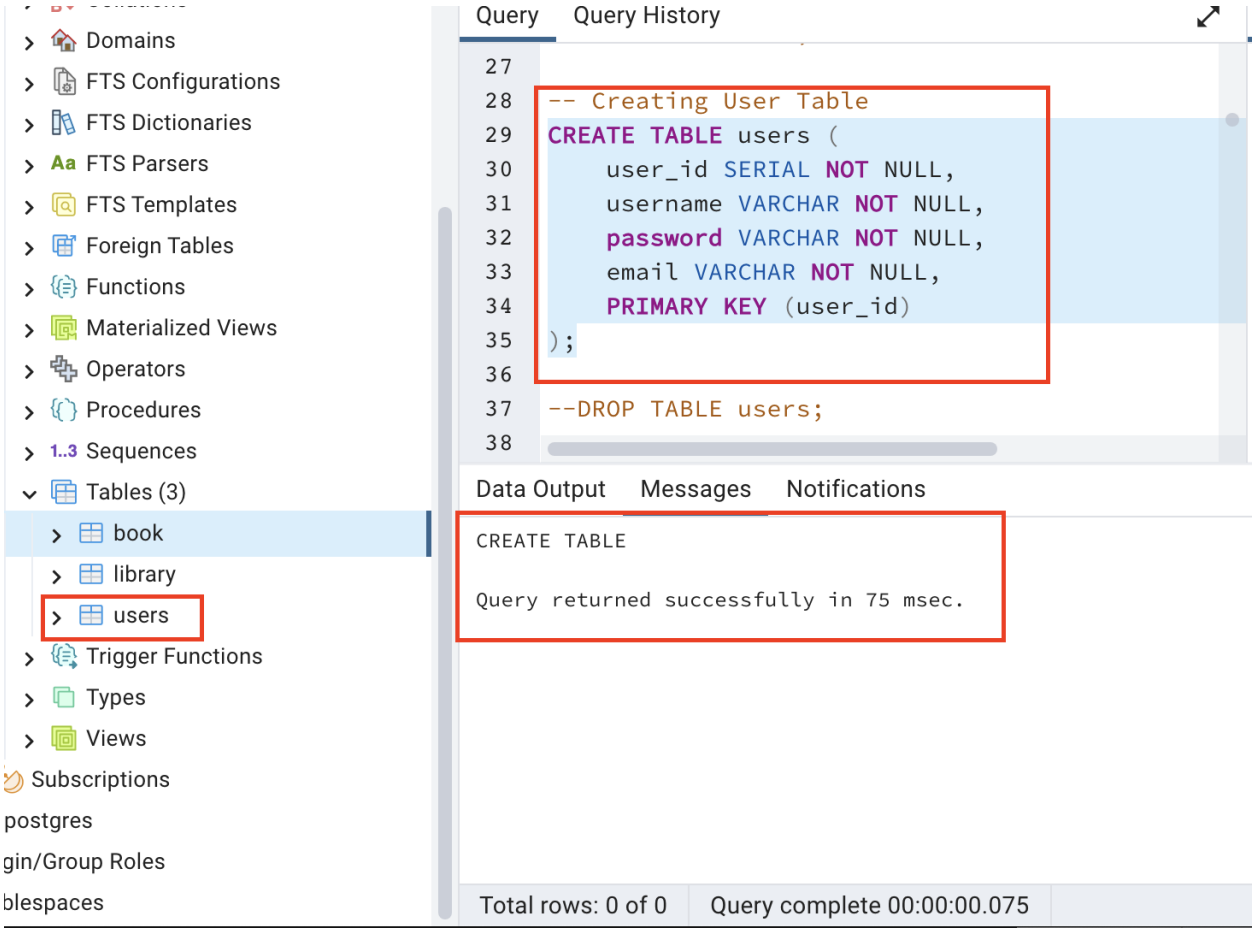
picture 1.1 create book table

Create Users Table

Next, we create a table called **users** for storing the customer that will be make a transactions (Borrowing books or Returning a book). Because this time we using a dummy data. So there will be a total **100** user on this table and the application.

Here is the syntax for create **users** table with the primary key in the user_id, and the other columns.

```
-- Creating User Table
CREATE TABLE users (
    user_id SERIAL NOT NULL,
    username VARCHAR NOT NULL,
    password VARCHAR NOT NULL,
    email VARCHAR NOT NULL,
    PRIMARY KEY (user_id)
);
-- DROP TABLE users;
```



picture 1.2 create users table

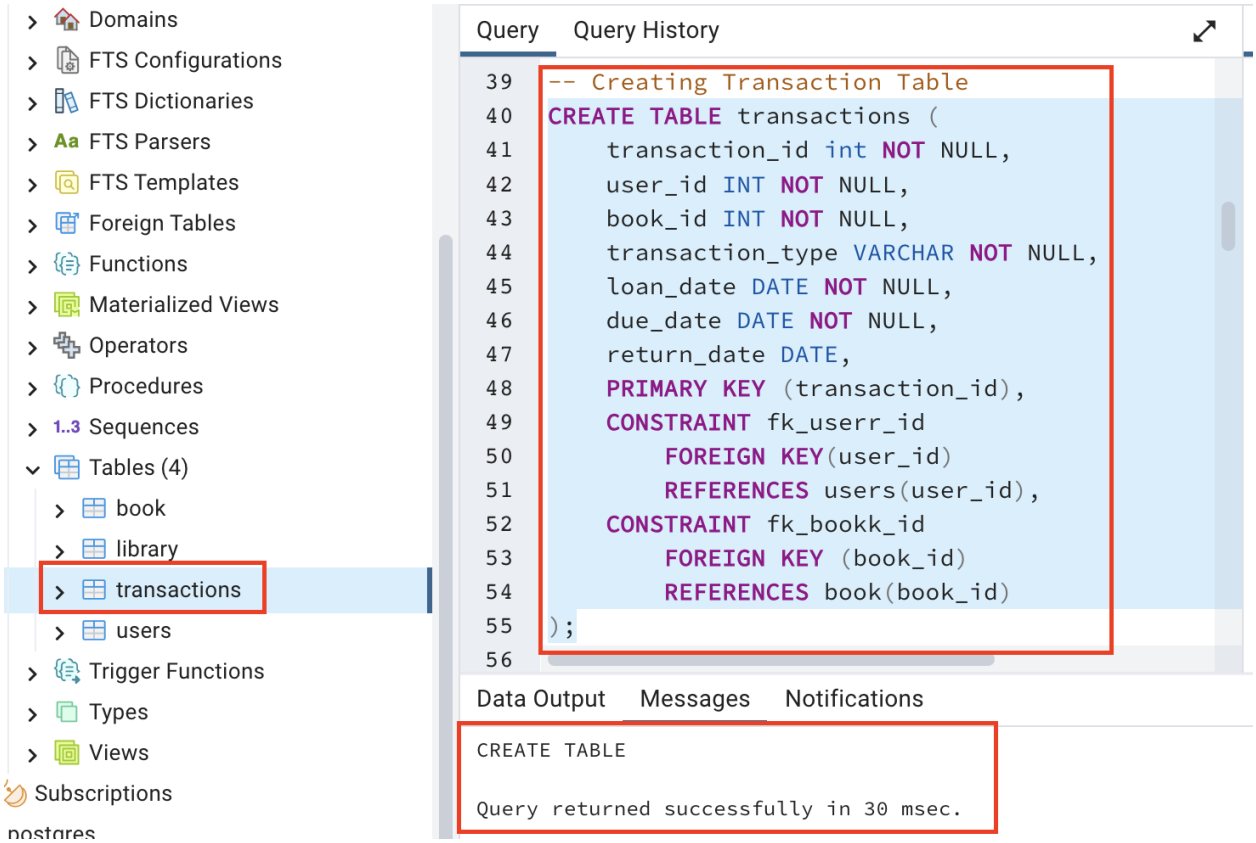
Create Transactions Table

Next, we create a table called **transactions** for storing the customer that will be make a transactions (Borrowing books or Returning a book) and for storing the information when the customer have borrowed and returned the books. There will be a **500** total transactions on this table.

