Project-SQL-Rooman

-- Task 1: Library Management System

/*

Project Description:

Design and develop a Library Management System using SQL. The system manages book

inventories, member details, and borrowing transactions using three tables:

Books, Members, and BorrowingRecords.

*/

/* Database Setup:

CREATE DATABASE library_management;

\c library_management

*/

```
CREATE TABLE Books (
    BOOK_ID SERIAL PRIMARY KEY,
    TITLE VARCHAR(100) NOT NULL,
    AUTHOR VARCHAR(100) NOT NULL,
    GENRE VARCHAR(50),
    YEAR_PUBLISHED INTEGER CHECK (YEAR_PUBLISHED > 0),
    AVAILABLE_COPIES INTEGER CHECK (AVAILABLE_COPIES >= 0)
);

CREATE TABLE Members (
    MEMBER_ID SERIAL PRIMARY KEY,
```

```
    NAME VARCHAR(100) NOT NULL,
    EMAIL VARCHAR(100) UNIQUE NOT NULL,
    PHONE_NO VARCHAR(15),
    ADDRESS TEXT,
    MEMBERSHIP_DATE DATE DEFAULT CURRENT_DATE
);

CREATE TABLE BorrowingRecords (
    BORROW_ID SERIAL PRIMARY KEY,
    MEMBER_ID INTEGER REFERENCES Members(MEMBER_ID),
    BOOK_ID INTEGER REFERENCES Books(BOOK_ID),
    BORROW_DATE DATE DEFAULT CURRENT_DATE,
    RETURN_DATE DATE,
    CONSTRAINT valid_dates CHECK (RETURN_DATE >= BORROW_DATE)
);

-- Data Creation:

-- Insert Books with careful genre distribution
INSERT INTO Books (TITLE, AUTHOR, GENRE, YEAR_PUBLISHED,
AVAILABLE_COPIES) VALUES
('To Kill a Mockingbird', 'Harper Lee', 'Fiction', 1960, 3),
('1984', 'George Orwell', 'Science Fiction', 1949, 4),
('Pride and Prejudice', 'Jane Austen', 'Romance', 1813, 2),
('The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 1937, 6),
('The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925, 5),
('Dune', 'Frank Herbert', 'Science Fiction', 1965, 3),
```

('The Catcher in the Rye', 'J.D. Salinger', 'Fiction', 1951, 4);

-- Insert Members

INSERT INTO Members (NAME, EMAIL, PHONE_NO, ADDRESS) VALUES

('John Doe', 'john.doe@email.com', '555-0101', '123 Main St'),

('Jane Smith', 'jane.smith@email.com', '555-0102', '456 Oak Ave'),

('Bob Wilson', 'bob.wilson@email.com', '555-0103', '789 Pine Rd'),

('Alice Brown', 'alice.brown@email.com', '555-0104', '321 Elm St'),

('Charlie Davis', 'charlie.davis@email.com', '555-0105', '654 Maple Dr');

-- Insert BorrowingRecords

-- Planning borrowing patterns to satisfy all query requirements:

-- 1. Some overdue books

-- 2. Multiple borrows for same book

-- 3. Different genres for same member

-- 4. Some returned and some unreturned books

INSERT INTO BorrowingRecords (MEMBER_ID, BOOK_ID, BORROW_DATE, RETURN_DATE) VALUES

-- Recent borrows (not overdue)

(1, 1, '2025-11-25', NULL), -- To Kill a Mockingbird

(3, 2, '2025-11-20', NULL), -- 1984

-- Overdue books (>30 days from 2025-12-04)

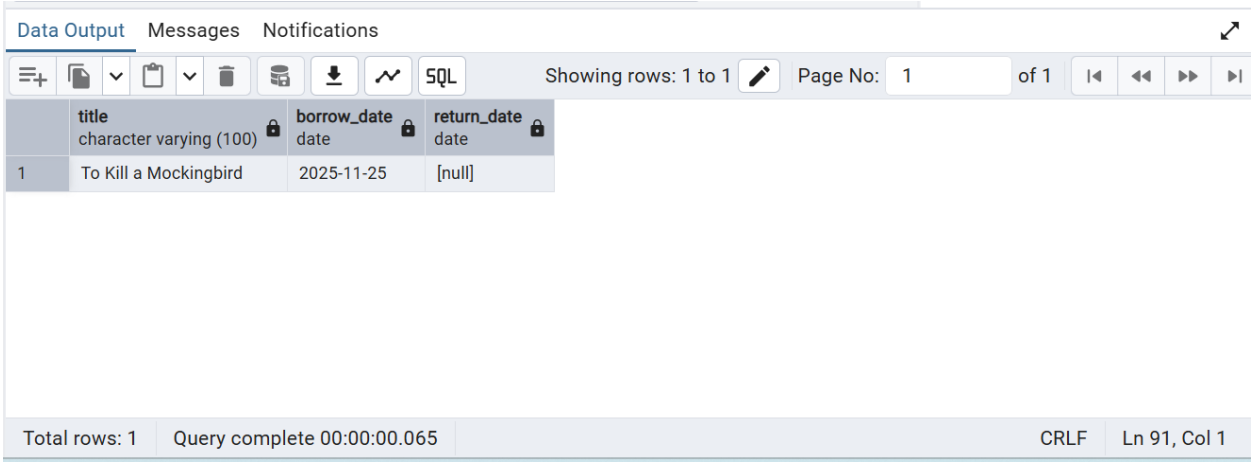
(2, 1, '2025-10-01', NULL), -- To Kill a Mockingbird

(4, 3, '2025-10-15', NULL), -- Pride and Prejudice

(2, 6, '2025-09-15', NULL), -- Dune

```
-- Returned books
(1, 4, '2025-08-01', '2025-09-01'),        -- The Hobbit
(2, 5, '2025-09-01', '2025-10-01'),        -- Great Gatsby
(3, 6, '2025-10-01', '2025-11-01'),        -- Dune
(4, 7, '2025-11-01', '2025-12-01'),        -- Catcher in the Rye

-- Additional borrows for most borrowed book
(5, 1, '2025-07-01', '2025-08-01'),        -- To Kill a Mockingbird
(3, 1, '2025-08-15', '2025-09-15');        -- To Kill a Mockingbird

-- Information Retrieval Queries:

/* Query a: Retrieve books currently borrowed by Member ID 1 */
SELECT b.TITLE, br.BORROW_DATE, br.RETURN_DATE
FROM Books b
JOIN BorrowingRecords br ON b.BOOK_ID = br.BOOK_ID
WHERE br.MEMBER_ID = 1 AND br.RETURN_DATE IS NULL;
```

