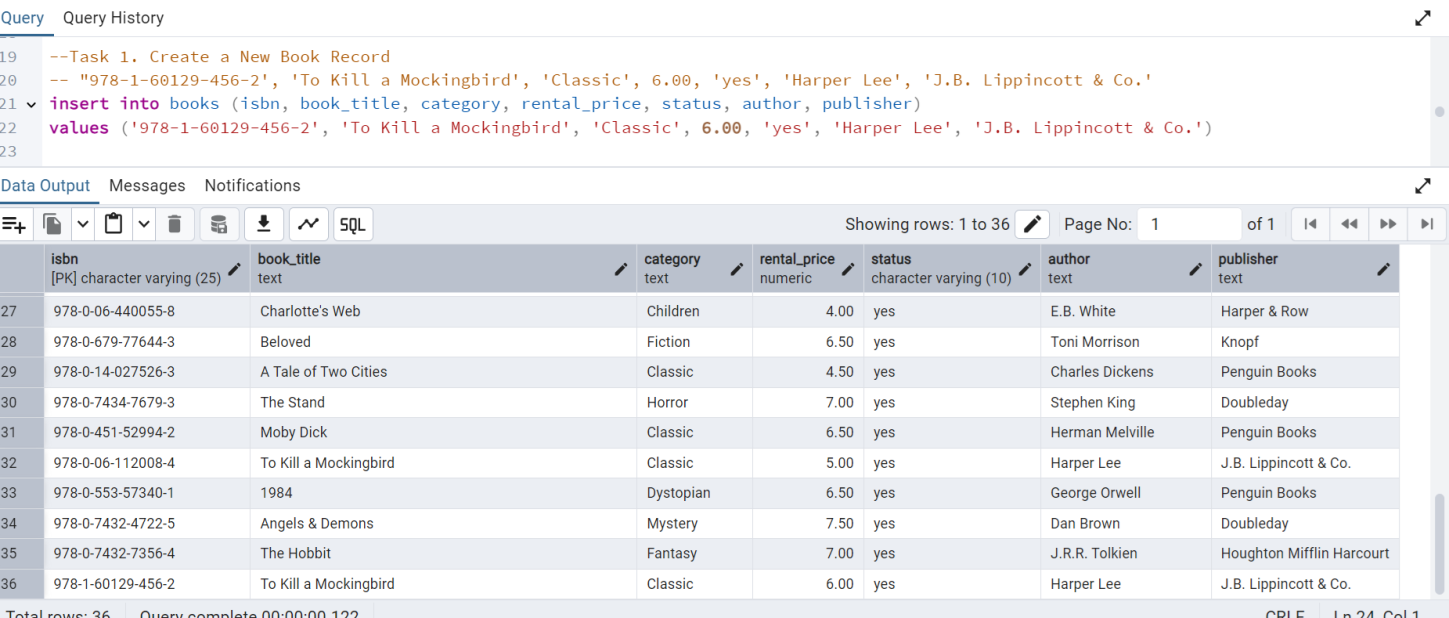
**CRUD Operations**

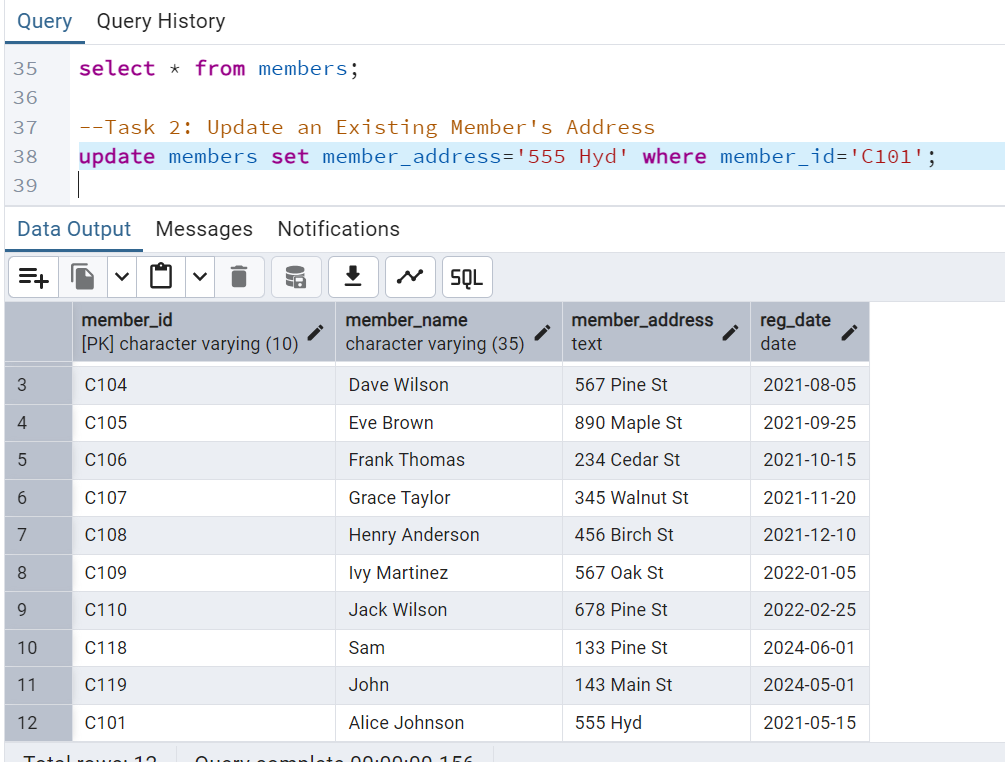
**Task 1**. Create a New Book Record -- "978-1-60129-456-2', 'To Kill a Mockingbird', 'Classic', 6.00, 'yes', 'Harper Lee', 'J.B. Lippincott & Co.'