Here is the syntax for create table transactions with the primary key in the transaction_id, and the other columns. And on this table there will be a **1 foreign key** that will be using a column **user_id** from library users. And **1 more Foreign key** on **book_id** column from book table

```
-- Creating Transaction Table
CREATE TABLE transactions (
    transaction_id int NOT NULL,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,
```

```
    return_date DATE,
    PRIMARY KEY (transaction_id),
    CONSTRAINT fk_userr_id
        FOREIGN KEY(user_id)
        REFERENCES users(user_id),
    CONSTRAINT fk_bookk_id
        FOREIGN KEY (book_id)
        REFERENCES book(book_id)
);
-- DROP TABLE transactions;
```



picture 1.3 create transaction table

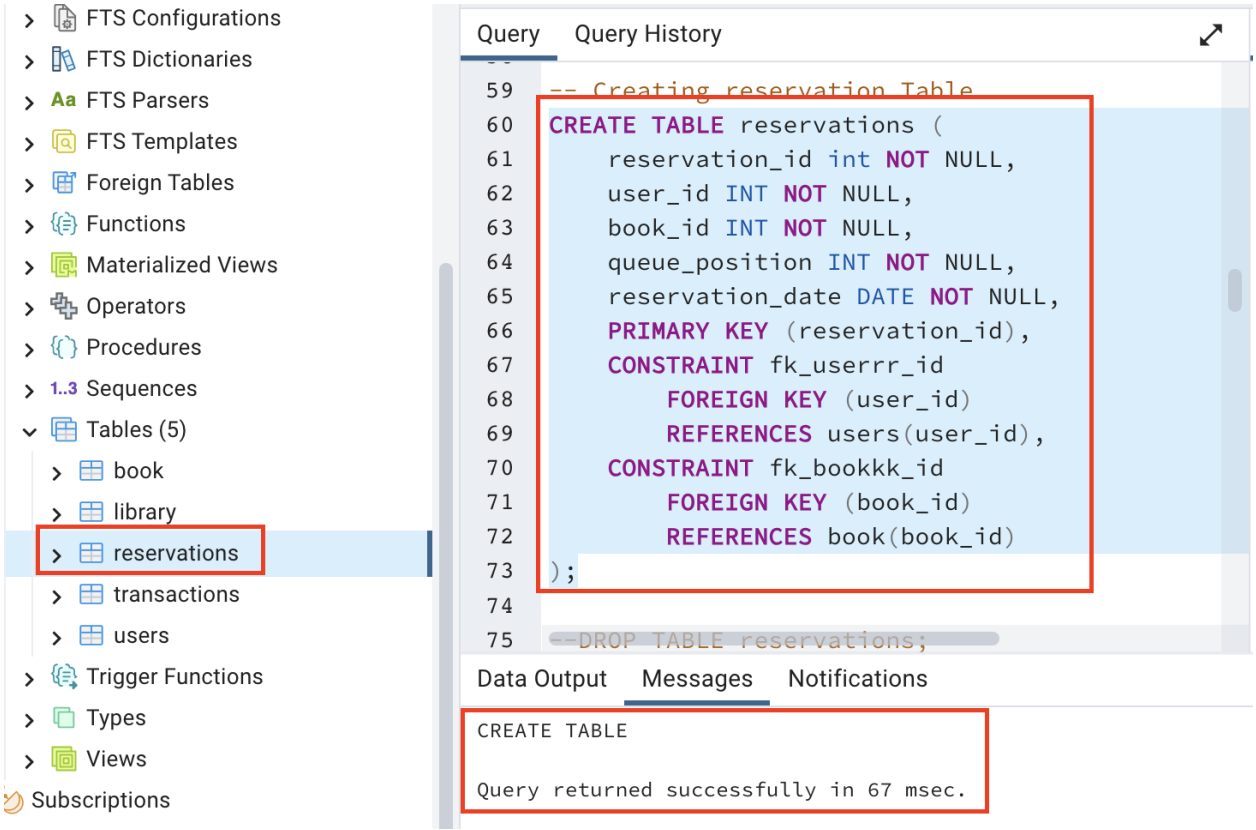
Create Reservations Table

Next, we create a table called **reservations.** If the customers didn't get the book that they want, because there has somebody else that have borrowed the books.. So, the customer need to queue their book. Because this time we using a dummy data. So there will be a total **200** reservation data on this table.

Here is the syntax for create **reservations table** with the primary key in the reservation_id, and the other columns. And on this table there will be a **1 foreign key** that will

be using a column **user_id** from library users. And **1 more Foreign key** on **book_id** column from book table

```
-- Creating reservations Table
CREATE TABLE reservations (
    reservation_id int NOT NULL,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    queue_position INT NOT NULL,
    reservation_date DATE NOT NULL,
    PRIMARY KEY (reservation_id),
    CONSTRAINT fk_userrr_id
        FOREIGN KEY (user_id)
        REFERENCES users(user_id),
    CONSTRAINT fk_bookkk_id
        FOREIGN KEY (book_id)
        REFERENCES book(book_id)
);
-- DROP TABLE reservations
```