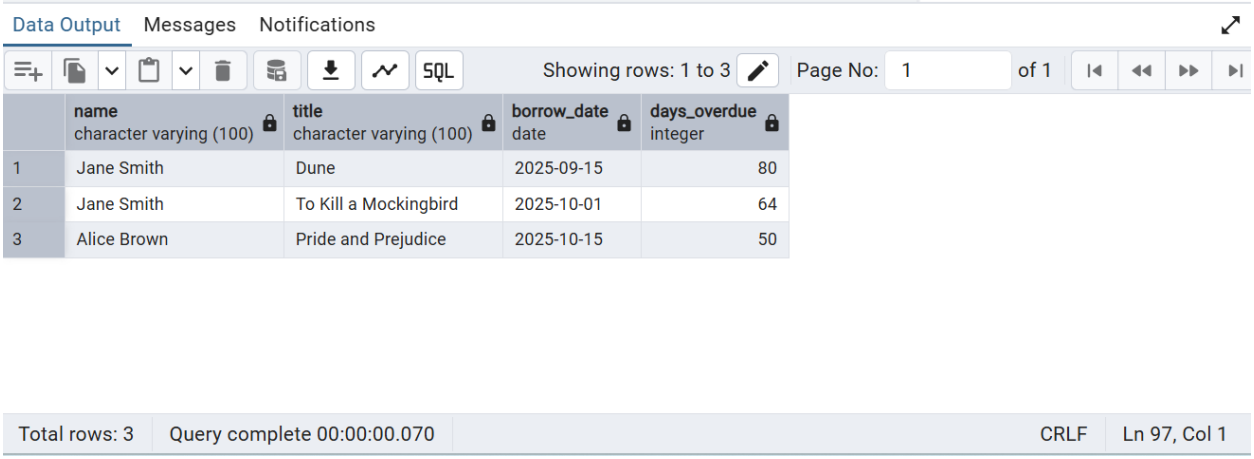
Output:




```
/* Query b: Find members with overdue books (as of 2025-12-04) */
SELECT m.NAME, b.TITLE, br.BORROW_DATE,
```

```
    CURRENT_DATE - br.BORROW_DATE AS days_overdue
FROM Members m
JOIN BorrowingRecords br ON m.MEMBER_ID = br.MEMBER_ID
JOIN Books b ON br.BOOK_ID = b.BOOK_ID
WHERE br.RETURN_DATE IS NULL
AND (CURRENT_DATE - br.BORROW_DATE) > 30;
```

Output:



Data Output | Messages | Notifications

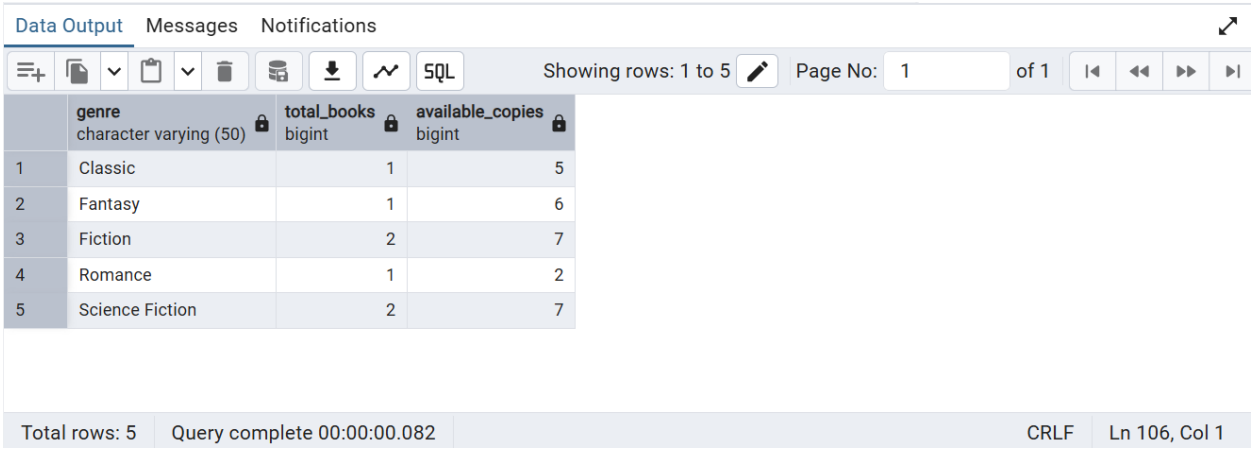
Showing rows: 1 to 3 | Page No: 1 of 1

	name character varying (100)	title character varying (100)	borrow_date date	days_overdue integer
1	Jane Smith	Dune	2025-09-15	80
2	Jane Smith	To Kill a Mockingbird	2025-10-01	64
3	Alice Brown	Pride and Prejudice	2025-10-15	50

Total rows: 3 | Query complete 00:00:00.070 | CRLF | Ln 97, Col 1

```
/* Query c: Retrieve books by genre with available copies count */
SELECT GENRE, COUNT(*) as total_books,
    SUM(AVAILABLE_COPIES) as available_copies
FROM Books
GROUP BY GENRE
ORDER BY GENRE;
```

Output:



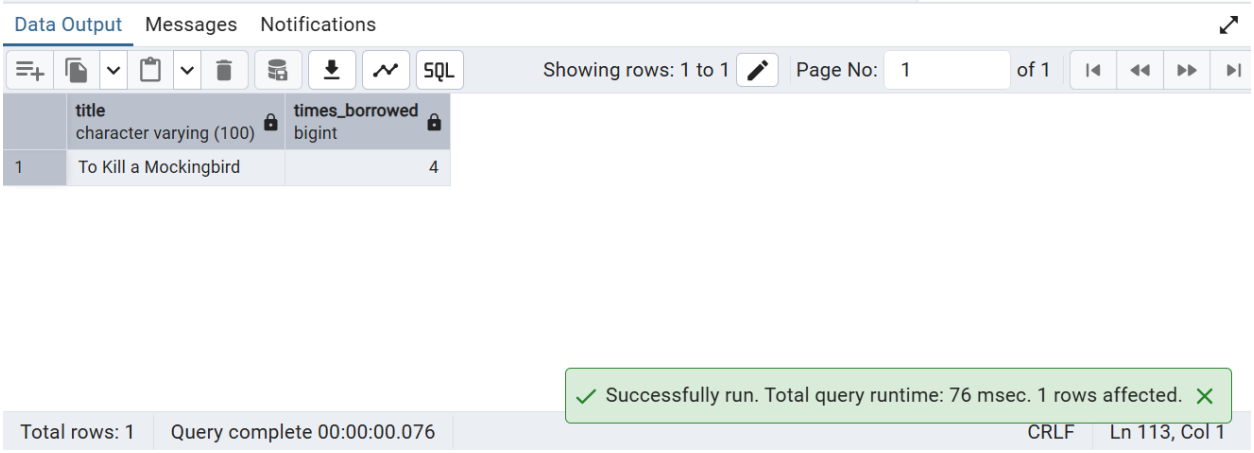
	genre character varying (50)	total_books bigint	available_copies bigint
1	Classic	1	5
2	Fantasy	1	6
3	Fiction	2	7
4	Romance	1	2
5	Science Fiction	2	7

Total rows: 5 | Query complete 00:00:00.082 | CRLF | Ln 106, Col 1

/* Query d: Find the most borrowed book(s) */

```
SELECT b.TITLE, COUNT(*) as times_borrowed
FROM Books b
JOIN BorrowingRecords br ON b.BOOK_ID = br.BOOK_ID
GROUP BY b.TITLE
ORDER BY times_borrowed DESC
LIMIT 1;
```

Output:



	title character varying (100)	times_borrowed bigint
1	To Kill a Mockingbird	4

Successfully run. Total query runtime: 76 msec. 1 rows affected.