**🡪 insert into books (isbn, book\_title, category, rental\_price, status, author, publisher) values ('978-1-60129-456-2', 'To Kill a Mockingbird', 'Classic', 6.00, 'yes', 'Harper Lee', 'J.B. Lippincott & Co.')**



**Task 2:** Update an Existing Member's Address

**🡪 update members set member\_address='555 Hyd' where member\_id='C101';**



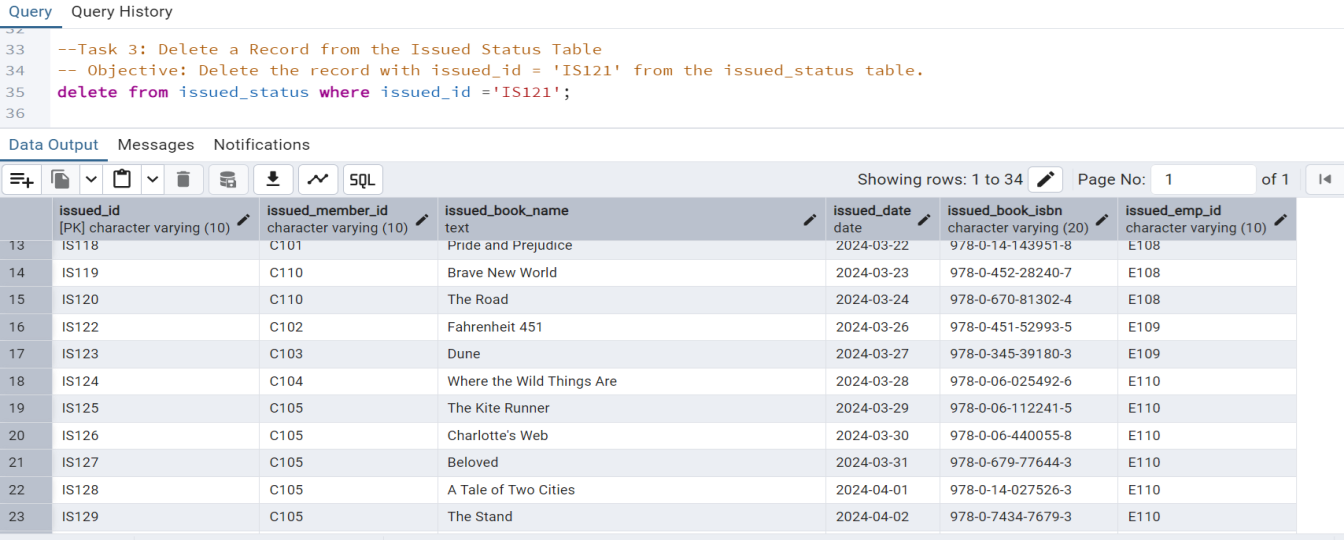
**Task 3:** Delete a Record from the Issued Status Table -- Objective: Delete the record with issued\_id = 'IS121' from the issued\_status table.

**🡪delete from issued\_status where issued\_id ='IS121';**

**Before deleting**

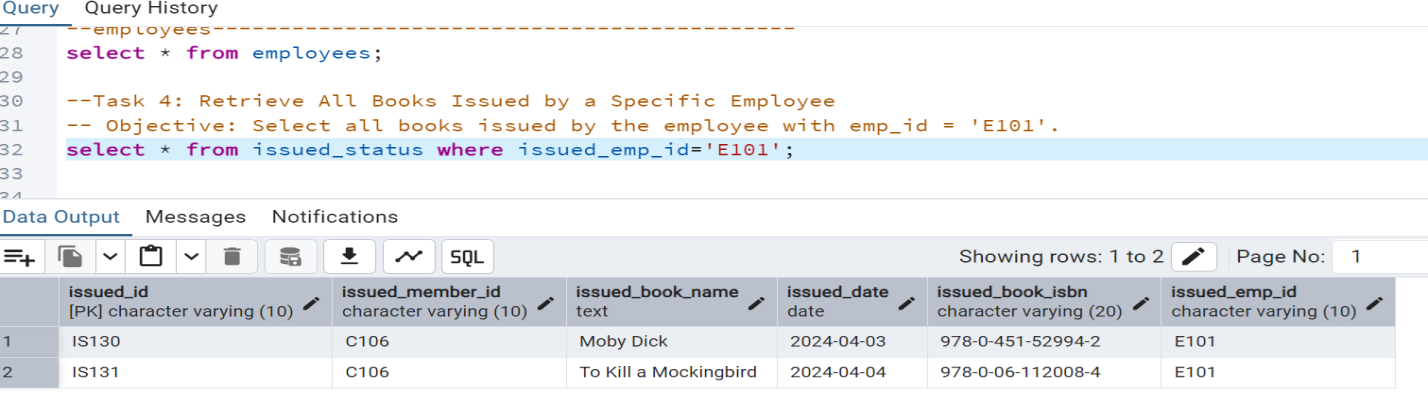
****

**After deleting**



**Task 4:** Retrieve All Books Issued by a Specific Employee -- Objective: Select all books issued by the employee with emp\_id = 'E101'.