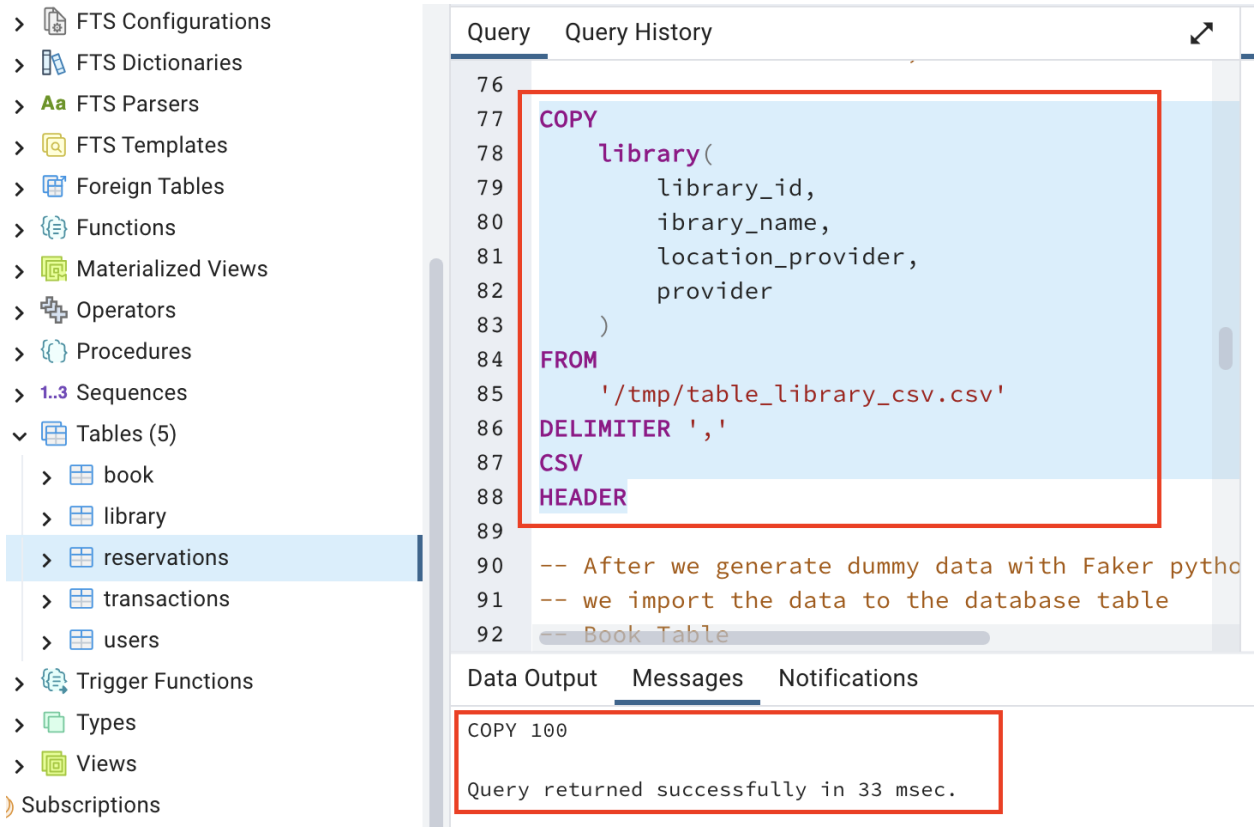


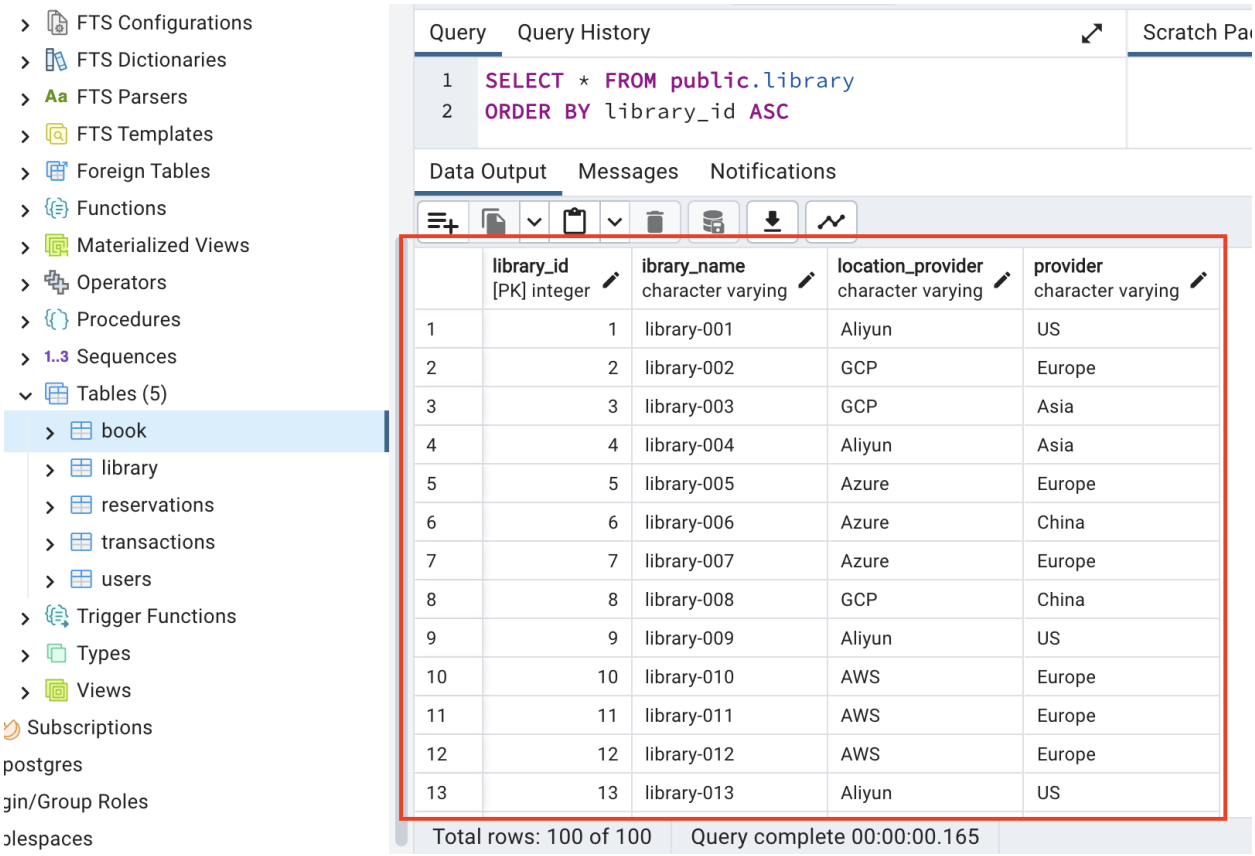
Picture 1.4 create reservations table

Importing dummy data after creating tables

Import dummy data for library table with Faker python.

```
-- After we generate dummy data with Faker python
-- we import the data to the database table
-- Library Table
COPY
    library(
        library_id,
        ibrary_name,
        location_provider,
        provider
    )
FROM
    '/tmp/table_library_csv.csv'
DELIMITER ','
CSV
HEADER
```