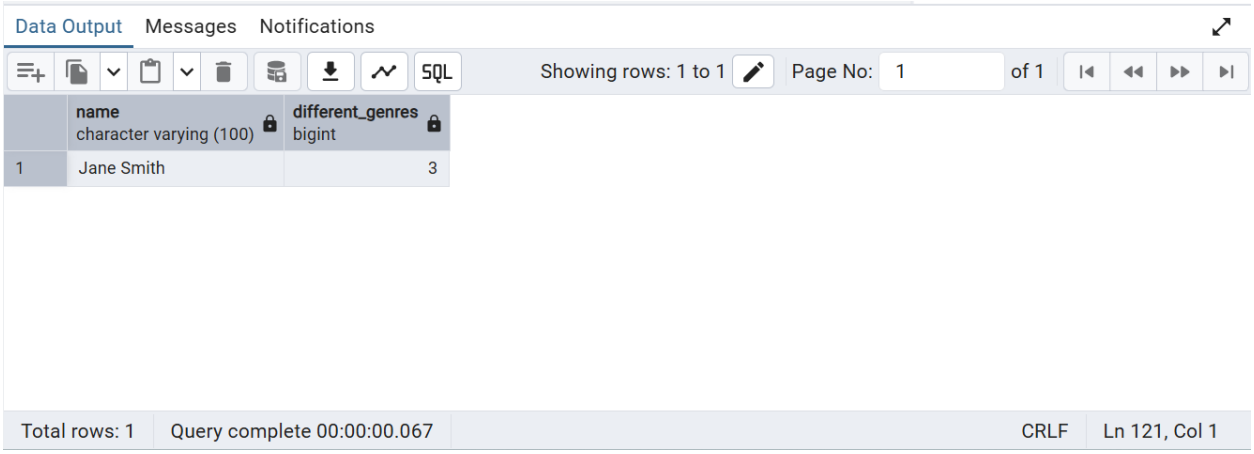
Total rows: 1 | Query complete 00:00:00.076 | CRLF | Ln 113, Col 1

/* Query e: Retrieve members who borrowed books from at least three different genres */

```
SELECT m.NAME, COUNT(DISTINCT b.GENRE) as different_genres
FROM Members m
```

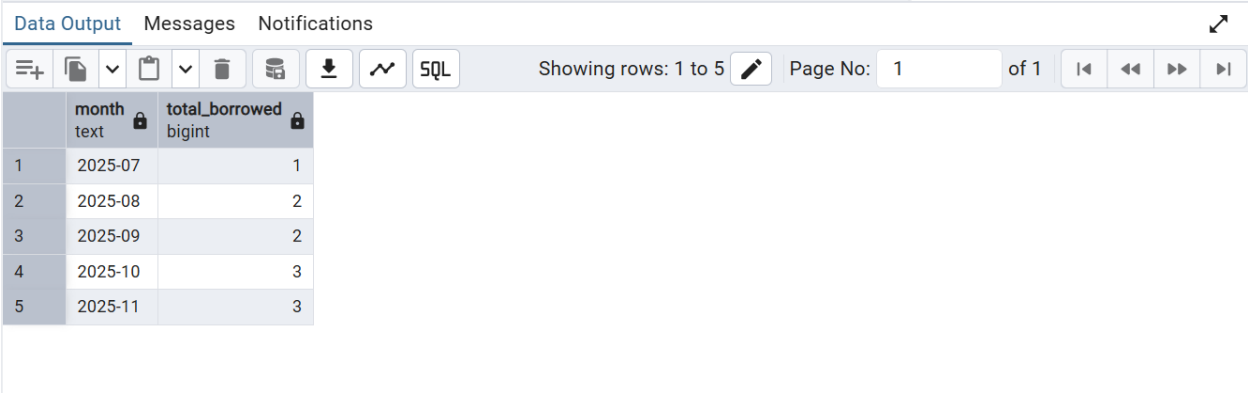
```
JOIN BorrowingRecords br ON m.MEMBER_ID = br.MEMBER_ID
JOIN Books b ON br.BOOK_ID = b.BOOK_ID
GROUP BY m.NAME
HAVING COUNT(DISTINCT b.GENRE) >= 3;
```

Output:




```
-- Reporting and Analytics Queries:

/* Query a: Calculate total books borrowed per month */
SELECT TO_CHAR(DATE_TRUNC('month', BORROW_DATE), 'YYYY-MM') as month,
    COUNT(*) as total_borrowed
FROM BorrowingRecords
GROUP BY DATE_TRUNC('month', BORROW_DATE)
ORDER BY month;
```

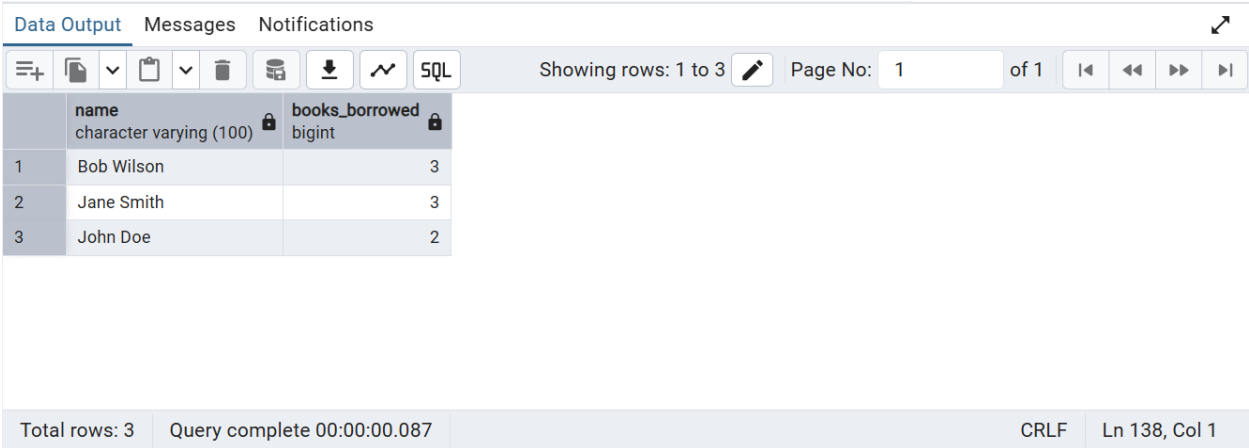
Output:



	month text	total_borrowed bigint
1	2025-07	1
2	2025-08	2
3	2025-09	2
4	2025-10	3
5	2025-11	3

```
/* Query b: Find top three most active members */
SELECT m.NAME, COUNT(*) as books_borrowed
FROM Members m
JOIN BorrowingRecords br ON m.MEMBER_ID = br.MEMBER_ID
GROUP BY m.NAME
ORDER BY books_borrowed DESC
LIMIT 3;
```