**🡪 select \* from issued\_status where issued\_emp\_id='E101';**



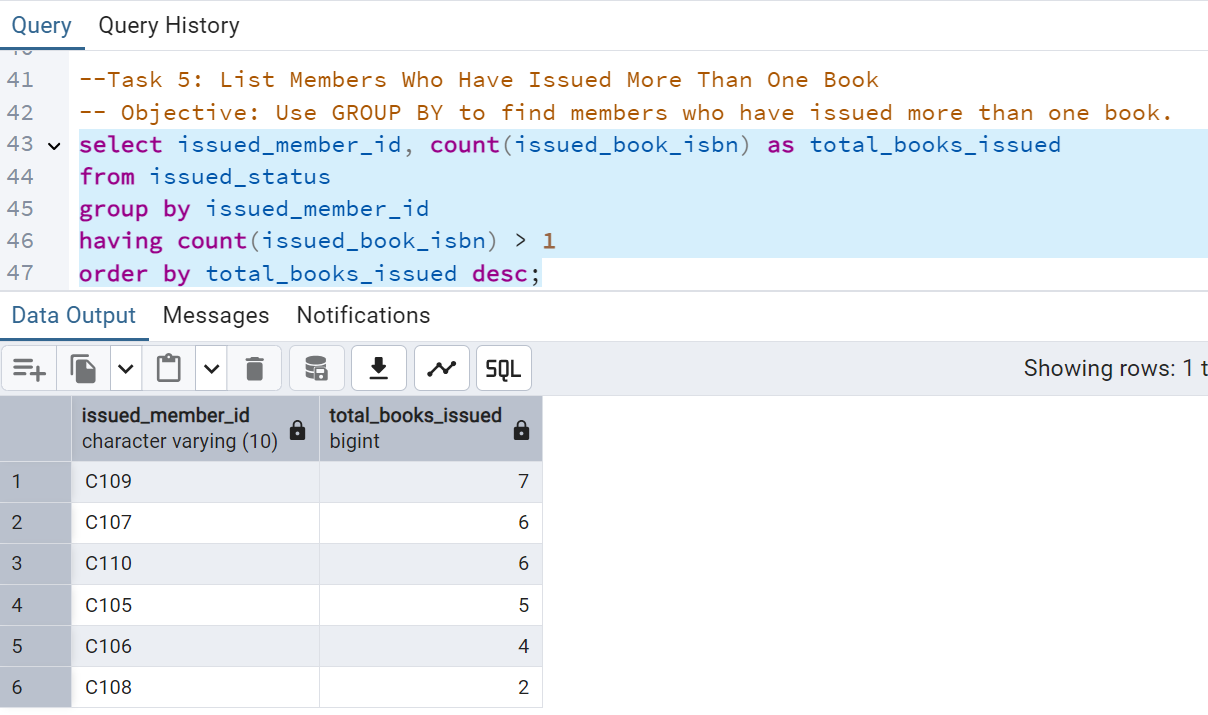
**Task 5:** List Members Who Have Issued More Than One Book -- Objective: Use GROUP BY to find members who have issued more than one book.

**🡪 select issued\_member\_id, count(issued\_book\_isbn) as total\_books\_issued**

**from issued\_status**

**group by issued\_member\_id**

**having count(issued\_book\_isbn) > 1**

**order by total\_books\_issued desc;**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CTAS (Create Table As Select)

**Task 6**: Create Summary Tables: Used CTAS to generate new tables based on query results - each book and total book\_issued\_cnt\*\*

**🡪 create table book\_issue\_summary as**

**select**

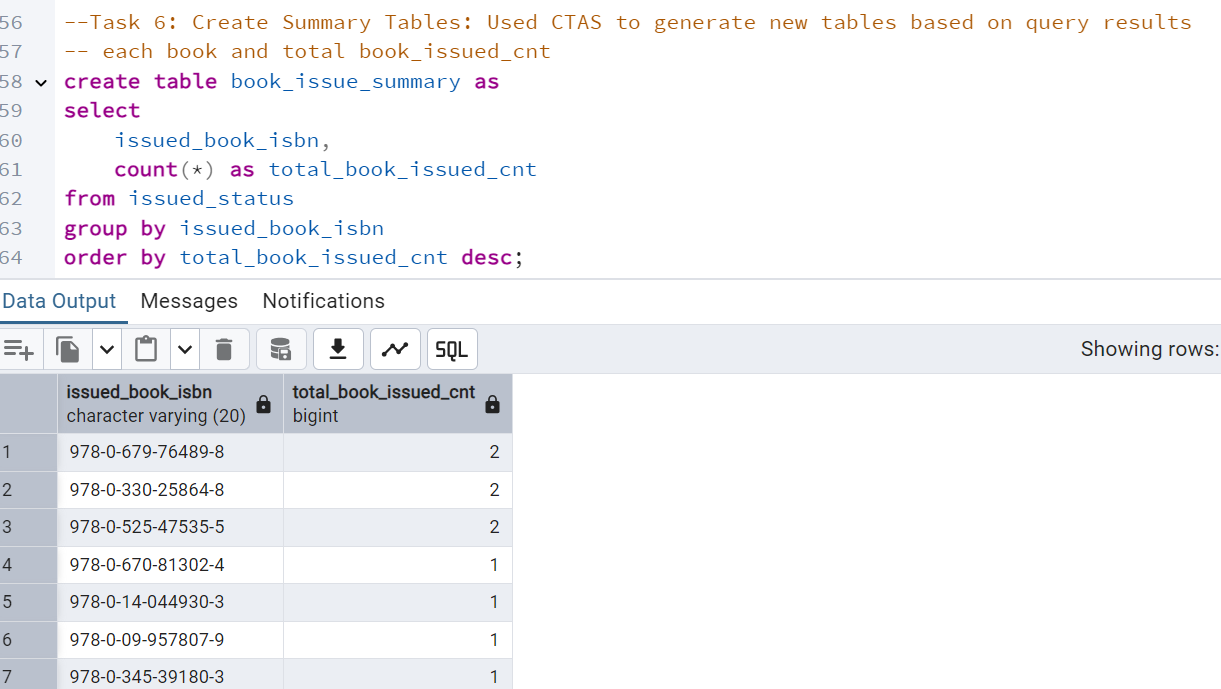
**issued\_book\_isbn,**

**count(\*) as total\_book\_issued\_cnt**

**from issued\_status**

**group by issued\_book\_isbn**

**order by total\_book\_issued\_cnt desc;**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Data Analysis & Findings**