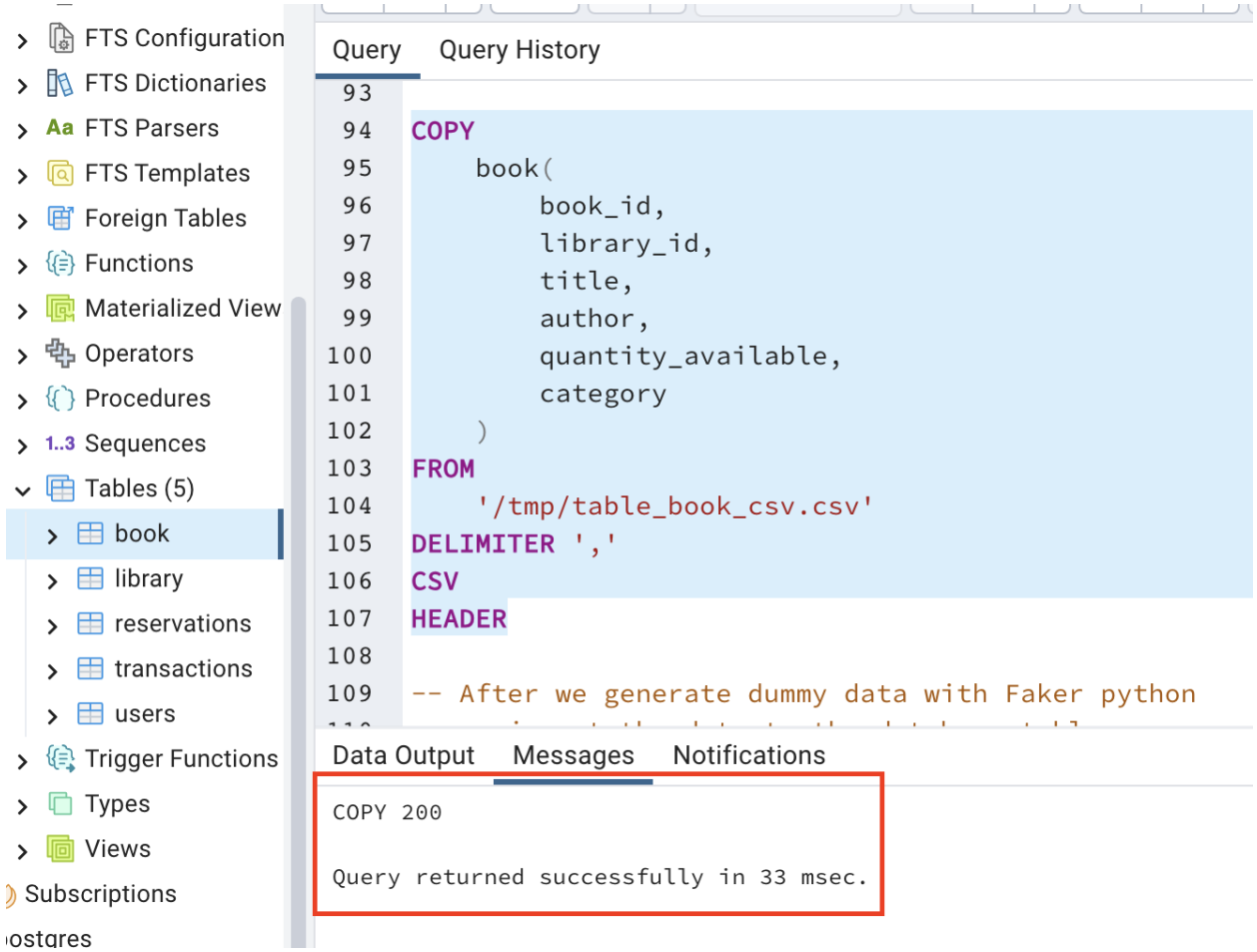


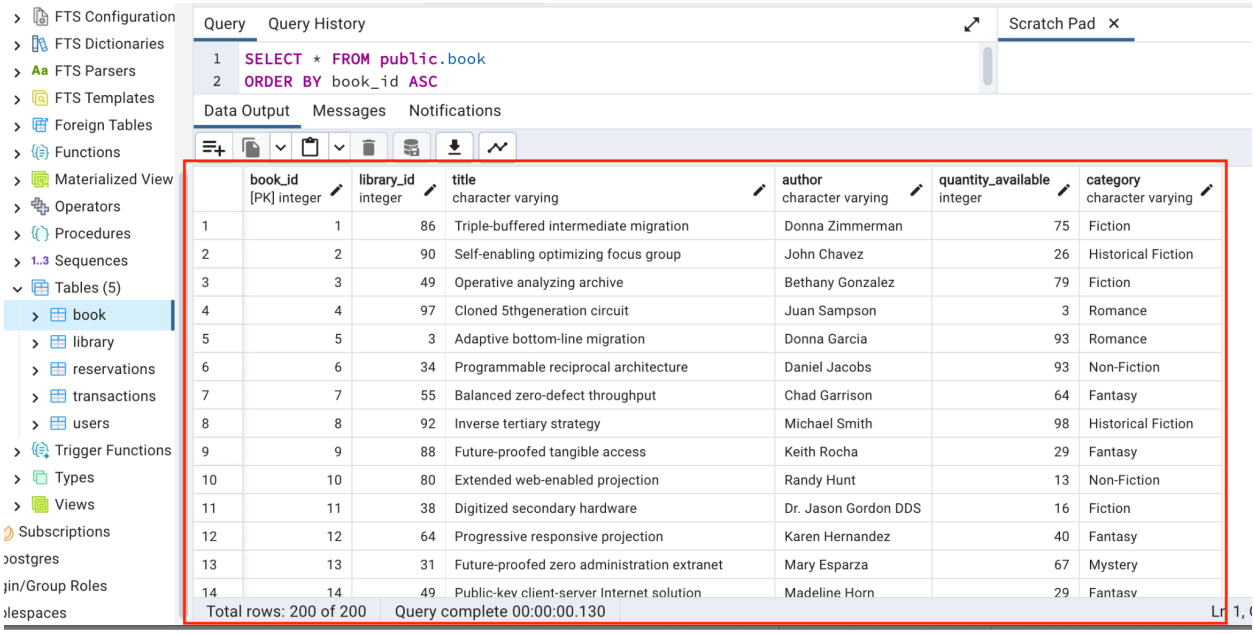


Picture 1.5 import data library table

Import dummy data for book table with Faker python.

```
-- After we generate dummy data with Faker python
-- we import the data to the database table
-- Book Table
COPY
    book(
        book_id,
        library_id,
        title,
        author,
        quantity_available,
        category
    )
FROM
    '/tmp/table_book_csv.csv'
DELIMITER ','
CSV
HEADER
```