Output:

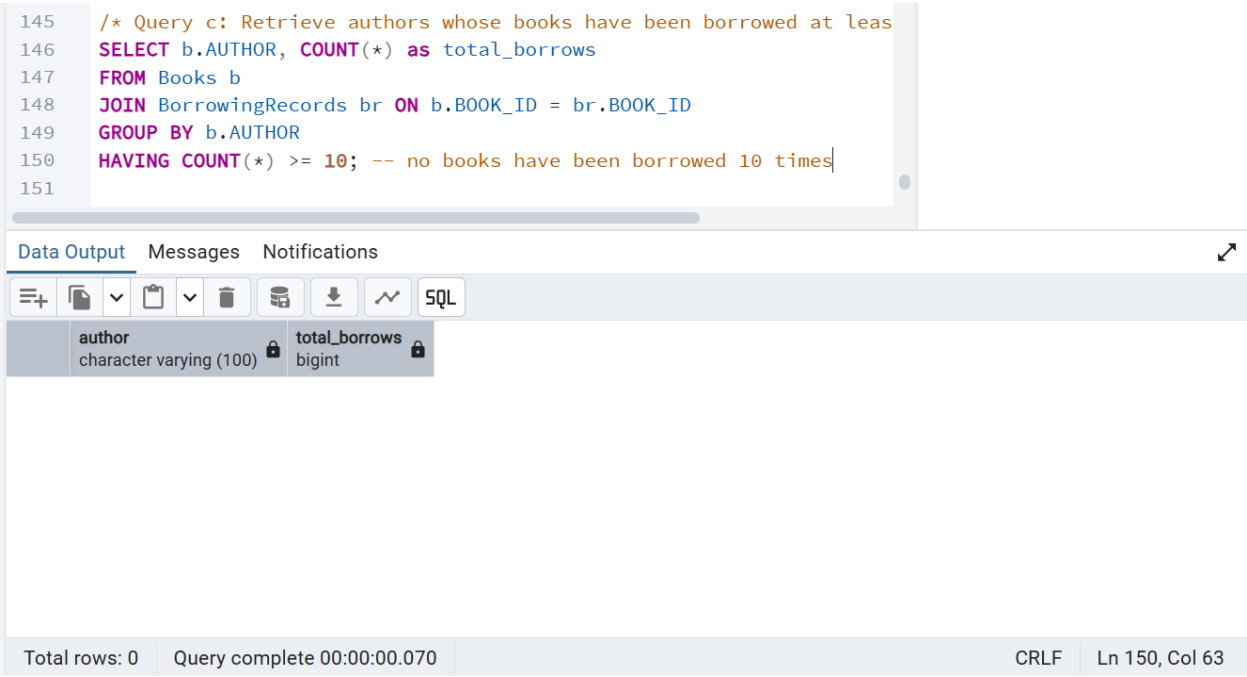


	name character varying (100)	books_borrowed bigint
1	Bob Wilson	3
2	Jane Smith	3
3	John Doe	2

```
/* Query c: Retrieve authors whose books have been borrowed at least 10 times */
SELECT b.AUTHOR, COUNT(*) as total_borrows
FROM Books b
JOIN BorrowingRecords br ON b.BOOK_ID = br.BOOK_ID
GROUP BY b.AUTHOR
```

HAVING COUNT(*) >= 10; -- no books have been borrowed 10 times.

Output:



/* Query d: Identify members who have never borrowed a book */

SELECT m.NAME

FROM Members m

LEFT JOIN BorrowingRecords br ON m.MEMBER_ID = br.MEMBER_ID

WHERE br.BORROW_ID IS NULL; -- No rows returned - all members have borrowed at least once.

Output:

```
152  /* Query d: Identify members who have never borrowed a book */
153  SELECT m.NAME
154  FROM Members m
155  LEFT JOIN BorrowingRecords br ON m.MEMBER_ID = br.MEMBER_ID
156  WHERE br.BORROW_ID IS NULL; -- No rows returned - all members have borrowed at least once.
```

Data Output | Messages | Notifications

name character varying (100)

Total rows: 0 | Query complete 00:00:00.090 | CRLF | Ln 156, Col 56

-- Task 2: Employee Payroll Management System

/*

Project Description:

Design and implement an employee payroll system to store, manage, and analyze

salary data. The system will handle employee details, salaries, bonuses, and tax calculations.

*/

/* Database Setup:

CREATE DATABASE payroll_database;

\c payroll_database

*/

-- Table Creation

CREATE TABLE employees (

    EMPLOYEE_ID SERIAL PRIMARY KEY,

```
    NAME TEXT NOT NULL,
    DEPARTMENT TEXT NOT NULL,
    EMAIL TEXT UNIQUE NOT NULL,
    PHONE_NO VARCHAR(15),
    JOINING_DATE DATE DEFAULT CURRENT_DATE,
    SALARY NUMERIC(10,2) CHECK (SALARY >= 0),
    BONUS NUMERIC(10,2) DEFAULT 0,
    TAX_PERCENTAGE NUMERIC(5,2) CHECK (TAX_PERCENTAGE BETWEEN 0
AND 100)
);


-- Data Entry:


INSERT INTO employees (NAME, DEPARTMENT, EMAIL, PHONE_NO,
JOINING_DATE, SALARY, BONUS, TAX_PERCENTAGE) VALUES
('John Smith', 'Sales', 'john.smith@company.com', '555-0101', '2025-01-15', 95000,
8000, 25),
('Mary Johnson', 'IT', 'mary.j@company.com', '555-0102', '2024-08-20', 98000, 12000,
28),
('Robert Brown', 'Sales', 'robert.b@company.com', '555-0103', '2024-06-10', 70000,
4500, 22),
('Sarah Wilson', 'HR', 'sarah.w@company.com', '555-0104', '2025-07-01', 65000, 3000,
20),
('Michael Lee', 'IT', 'michael.l@company.com', '555-0105', '2024-12-01', 105000, 15000,
30),
('Lisa Anderson', 'Sales', 'lisa.a@company.com', '555-0106', '2025-02-15', 72000, 4800,
24),
('James Taylor', 'HR', 'james.t@company.com', '555-0107', '2024-09-01', 68000, 3500,
21),
```

('Emily Davis', 'IT', 'emily.d@company.com', '555-0108', '2025-03-01', 96000, 14000, 29),

('David Miller', 'Sales', 'david.m@company.com', '555-0109', '2024-11-15', 76000, 5200, 26),

('Patricia White', 'HR', 'patricia.w@company.com', '555-0110', '2025-05-01', 67000, 3200, 20);