**Task 7:** Retrieve All Books in a Specific Category:

**🡪 select \* from books where category ='Classic';**



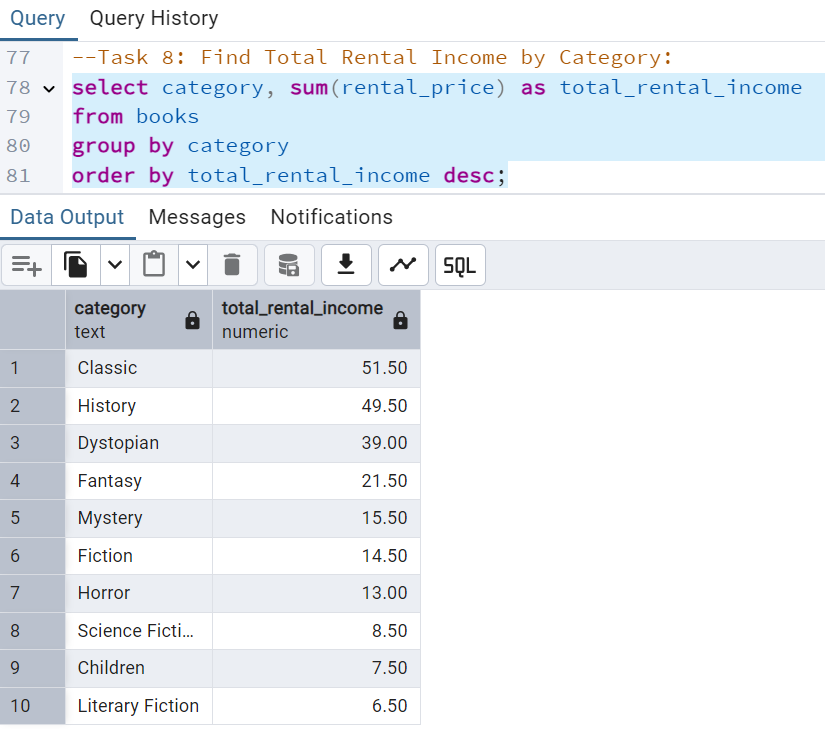
**Task 8:** Find Total Rental Income by Category:

**🡪select category, sum(rental\_price) as total\_rental\_income**

**from books**

**group by category**

**order by total\_rental\_income desc;**

****

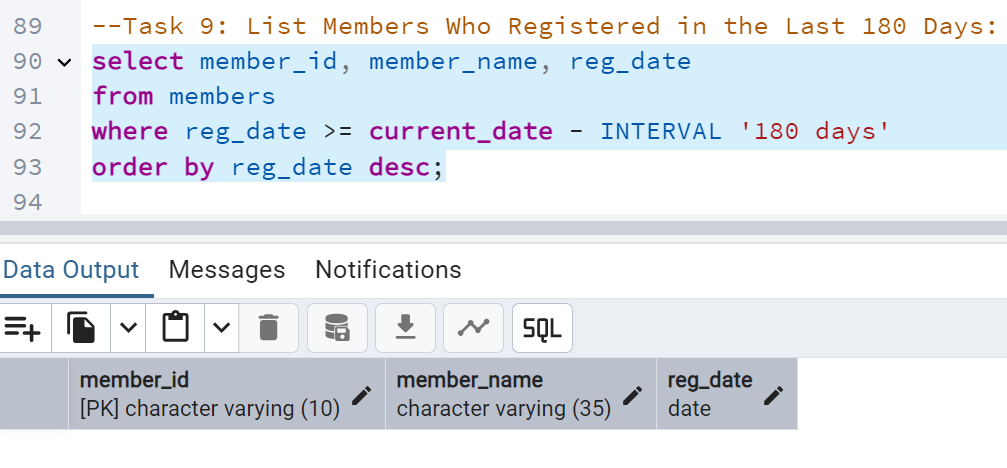
**Task 9:** List Members Who Registered in the Last 180 Days:

**🡪 select member\_id, member\_name, reg\_date**

**from members**

**where reg\_date >= current\_date - INTERVAL '180 days'**

**order by reg\_date desc;**

**Task 10:** List Employees with Their Branch Manager's Name and their branch details:

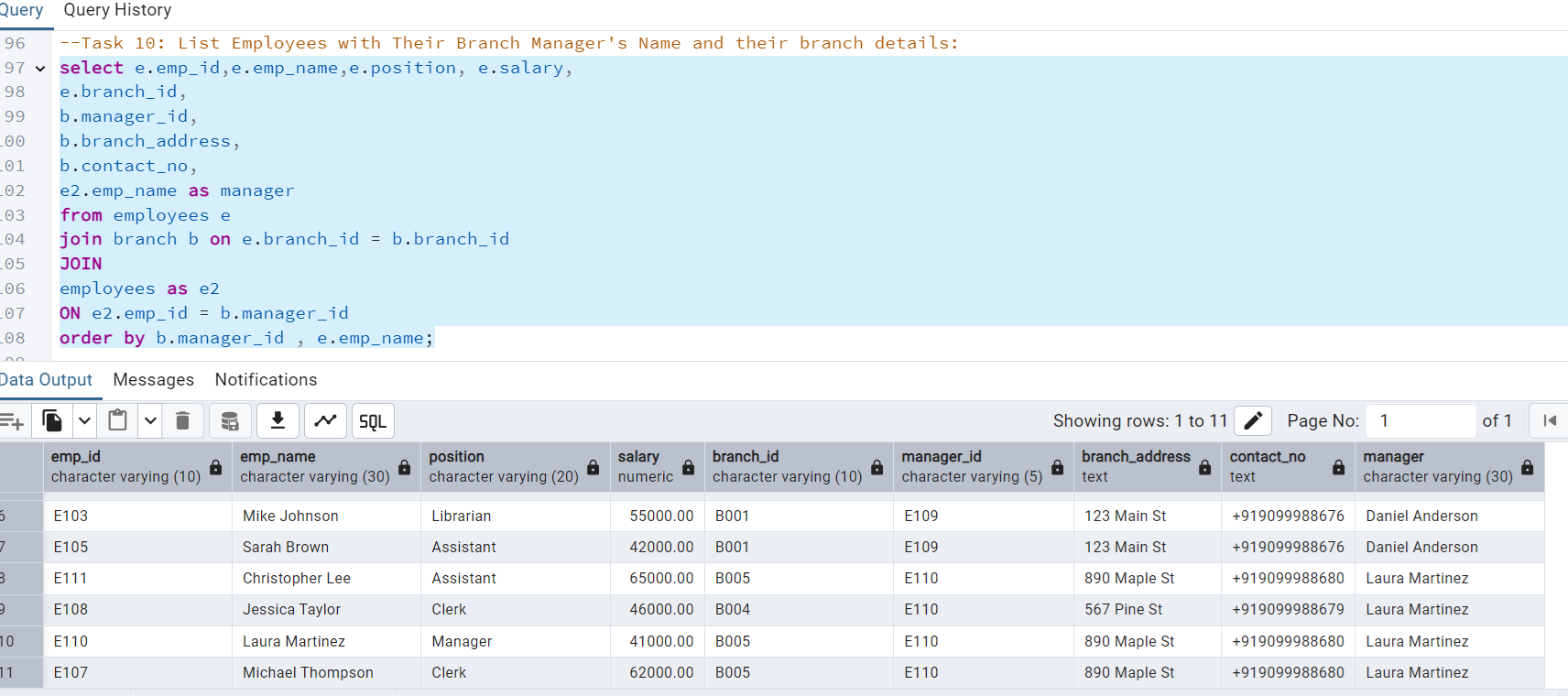
**🡪 select e.emp\_id,e.emp\_name,e.position, e.salary,**