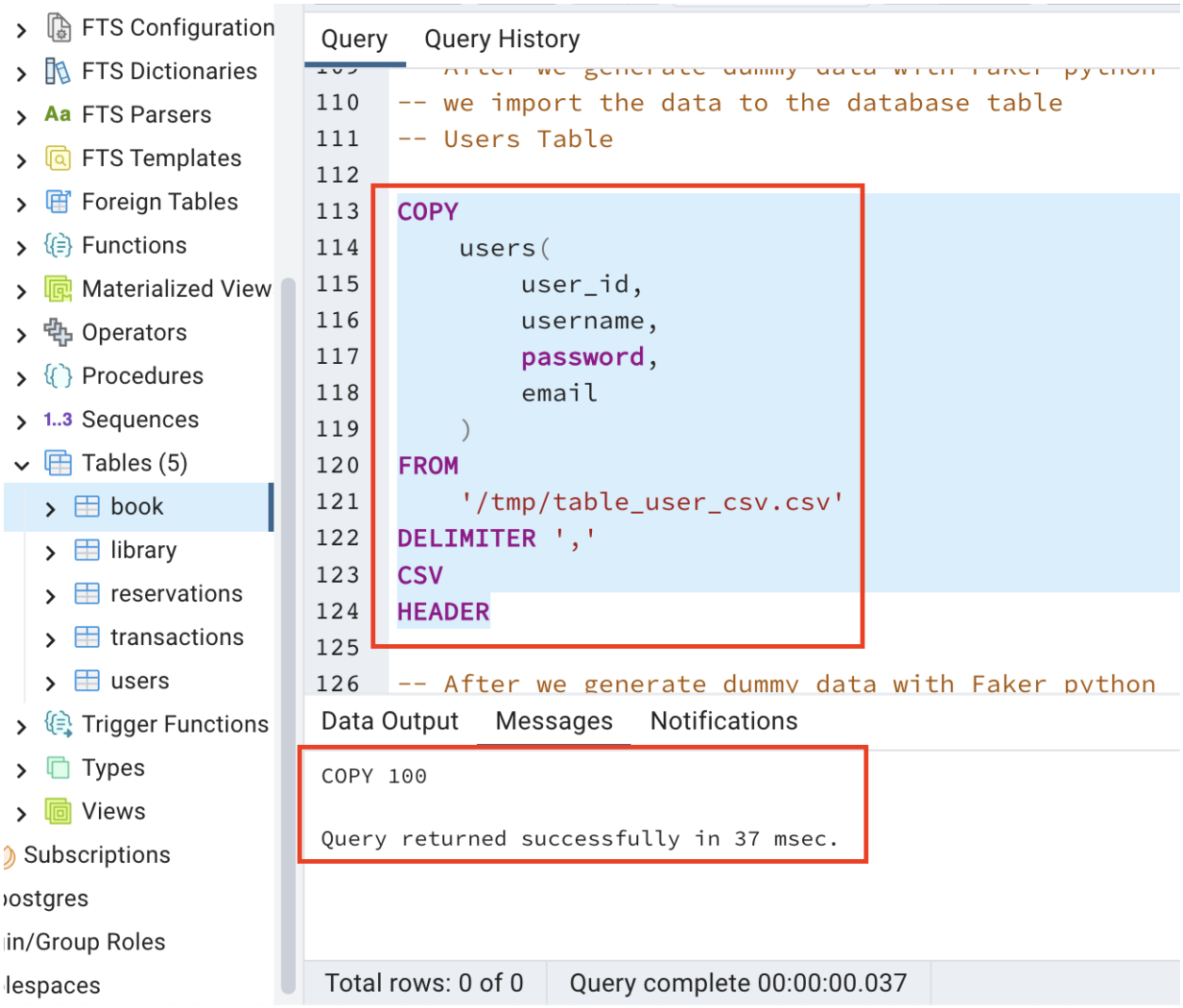


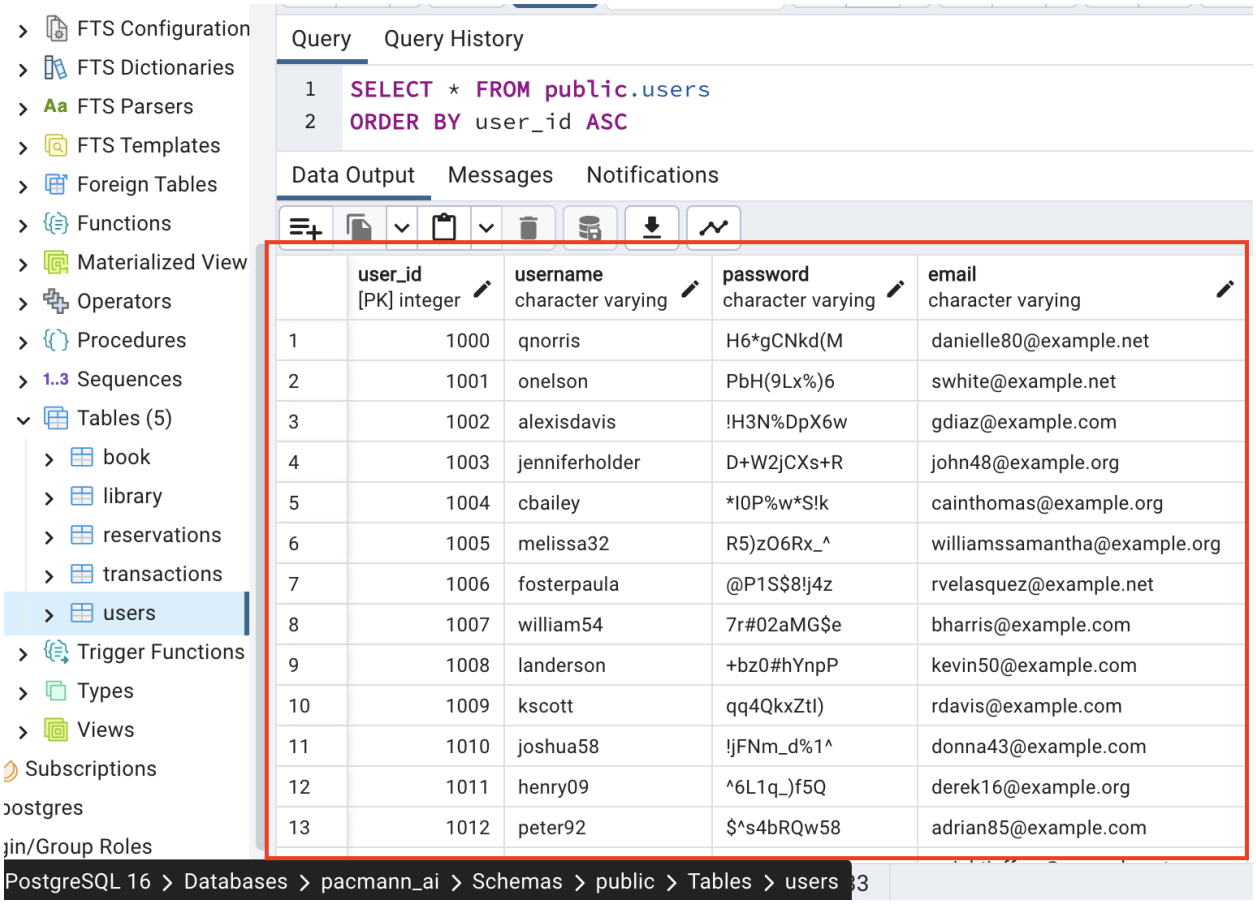


Picture 1.6 import data book table

Import dummy data for Users table with Faker python.

```
-- After we generate dummy data with Faker python
-- we import the data to the database table
-- Users Table
COPY
    users(
        user_id,
        username,
        password,
        email
    )
FROM
    '/tmp/table_user_csv.csv'
DELIMITER ','
CSV
HEADER
```


FTS Configuration
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized View
Operators
Procedures
Sequences
Tables (5)
book
library
reservations
transactions
users
Trigger Functions
Types
Views
Subscriptions
ostgres
in/Group Roles
lespaces
Query
Query History
-- we import the data to the database table
-- Users Table
COPY
users(
user_id,
username,
password,
email
)
FROM
'/tmp/table_user_csv.csv'
DELIMITER ','
CSV
HEADER
-- After we generate dummy data with Faker python
Data Output
Messages
Notifications
COPY 100
Query returned successfully in 37 msec.
Total rows: 0 of 0
Query complete 00:00:00.037

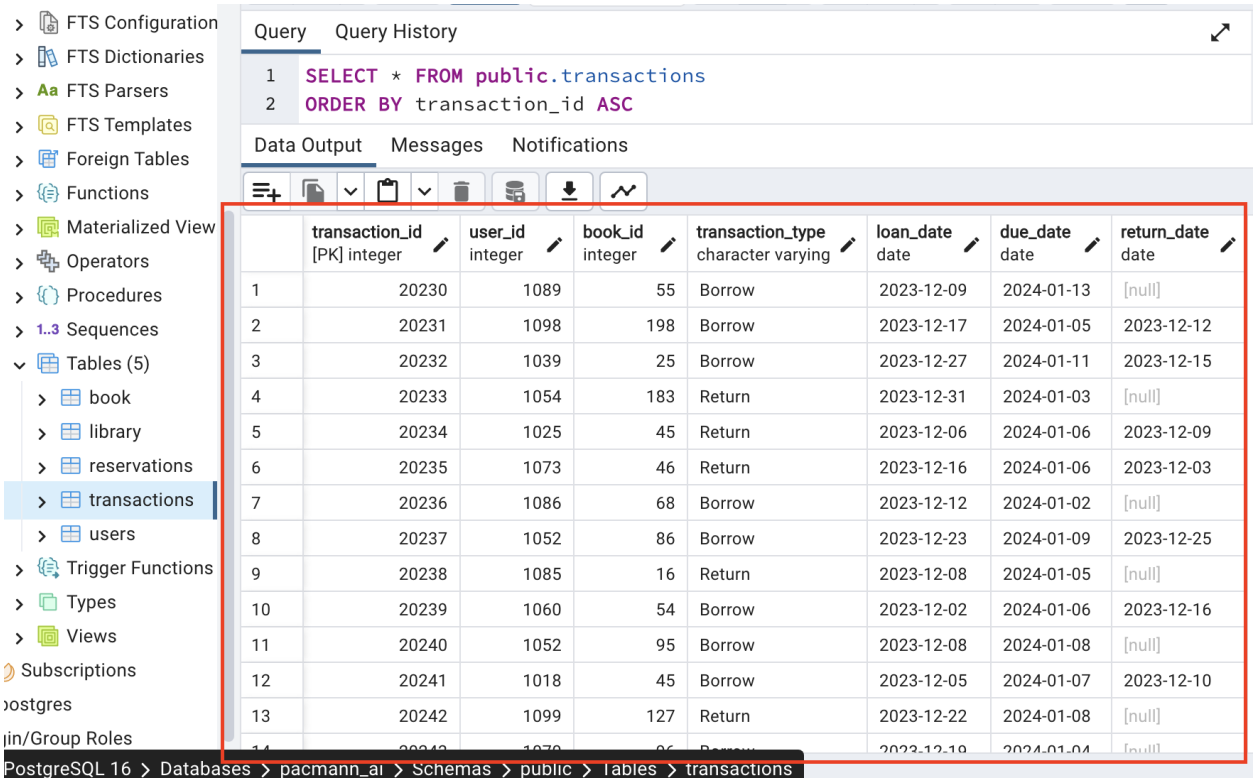


Picture 1.7 import data users table

Import dummy data for Transactions table with Faker python.

```
-- After we generate dummy data with Faker python
-- we import the data to the database table
-- Transactions Table
COPY
    transactions(
        transaction_id,
        user_id,
        book_id,
        loan_date,
        due_date,
        return_date
    )
FROM
    '/tmp/table_transaction_csv.csv'
DELIMITER ','
CSV
HEADER
```