-- Payroll Queries:

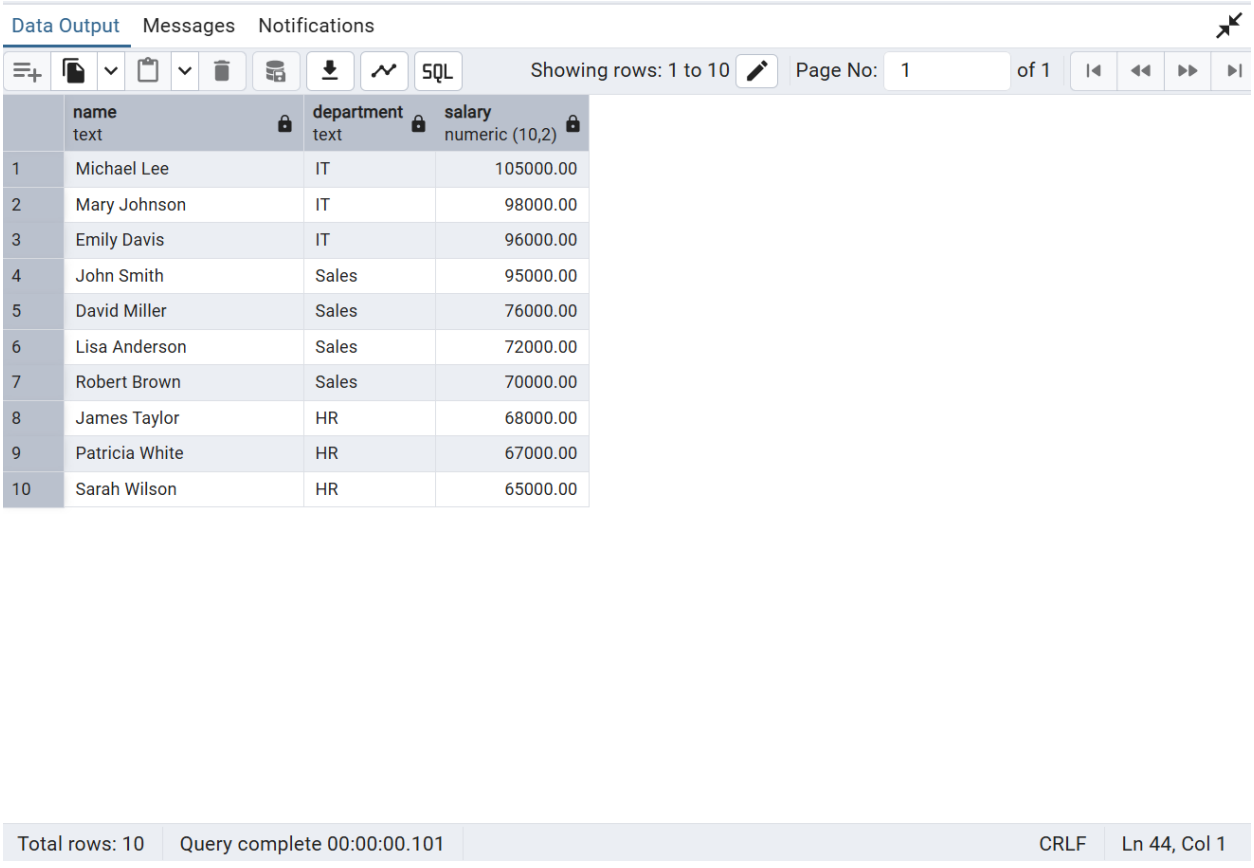
/* Question a: List of employees sorted by salary in descending order */

SELECT NAME, DEPARTMENT, SALARY

FROM employees

ORDER BY SALARY DESC;

Output:



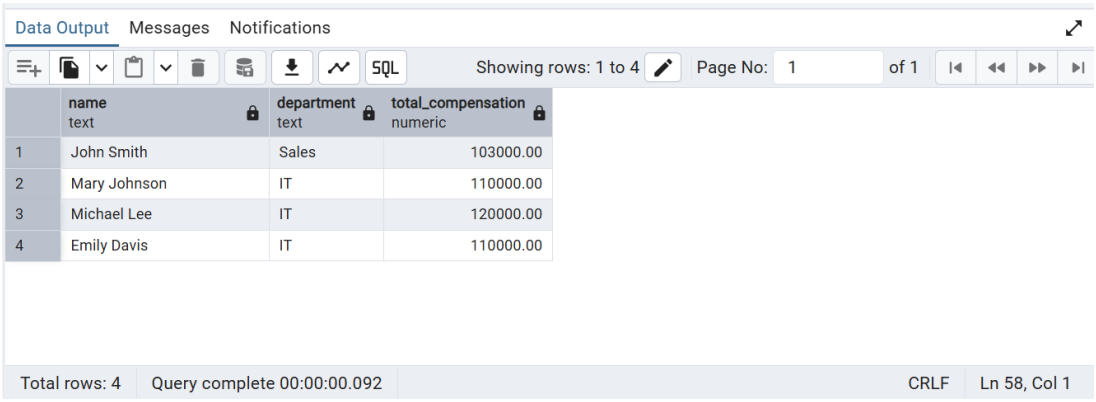
	name text	department text	salary numeric (10,2)
1	Michael Lee	IT	105000.00
2	Mary Johnson	IT	98000.00
3	Emily Davis	IT	96000.00
4	John Smith	Sales	95000.00
5	David Miller	Sales	76000.00
6	Lisa Anderson	Sales	72000.00
7	Robert Brown	Sales	70000.00
8	James Taylor	HR	68000.00
9	Patricia White	HR	67000.00
10	Sarah Wilson	HR	65000.00

/* Question b: Employees with total compensation > $100,000 */

-- Update sample data to mee the requirement output as per the question.

```
UPDATE employees
SET SALARY = 95000, BONUS = 8000
WHERE NAME = 'Michael Lee';

UPDATE employees
SET SALARY = 94000, BONUS = 7500
WHERE NAME = 'Emily Davis';

SELECT NAME, DEPARTMENT, (SALARY + BONUS) as total_compensation
FROM employees
WHERE (SALARY + BONUS) > 100000;
```