**e.branch\_id, b.manager\_id,b.branch\_address,b.contact\_no,e2.emp\_name as manager**

**from employees e**

**join branch b on e.branch\_id = b.branch\_id**

**JOIN employees as e2 ON e2.emp\_id = b.manager\_id**

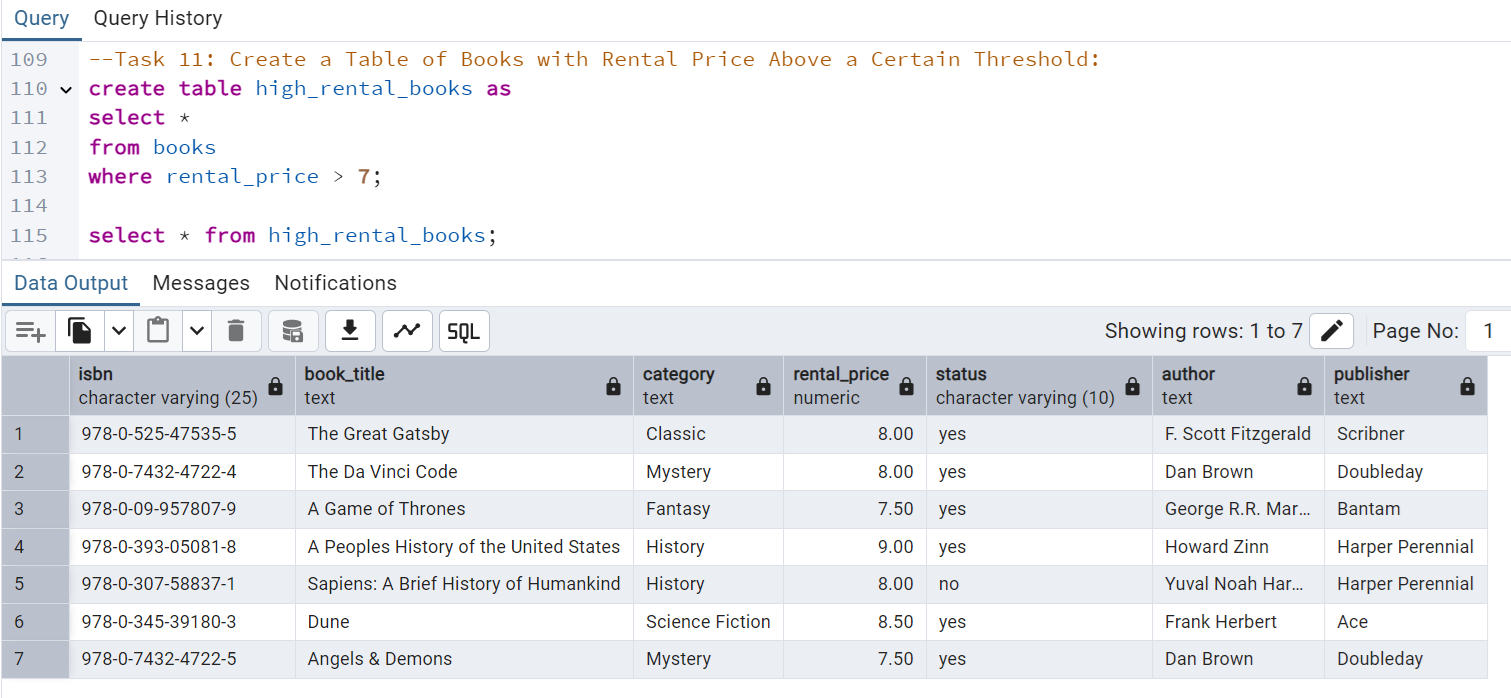
**order by b.manager\_id , e.emp\_name;**

**Task 11:** Create a Table of Books with Rental Price Above a Certain Threshold:

**🡪 create table high\_rental\_books as**

**select \* from books**

**where rental\_price > 7;**



**Task 12:** Retrieve the List of Books Not Yet Returned

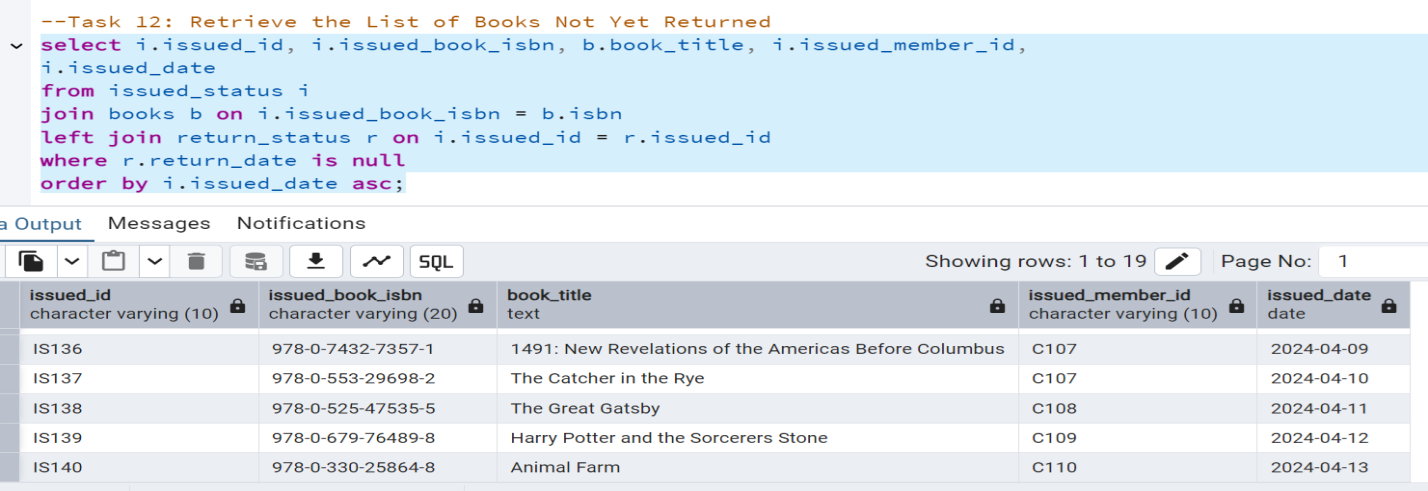
**🡪 select i.issued\_id, i.issued\_book\_isbn, b.book\_title, i.issued\_member\_id, i.issued\_date**