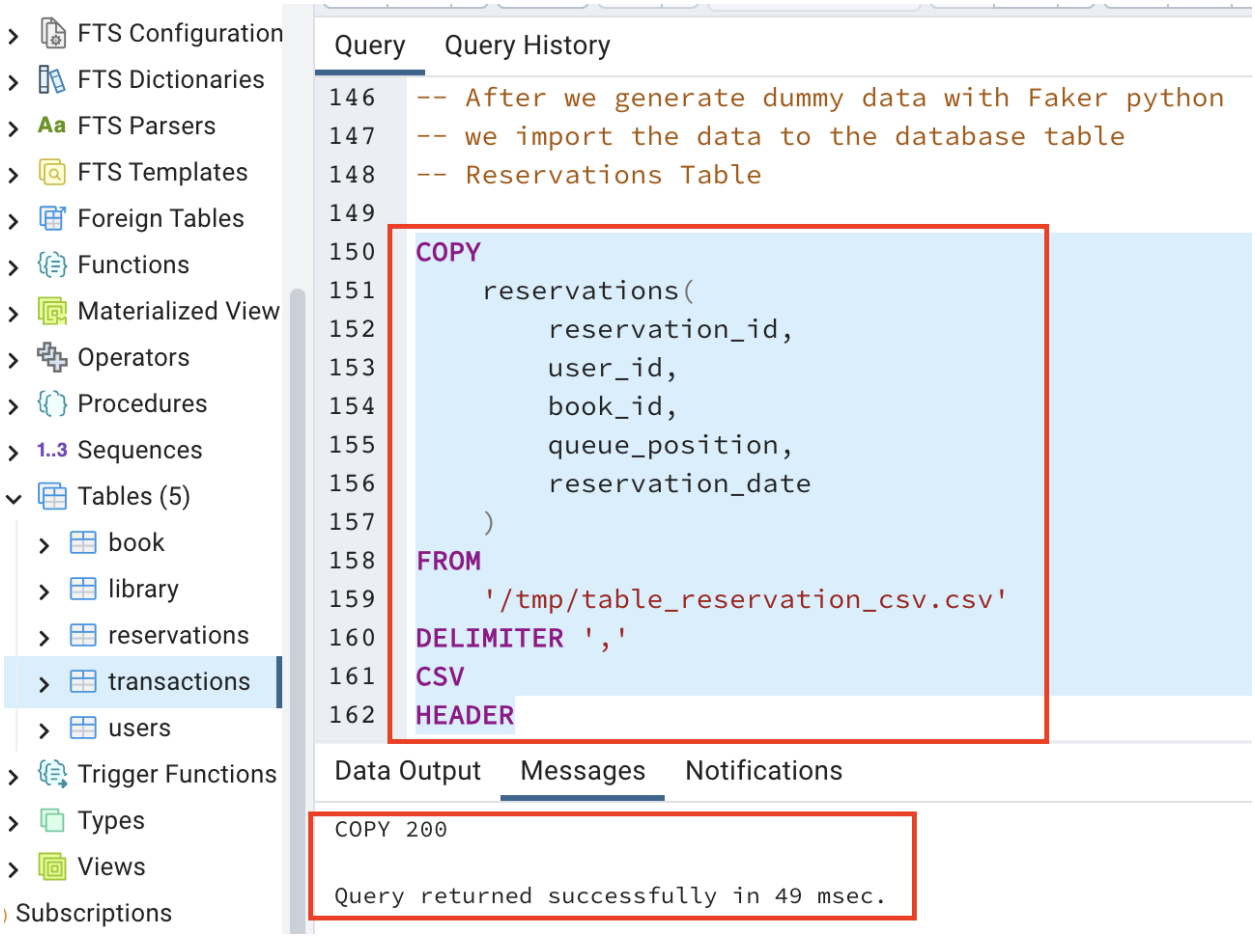

FTS Configuration
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized View
Operators
Procedures
Sequences
Tables (5)
book
library
reservations
transactions
users
Trigger Functions
Types
Views
Subscriptions
ostgres
Query
Query History
130 COPY
131 transactions(
132 transaction_id,
133 user_id,
134 book_id,
135 transaction_type,
136 loan_date,
137 due_date,
138 return_date
139)
140 FROM
141 '/tmp/table_transaction_csv.csv'
142 DELIMITER ','
143 CSV
144 HEADER
145
146 -- After we generate dummy data with Faker python
Data Output
Messages
Notifications
COPY 500
Query returned successfully in 71 msec.

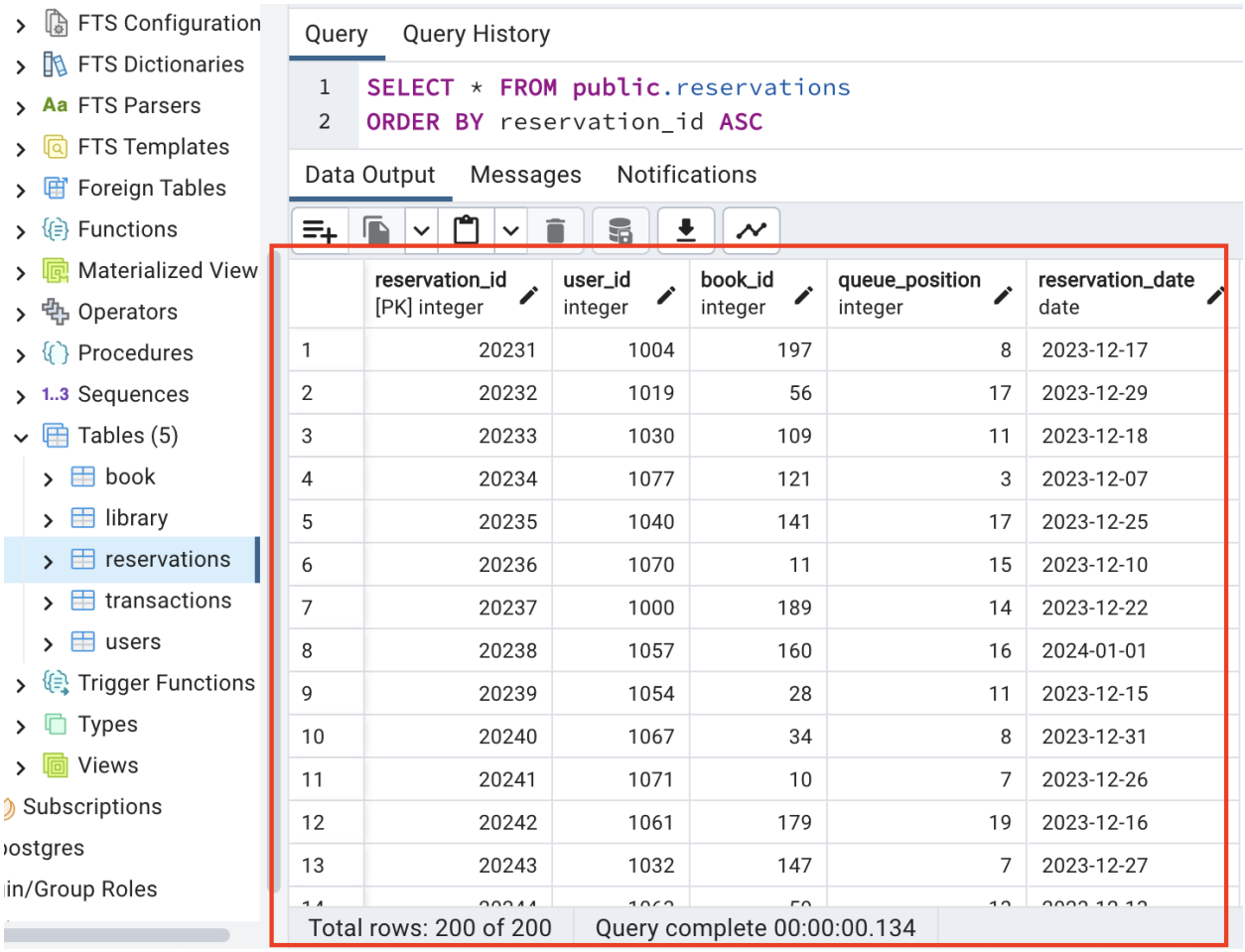


Picture 1.8 import data transactions table

Import dummy data for Reservations table with Faker python.

```
-- After we generate dummy data with Faker python
-- we import the data to the database table
-- Reservations Table
COPY
    reservations(
        reservation_id,
        user_id,
        book_id,
        queue_position,
        reservation_date
    )
FROM
    '/tmp/table_reservation_csv.csv'
DELIMITER ','
CSV
HEADER
```


FTS Configuration
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized View
Operators
Procedures
Sequences
Tables (5)
book
library
reservations
transactions
users
Trigger Functions
Types
Views
Subscriptions
Query
Query History
-- After we generate dummy data with Faker python
-- we import the data to the database table
-- Reservations Table
COPY
reservations(
reservation_id,
user_id,
book_id,
queue_position,
reservation_date
)
FROM
'/tmp/table_reservation_csv.csv'
DELIMITER ','
CSV
HEADER
Data Output
Messages
Notifications
COPY 200
Query returned successfully in 49 msec.