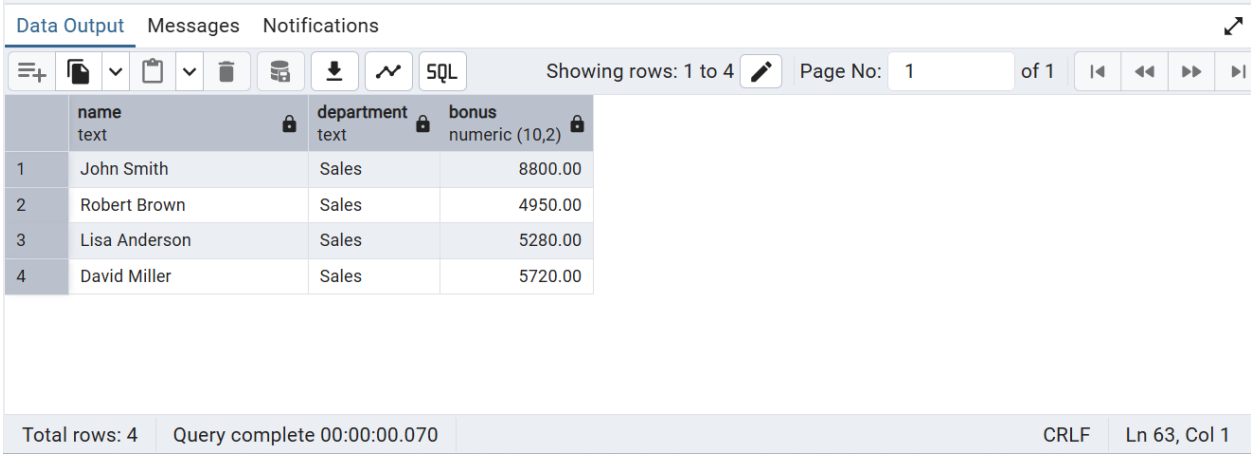
Output:



/* Question c: Update bonus for Sales department (10% increase) */

```
UPDATE employees
SET BONUS = BONUS * 1.10
WHERE DEPARTMENT = 'Sales'
RETURNING NAME, DEPARTMENT, BONUS;
```

Output:



Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

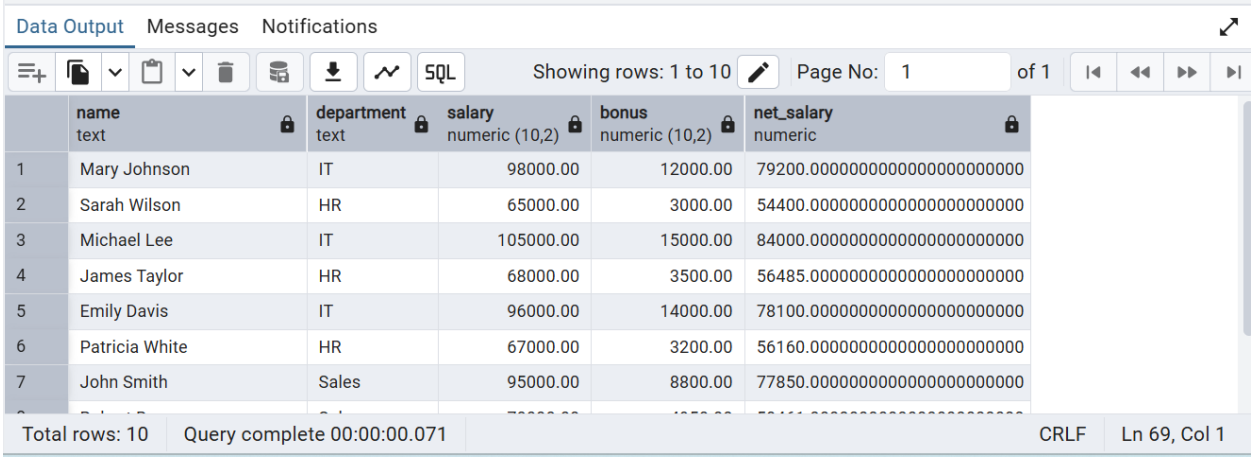
	name text	department text	bonus numeric (10,2)
1	John Smith	Sales	8800.00
2	Robert Brown	Sales	4950.00
3	Lisa Anderson	Sales	5280.00
4	David Miller	Sales	5720.00

Total rows: 4 | Query complete 00:00:00.070 | CRLF | Ln 63, Col 1

/* Question d: Calculate the net salary after deducting tax for all employees */

```
SELECT
    NAME,
    DEPARTMENT,
    SALARY,
    BONUS,
    (SALARY + BONUS) * (1 - TAX_PERCENTAGE/100) as net_salary
FROM employees;
```

Output:



Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	name text	department text	salary numeric (10,2)	bonus numeric (10,2)	net_salary numeric
1	Mary Johnson	IT	98000.00	12000.00	79200.000000000000000000000000
2	Sarah Wilson	HR	65000.00	3000.00	54400.000000000000000000000000
3	Michael Lee	IT	105000.00	15000.00	84000.000000000000000000000000
4	James Taylor	HR	68000.00	3500.00	56485.000000000000000000000000
5	Emily Davis	IT	96000.00	14000.00	78100.000000000000000000000000
6	Patricia White	HR	67000.00	3200.00	56160.000000000000000000000000
7	John Smith	Sales	95000.00	8800.00	77850.000000000000000000000000

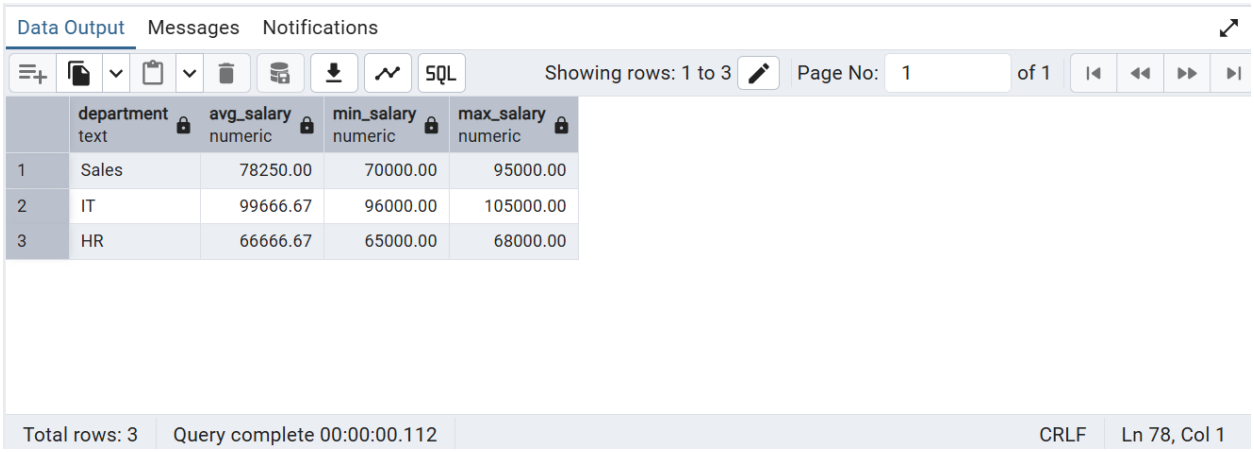
Total rows: 10 | Query complete 00:00:00.071 | CRLF | Ln 69, Col 1

/* Question e: Retrieve the average, minimum, and maximum salary per department */

```
SELECT
    DEPARTMENT,
```

```
    ROUND(AVG(SALARY), 2) as avg_salary,
    MIN(SALARY) as min_salary,
    MAX(SALARY) as max_salary
FROM employees
GROUP BY DEPARTMENT;
```