**from issued\_status i**

**join books b on i.issued\_book\_isbn = b.isbn**

**left join return\_status r on i.issued\_id = r.issued\_id**

**where r.return\_date is null**

**order by i.issued\_date asc;**

**Advanced SQL Operations**

**Task 13:** Identify Members with Overdue Books

Write a query to identify members who have overdue books (assume a 30-day return period). Display the member's\_id, member's name, book title, issue date, and days overdue.

**🡪** **select i.issued\_member\_id, m.member\_name, b.book\_title, i.issued\_date,**

**current\_date - i.issued\_date as days\_overdue**

**from issued\_status i**

**join books b on i.issued\_book\_isbn = b.isbn**

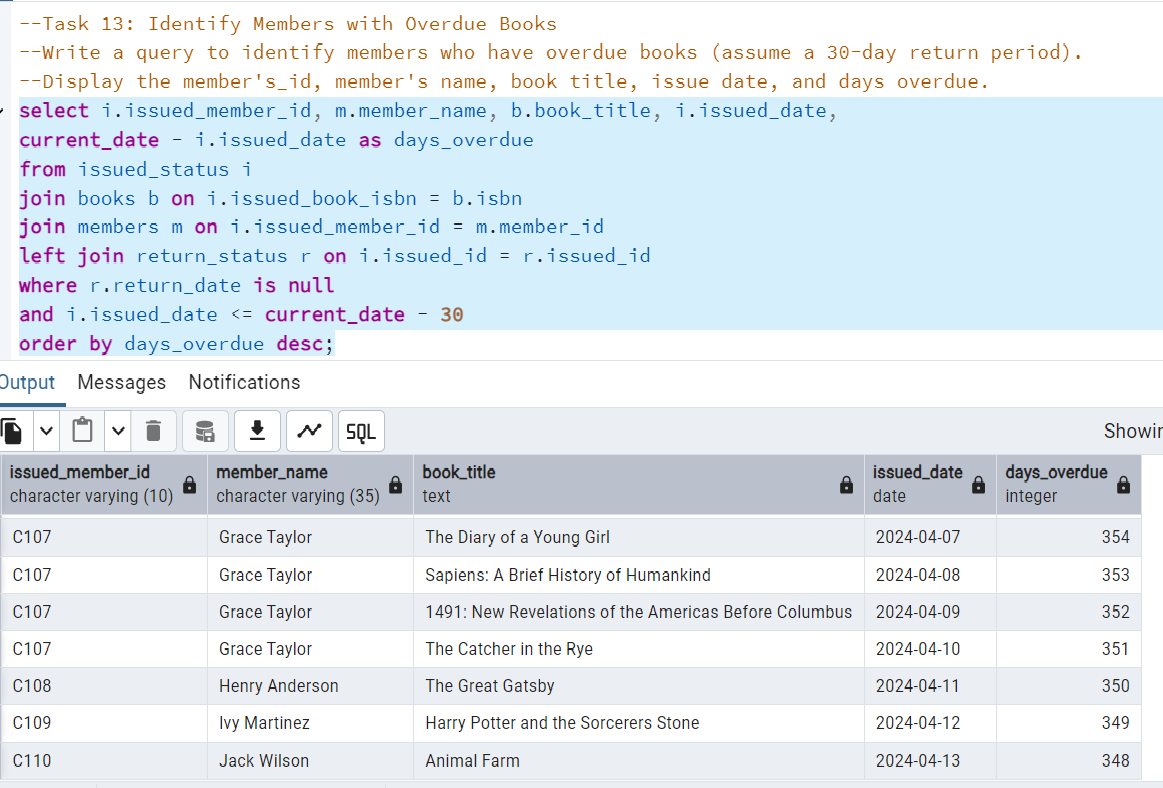
**join members m on i.issued\_member\_id = m.member\_id**

