For this assignment, you will need to create a database in PostgreSQL.

Your database will contain information about a bookstore. You can choose any name for the database that suits you. For example, the name could be **bookstore** :)

After you create the database, create tables inside it. To do this, use the queries in the **Tables.sql** file.

The provided SQL script outlines a small relational database designed for a **bookstore**. The database consists of five key tables:

1. **Genres**: This table stores information about the different genres of books available in the bookstore. Each genre is uniquely identified by **genre_id** and includes the **genre_name**.
2. **Authors**: This table captures details about the authors whose books are available in the bookstore. It includes fields such as **author_id**, **name**, and **date_of_birth** to uniquely identify and describe each author.
3. **Books**: This table contains data about the books in the bookstore, including **book_id**, **title**, **author_id**, **published_date**, **genre_id**, and **price**. The table also establishes relationships with the **Authors** and **Genres** tables through foreign keys, linking each book to its respective author and genre.
4. **Customers**: This table records information about the customers who purchase books from the bookstore. Each customer is identified by **customer_id** and has associated details such as **first_name**, **last_name**, **email**, and **join_date**.
5. **Sales**: This table tracks each sale transaction in the bookstore. It includes fields like **sale_id**, **book_id**, **customer_id**, **quantity**, and **sale_date**, connecting purchases with specific books and customers through foreign keys.

For full details, see the **Tables.sql** file.

We would like to emphasize that this is a simplified database. And we use it only for **demonstration and training purposes**.

After creating the tables, fill them with data. To fill the tables with data, use the queries in the **Data.sql** file.

If you need additional data in tables or new tables to solve the tasks, feel free to add them.

Do all the tasks from the assignment on the created database.

Task 1: Identify Authors with the Most Published Books

Create a CTE that calculates the total number of books each author has published. Then, create SELECT query that will use this CTE, and retrieve a list of authors who have published more than 3 books, including the number of books they have published.

Task 2: Identify Books with Titles Containing 'The' Using Regular Expressions

Create a CTE that identifies books whose titles contain the word "The" in any letter case using regular expressions. For example, books with titles like "The Great Gatsby," "The Shining", "The Old Man and the Sea", or "To the Lighthouse" will meet this criterion. Then, create the SELECT that will this CTE, and retrieve the book titles, corresponding authors, genre, and publication date.

Task 3: Rank Books by Price within Each Genre Using the RANK() Window Function

Create a query that **ranks books by their price within each genre using the RANK() window function**. The goal is to assign a rank to each book based on its price, with the highest-priced book in each genre receiving a rank of 1.

Task 4: Bulk Update Book Prices by Genre

Create a stored procedure that increases the prices of all books in a specific genre by a specified percentage. The procedure should also output the number of books that were updated. Use RAISE for it.

Potential procedure name: sp_bulk_update_book_prices_by_genre

Parameters for procedure:

p_genre_id INTEGER

p_percentage_change NUMERIC(5, 2)

Example of usage:

-- This increases the prices of all books in genre ID 3 by 5% and output the updated number

CALL sp_bulk_update_book_prices_by_genre(3, 5.00);

Task 5: Update Customer Join Date Based on First Purchase

Create a stored procedure that updates the **join_date** of each customer to the **date of their first purchase if it is earlier** than the current **join_date**. This ensures that the **join_date** reflects the true start of the customer relationship.

Potential procedure name: sp_update_customer_join_date

Parameters for procedure: NONE

Example of usage:

-- This updates the join dates of customers to reflect the date of their first purchase if earlier than

-- the current join date.

CALL sp_update_customer_join_date();

Task 6: Calculate Average Book Price by Genre

Create a function that calculates the average price of books within a specific genre.

Potential function name: fn_avg_price_by_genre

Parameters: p_genre_id INTEGER

Return Type: NUMERIC(10, 2)

Example of usage:

-- This would return the average price of books in the genre with ID 1.

SELECT fn_avg_price_by_genre(1);

Task 7: Get Top N Best-Selling Books by Genre

Create a function that returns the top N best-selling books in a specific genre, based on total sales revenue.

Potential function name: fn_get_top_n_books_by_genre

Parameters:

p_genre_id INTEGER

p_top_n INTEGER

Example of usage:

-- This would return the top 5 best-selling books in genre with ID 1

```
SELECT * FROM fn_get_top_n_books_by_genre(1, 5);
```

Task 8: Log Changes to Sensitive Data

Create a trigger that logs any changes made to sensitive data in a Customers table. Sensitive data: first name, last name, email address. The trigger should insert a record into an audit log table each time a change is made. You need to create this log table by yourself.

Log table could have such structure:

```
CREATE TABLE CustomersLog (
  log_id SERIAL PRIMARY KEY,
  column_name VARCHAR(50),
  old_value TEXT,
  new_value TEXT,
  changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  changed_by VARCHAR(50) -- This assumes you can track the user making the change
);
```

Potential trigger name: tr_log_sensitive_data_changes

Trigger Timing: AFTER UPDATE

Trigger Event: ON table Customers

Task 9: Automatically Adjust Book Prices Based on Sales Volume

Create a trigger that automatically **increases the price of a book by 10%** if the **total quantity sold reaches a certain threshold** (e.g., 10 units). This helps to dynamically adjust pricing based on the popularity of the book.

Potential trigger name: tr_adjust_book_price

Trigger Timing: AFTER INSERT

Trigger Event: ON table Sales

Task 10: Archive Old Sales Records

Create a stored procedure that **uses a cursor to iterate over sales records older than a specific date, move them to an archive table (SalesArchive), and then delete them from the original Sales table**.

Potential procedure name: sp_archive_old_sales

Parameters: p_cutoff_date DATE

Potential Steps:

1. Declare a cursor to fetch sales records older than a given date.
2. For each record, insert it into the **SalesArchive** table.
3. After inserting record to **SalesArchive** table, delete the record from the **Sales** table.

The procedure should take the cutoff date as an input parameter.

For this task, you will need to create a table called SalesArchive. The structure of this table will be the same as the Sales table.