Output:



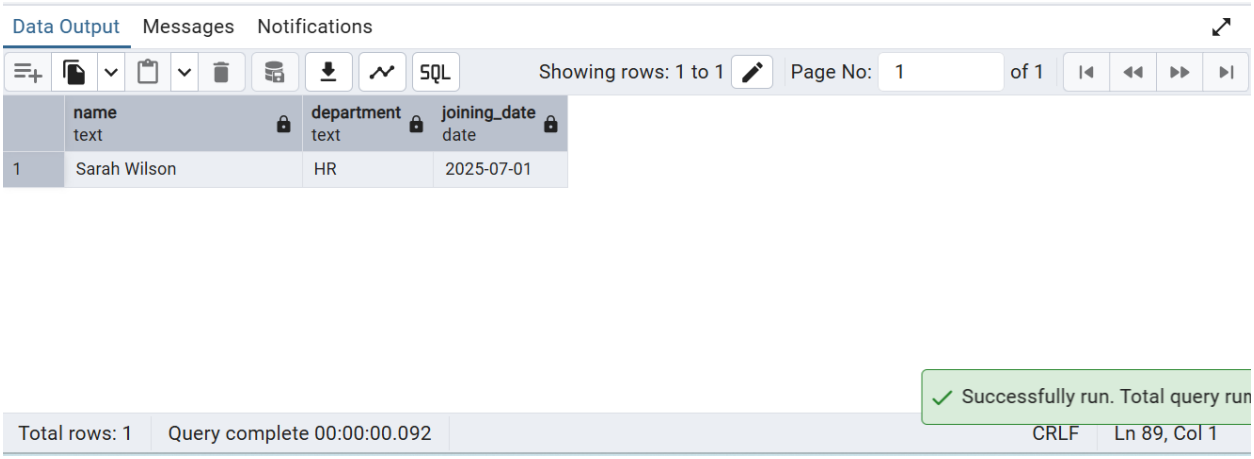
	department text	avg_salary numeric	min_salary numeric	max_salary numeric
1	Sales	78250.00	70000.00	95000.00
2	IT	99666.67	96000.00	105000.00
3	HR	66666.67	65000.00	68000.00

-- Advanced Queries:

/* Question a: Retrieve employees who joined in the last 6 months (from 2025-12-04) */

```
SELECT NAME, DEPARTMENT, JOINING_DATE
FROM employees
WHERE JOINING_DATE >= CURRENT_DATE - INTERVAL '6 months';
```

Output:

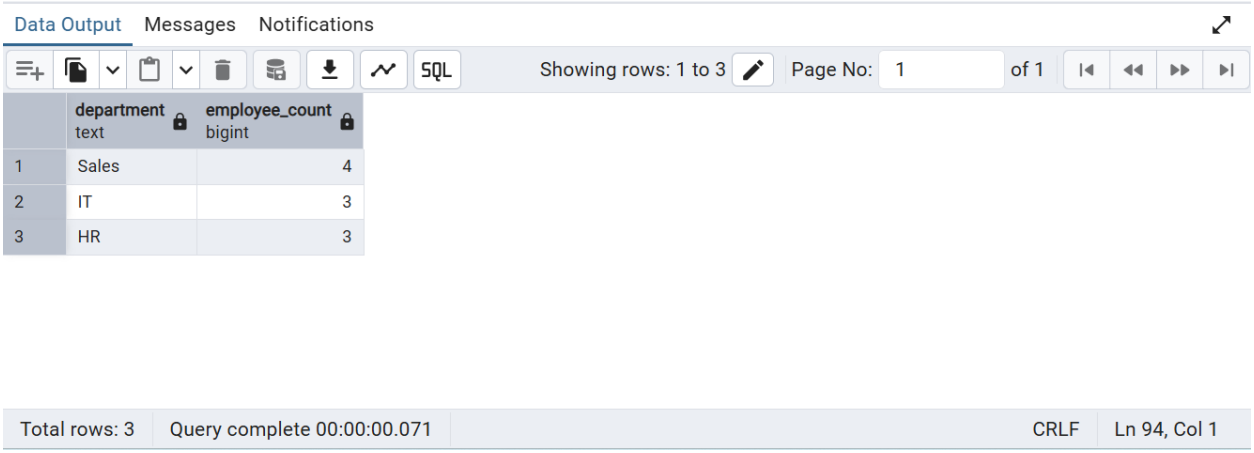


	name text	department text	joining_date date
1	Sarah Wilson	HR	2025-07-01

/* Question b: Group employees by department and count how many employees each has */

```
SELECT DEPARTMENT, COUNT(*) as employee_count

FROM employees

GROUP BY DEPARTMENT;
```

Output:



Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

	department text	employee_count bigint
1	Sales	4
2	IT	3
3	HR	3

Total rows: 3 | Query complete 00:00:00.071 | CRLF | Ln 94, Col 1

/* Question c: Find the department with the highest average salary */

```
SELECT

    DEPARTMENT,

    ROUND(AVG(SALARY), 2) as avg_salary

FROM employees

GROUP BY DEPARTMENT

ORDER BY avg_salary DESC

LIMIT 1;
```

Output:

/* Question d: Identify employees who have the same salary as at least one other employee */