**left join return\_status r on i.issued\_id = r.issued\_id**

**where r.return\_date is null**

**and i.issued\_date <= current\_date - 30**

**order by days\_overdue desc;**



**Task 14:** Update Book Status on Return

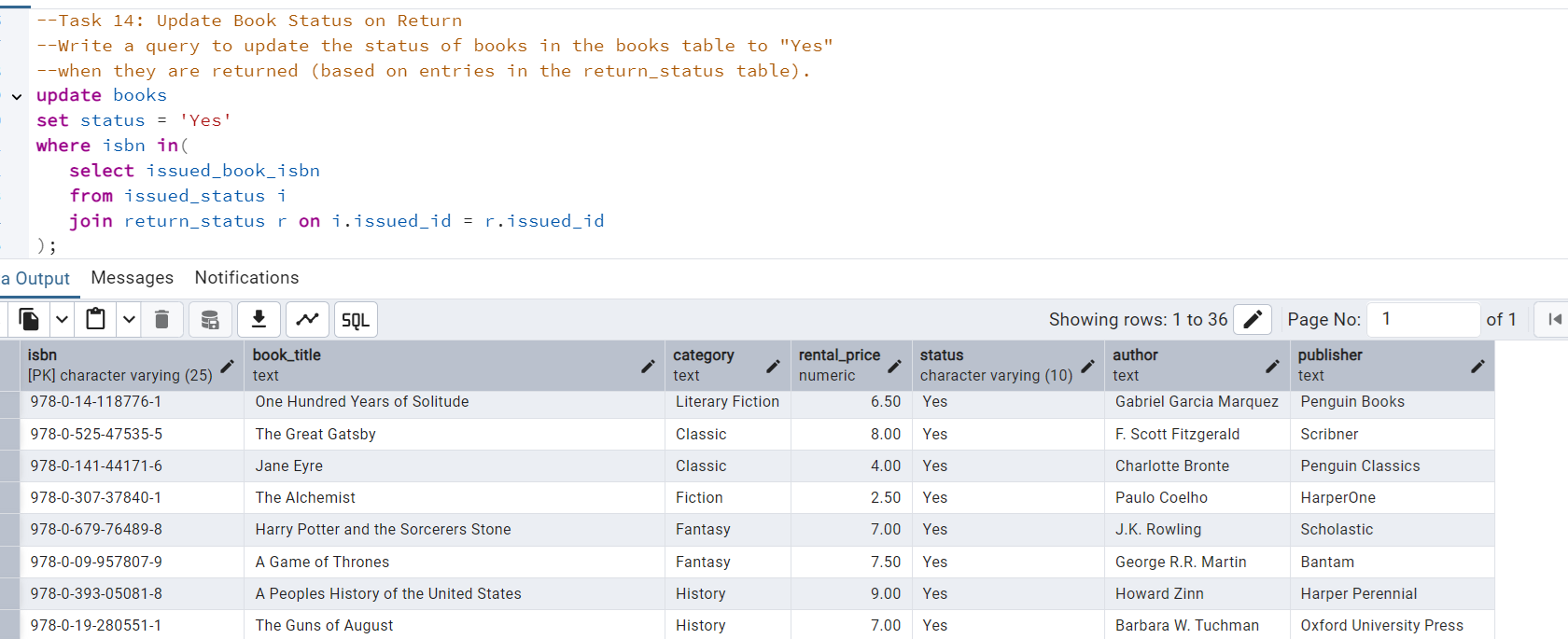
Write a query to update the status of books in the books table to "Yes" when they are returned (based on entries in the return\_status table).

**🡪 update books set status = 'Yes'**

**where isbn in( select issued\_book\_isbn**

**from issued\_status i**

**join return\_status r on i.issued\_id = r.issued\_id );**



**Task 15:** Branch Performance Report

Create a query that generates a performance report for each branch, showing the number of books issued, the number of books returned, and the total revenue generated from book rentals.

**🡪 select**

**b.branch\_id,b.branch\_address,**

**count(distinct i.issued\_id) as total\_books\_issued,**

**count(distinct r.issued\_id) as total\_books\_returned,**

**coalesce(sum(bk.rental\_price), 0)as total\_revenue**

**from branch b**

**left join employees e on b.branch\_id = e.branch\_id**

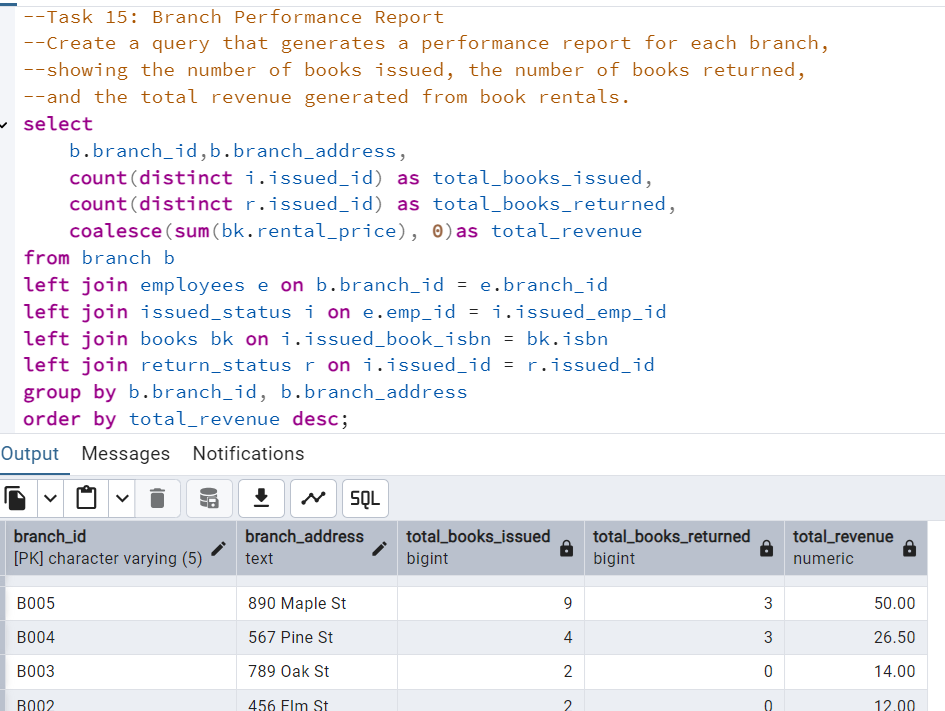
**left join issued\_status i on e.emp\_id = i.issued\_emp\_id**

**left join books bk on i.issued\_book\_isbn = bk.isbn**

**left join return\_status r on i.issued\_id = r.issued\_id**

**group by b.branch\_id, b.branch\_address**

**order by total\_revenue desc;**



**Task 16:** CTAS: Create a Table of Active Members

Use the CREATE TABLE AS (CTAS) statement to create a new table active\_members containing members who have issued at least one book in the last 2 months.