Picture 1.9 import data reservations table

5 Objective about the table inside e-library Database

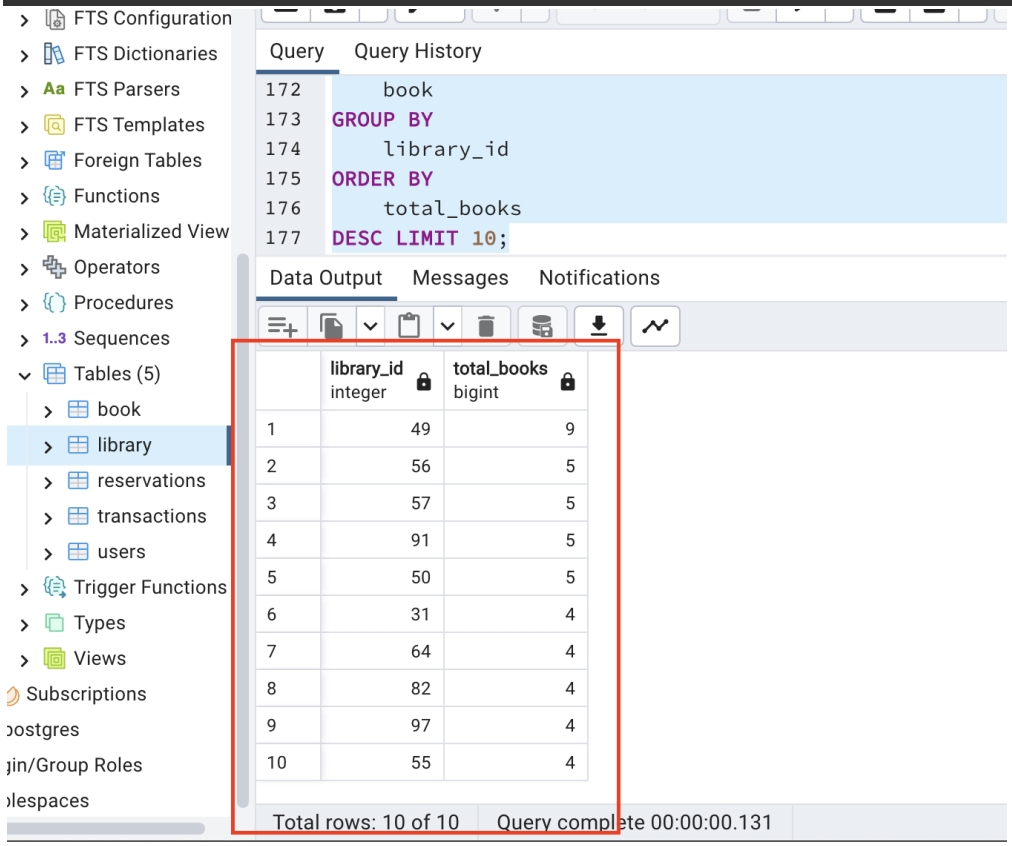
1. ***Question*: Which library has the highest number of books available?**

For answering the question above. We can find which library has highest books available from counting the book_id from book table. And for which library we can select from library_id. Because in the book table the library_id are a foreign key from library table.

Why this question important?

- Because after we can see which one the library has the highest book available. We can distribute another books to other Library. So we can equalize the distribution of all books to avoid piling up in in one library. This question can be useful information for the management of library application.

```
SELECT
    library_id, COUNT(book_id) AS total_books
FROM
    book
GROUP BY
    library_id
ORDER BY
    total_books
DESC LIMIT 1;
```



Picture 2.0 Question 1

Analyze the results:

- After we see the results from query, the highest total books are library_id 49, 56, 57 which is 9, 5, 5 books.
- From the data above we can summary that the distribution of the books still not equalize. The management of application should equalize the distribution of the books to another library except the library_id 49.

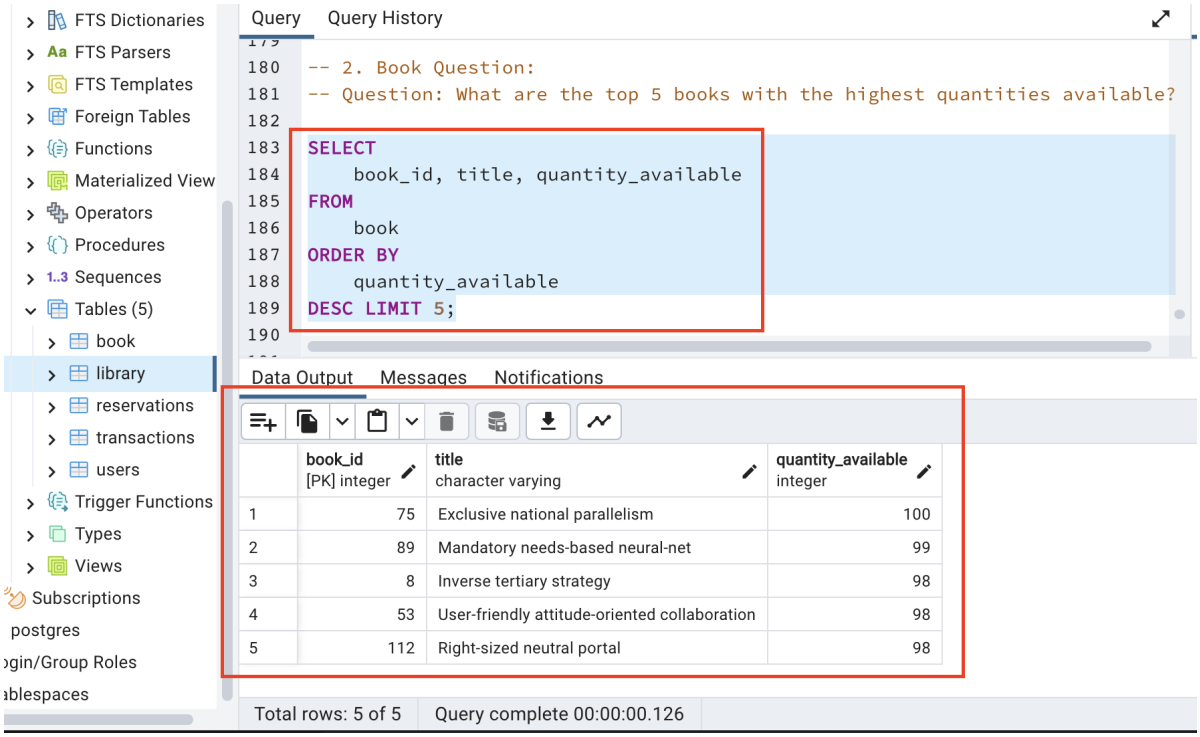
2. ***Question:*** **What are the top 5 books with the highest quantities available?**

For answering the question above. We can find which book has the highest quantity from selecting the quantity_available column from book table. And we also can include title, and book_id column for giving more insightful information.

Why this question important?

- Because after we seeing the top 5 books that available for reserve borrow. We can tell to the management of the e-library application to create more campaign or advertising the top 5 books, so there will be more engagement to the top 5 available books.

```
SELECT
	book_id, title, quantity_available
FROM
	book
ORDER BY
	quantity_available
DESC LIMIT 5;
```



Picture 2.1 Question 2

Analyze the results:

- After we see the results from query, the highest total books available are book_id *"Exclusive national parallelism" (Book ID: 75), "Mandatory needs-based neural-net" (Book ID: 89), "Inverse tertiary strategy" (Book ID: 8), "User-friendly attitude-oriented collaboration" (Book ID: 53), "Right-sized neutral portal" (Book ID: 112)*

- From the top 5 book above. The management team can create more campaign and advertisement to increasing the customer to reserve borrowing the book.

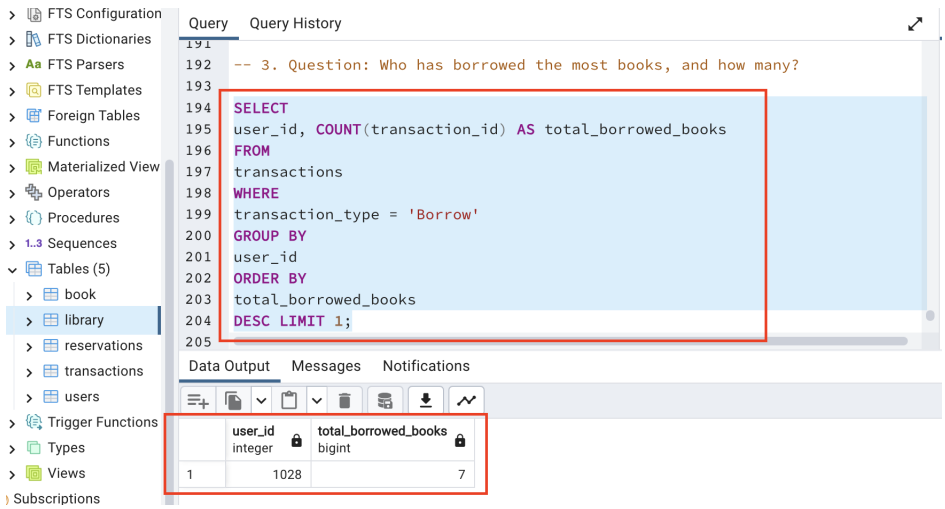
3. ***Question*: Who has borrowed the most books, and how many?**

For answering the question above. We can find which user_id has the highest quantity of borrowing book from counting the transaction_id and selecting the user_id. And grouping by the user_id and order by counted transaction_id from transaction table.