**🡪 create table active\_members as**

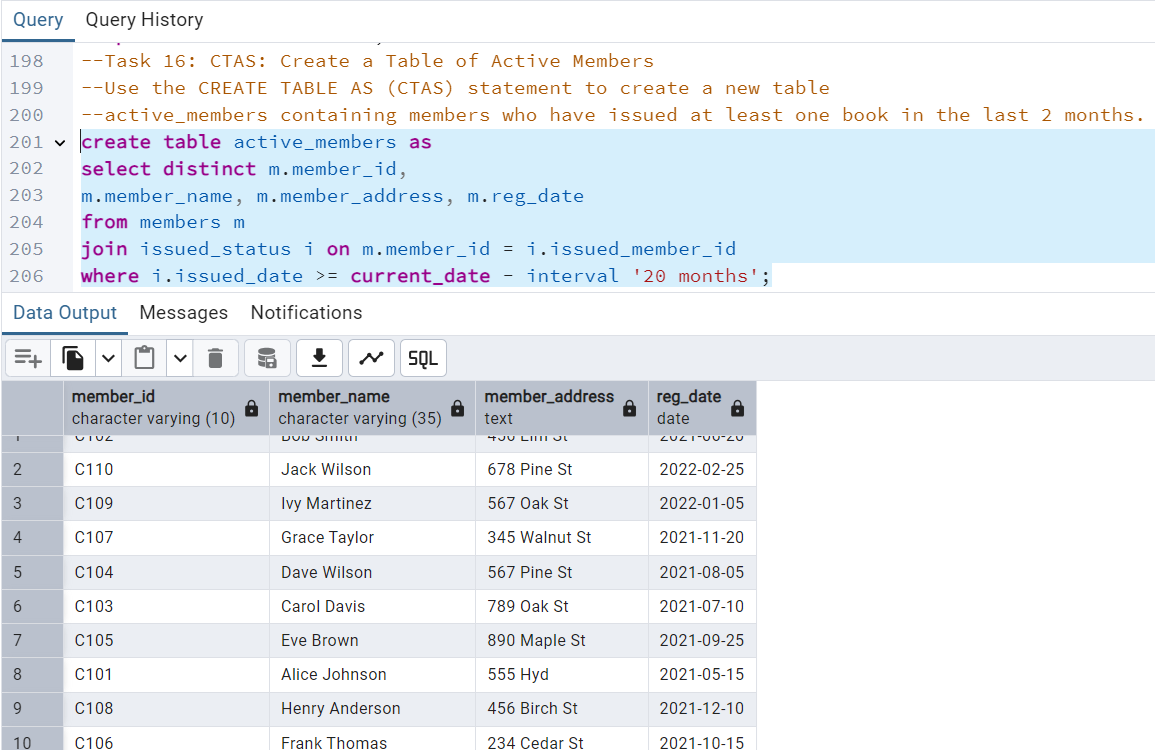
**select distinct m.member\_id,**

**m.member\_name, m.member\_address, m.reg\_date**

**from members m**

**join issued\_status i on m.member\_id = i.issued\_member\_id**

**where i.issued\_date >= current\_date - interval '20 months';**



**Task 17:** Find Employees with the Most Book Issues Processed

Write a query to find the top 3 employees who have processed the most book issues. Display the employee name, number of books processed, and their branch.

**🡪 select e.emp\_name,**

**e.branch\_id,**

**count(i.issued\_id) as books\_processed**

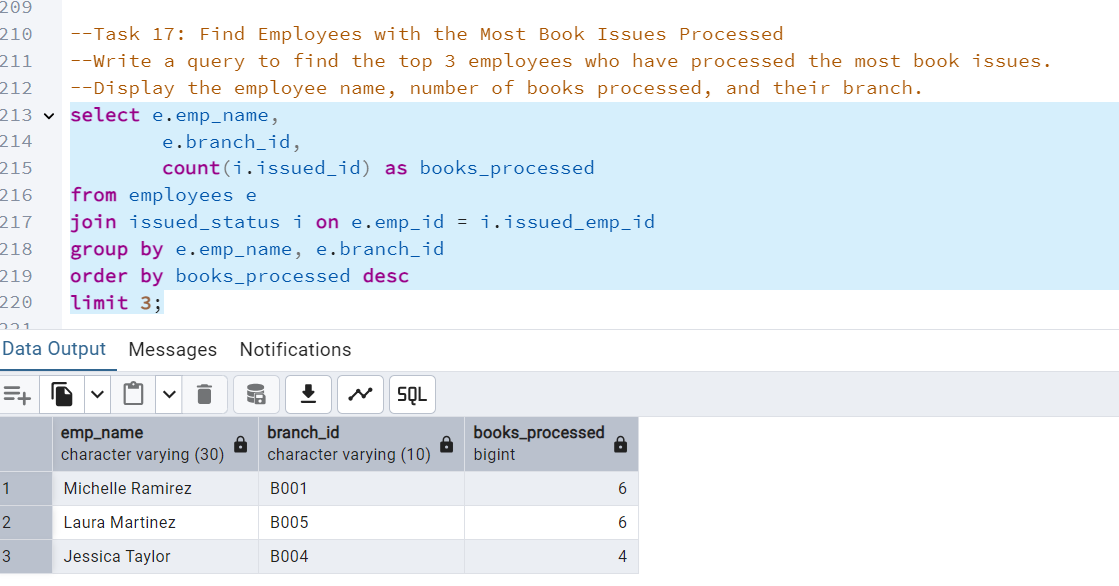
**from employees e**

**join issued\_status i on e.emp\_id = i.issued\_emp\_id**

**group by e.emp\_name, e.branch\_id**

**order by books\_processed desc**

**limit 3;**



**Task 18:** Identify Members Issuing High-Risk Books

Write a query to identify members who have issued books more than twice with the status "damaged" in the books table. Display the member name, book title, and the number of times they've issued damaged books.

**Task 19:** Stored Procedure Objective: Create a stored procedure to manage the status of books in a library system. Description: Write a stored procedure that updates the status of a book in the library based on its issuance. The procedure should function as follows: The stored procedure should take the book\_id as an input parameter. The procedure should first check if the book is available (status = 'yes'). If the book is available, it should be issued, and the status in the books table should be updated to 'no'. If the book is not available (status = 'no'), the procedure should return an error message indicating that the book is currently not available.

**Task 20:** Create Table As Select (CTAS) Objective: Create a CTAS (Create Table As Select) query to identify overdue books and calculate fines.

**Description:** Write a CTAS query to create a new table that lists each member and the books they have issued but not returned within 30 days. The table should include: The number of overdue books. The total fines, with each day's fine calculated at $0.50. The number of books issued by each member. The resulting table should show: Member ID Number of overdue books Total fines