Why this question important?

- Because sometimes the management team can give some discount or voucher to the loyal customer who has the highest transaction activity. From the result of this question the management team can decide which customer will be given the prizes.

```sql
SELECT
user_id, COUNT(transaction_id) AS total_borrowed_books
FROM
transactions
WHERE
transaction_type = 'Borrow'
GROUP BY
user_id
ORDER BY
total_borrowed_books
DESC LIMIT 1;
```



Picture 2.2 Question 3

Analyze the results:

- After we see the results from query, the most user_id who borrowed books are *"jennifertucker", (user_id: 1028) 7 books, "dana58" (user_id: 1092) 6 books, "ryan08" (user_id: 1066) 6 books, "rvelazquez" (user_id: 1020) 5 books, "uwashington" (user_id: 1060) 5 books.*
- From the result above the management can pick who will they send a discount or prizes because they are loyal to the company.

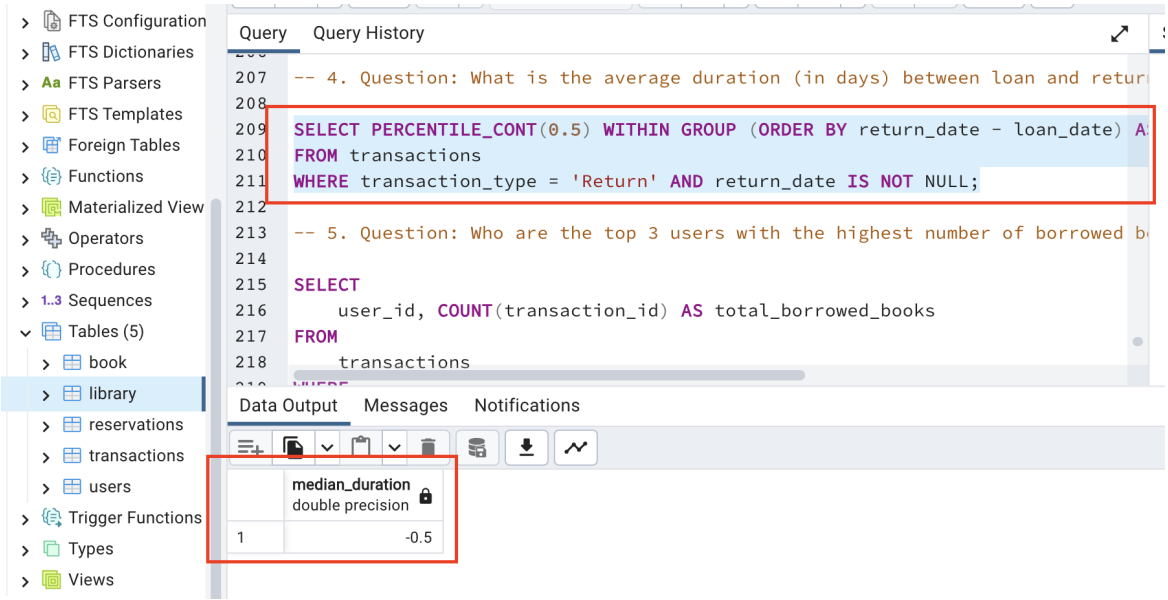
4. ***Question*: What is the average duration (in days) between loan and return dates for returned books?**

For answering the question above. We can find the average duration between borrow and return using percentile_cont and then grouping return_date - loan_date.

Why this question important?

- Through this calculation, the management of the e-library app can derive valuable insights. For instance, by understanding the typical duration for which people borrow books, the management team can adjust or establish new due dates for book loans. Additionally, if there is data available on the number of pages in each book, it can be used to calculate the average time it takes to finish one book based on the page count.

```
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY return_date - loan_date) AS
median_duration
FROM transactions
WHERE transaction_type = 'Return' AND return_date IS NOT NULL;
```



Picture 2.3 Question 4

Analyze the results:

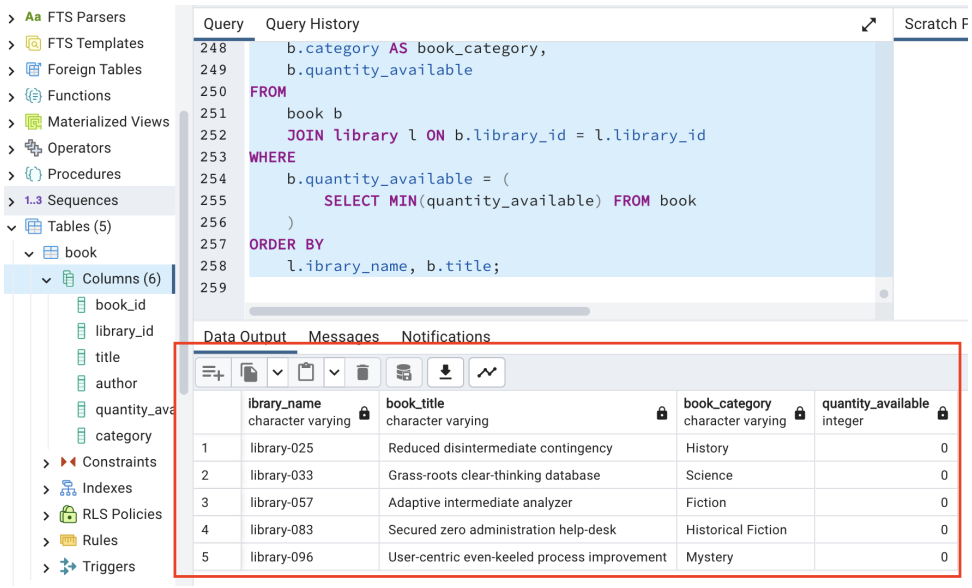
- After we see the results from query. The median value between return_date and loan_date is 0.5
- The median value above maybe have some kind problem. Because there is still missing data in the return_date value.

5. ***Question: Which library has the lowest quantity of available books, and what are the titles of those books?***

Why this question important?

- By analyzing the quantity of available books, we can gain insights into which book categories are customer favorites. When the quantity of a book is low, it indicates high user demand. In such cases, the management team can identify popular books and consider adding more of them to the application to boost sales.

```sql
SELECT
    l.ibrary_name,
    b.title AS book_title,
    b.category AS book_category,
    b.quantity_available
FROM
    book b
    JOIN library l ON b.library_id = l.library_id
WHERE
    b.quantity_available = (
        SELECT MIN(quantity_available) FROM book
    )
ORDER BY
    l.ibrary_name, b.title;
```



Picture 2.4 Question 5

Analyze the results:

- After we see the results from query. The result are
  "Reduced disintermediate contingency" "History"
  "Grass-roots clear-thinking database" "Science"
  "Adaptive intermediate analyzer" "Fiction"
  "Secured zero administration help-desk" "Historical Fiction"
  "User-centric even-keeled process improvement" "Mystery"
- *The management of application can add more of the book Category that shown in the table above to get more sales on the application.*

References

Thanks to LMS Pacmann for the tutorial about how to generate dummy data with Faker and how to use Faker and visualize in table with Tabulate and export to CSV File and import/copy it to the table inside the database. And thanks to chatGPT that helps me to correct many of my sql syntax about how to create table with specific constraints and key. And thanks to drawio for the helps to create a clear diagram that shows ERD Database.

And thank you also from websites:

1. https://faker.readthedocs.io/en/master/index.html
2. https://pacmann.io/course/class-detail/805
3. https://app.diagrams.net/#HCakurniawan%2Fpacmann-sql%2Fmain%2FDiagram%20Tanpa%20Judul.drawio
4. https://chat.openai.com/

Project Repository

For this project files and supporting document. I store all in the github repository

1. https://github.com/Cakurniawan/pacmann-sql