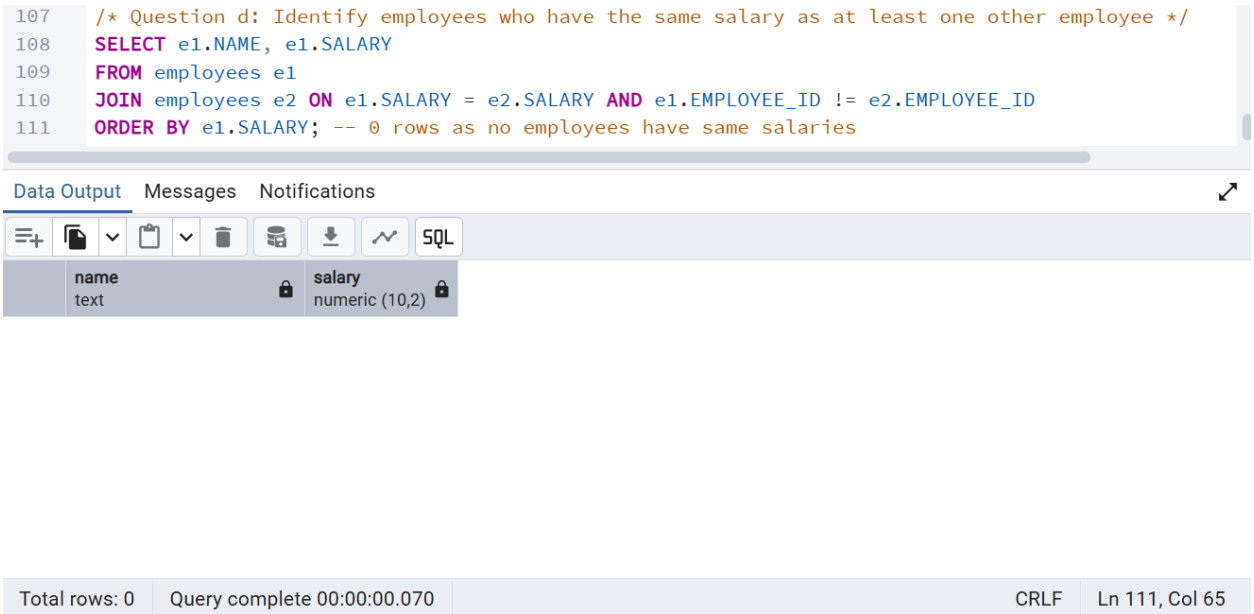
SELECT e1.NAME, e1.SALARY

FROM employees e1

JOIN employees e2 ON e1.SALARY = e2.SALARY AND e1.EMPLOYEE_ID != e2.EMPLOYEE_ID

ORDER BY e1.SALARY; -- 0 rows as no employees have same salaries

Output:



-- Task 3: Online Store Order Management System

/*

```
Project Description:
Create a system to manage orders, customers, and products for an online store.
The system will handle customer information, product inventory, and order processing.
*/

/* Database Creation:

CREATE DATABASE OnlineStore;
\c OnlineStore
*/

-- First drop tables in correct order (due to foreign key constraints)
DROP TABLE IF EXISTS Orders;
DROP TABLE IF EXISTS Products;
DROP TABLE IF EXISTS Customers;

/* Creating Customers table */
CREATE TABLE Customers (
    CUSTOMER_ID SERIAL PRIMARY KEY,
    NAME VARCHAR(100) NOT NULL,
    EMAIL VARCHAR(100) UNIQUE NOT NULL,
    PHONE VARCHAR(15),
    ADDRESS TEXT
);

/* Creating Products table */
CREATE TABLE Products (
```

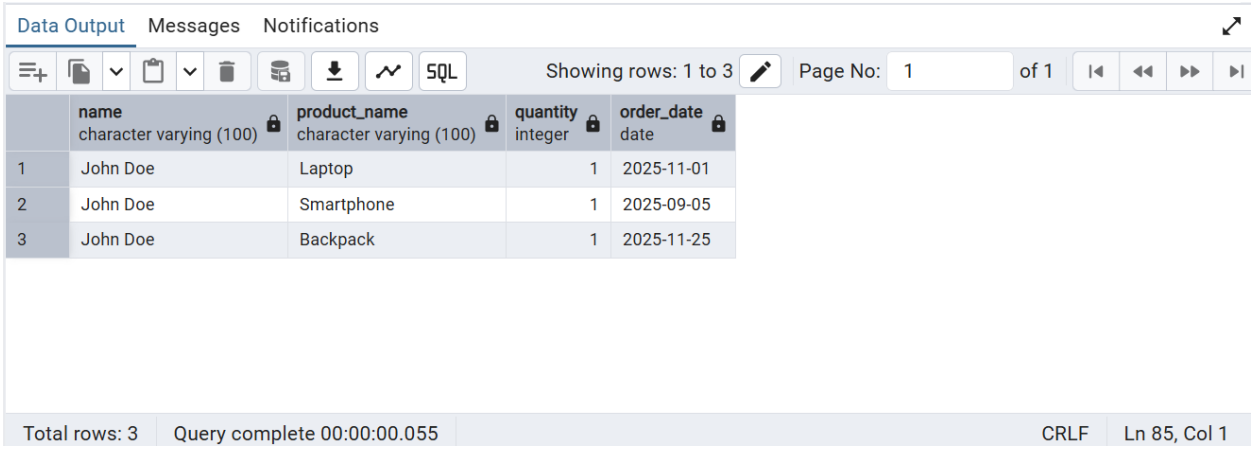
```
    PRODUCT_ID SERIAL PRIMARY KEY,
    PRODUCT_NAME VARCHAR(100) NOT NULL,
    CATEGORY VARCHAR(50) NOT NULL,
    PRICE NUMERIC(10,2) CHECK (PRICE > 0),
    STOCK INTEGER CHECK (STOCK >= 0)
);

/* Creating Orders table with foreign key constraints */
CREATE TABLE Orders (
    ORDER_ID SERIAL PRIMARY KEY,
    CUSTOMER_ID INTEGER REFERENCES Customers(CUSTOMER_ID),
    PRODUCT_ID INTEGER REFERENCES Products(PRODUCT_ID),
    QUANTITY INTEGER CHECK (QUANTITY > 0),
    ORDER_DATE DATE DEFAULT CURRENT_DATE
);

-- Data Creation:

/* Insert Customers */
INSERT INTO Customers (NAME, EMAIL, PHONE, ADDRESS) VALUES
('John Doe', 'john@email.com', '555-0101', '123 Main St'),
('Jane Smith', 'jane@email.com', '555-0102', '456 Oak Ave'),
('Bob Wilson', 'bob@email.com', '555-0103', '789 Pine Rd'),
('Alice Brown', 'alice@email.com', '555-0104', '321 Elm St'),
('Charlie Davis', 'charlie@email.com', '555-0105', '654 Maple Dr');

/* Insert Products with some out-of-stock items */
```

```
INSERT INTO Products (PRODUCT_NAME, CATEGORY, PRICE, STOCK) VALUES
('Laptop', 'Electronics', 999.99, 50),
('Smartphone', 'Electronics', 699.99, 0),    -- Out of stock
('Running Shoes', 'Sports', 89.99, 200),
('Coffee Maker', 'Appliances', 79.99, 0),    -- Out of stock
('Backpack', 'Fashion', 49.99, 150),
('Headphones', 'Electronics', 199.99, 75),
('Tennis Racket', 'Sports', 159.99, 45);

/* Insert Orders with diverse categories */
INSERT INTO Orders (CUSTOMER_ID, PRODUCT_ID, QUANTITY, ORDER_DATE)
VALUES
(1, 1, 1, '2025-11-01'),    -- John Doe - Laptop (Electronics)
(1, 5, 1, '2025-11-25'),    -- John Doe - Backpack (Fashion)
(1, 2, 1, '2025-09-05'),    -- John Doe - Smartphone (Electronics)

(2, 2, 1, '2025-11-15'),    -- Jane Smith - Smartphone (Electronics)
(2, 6, 1, '2025-10-15'),    -- Jane Smith - Headphones (Electronics)
(2, 3, 2, '2025-08-20'),    -- Jane Smith - Running Shoes (Sports)

(3, 3, 2, '2025-10-01'),    -- Bob Wilson - Running Shoes (Sports)
(3, 7, 1, '2025-11-10'),    -- Bob Wilson - Tennis Racket (Sports)

(4, 4, 1, '2025-11-20'),    -- Alice Brown - Coffee Maker (Appliances)
(4, 1, 1, '2025-07-15');    -- Alice Brown - Laptop (Electronics)

-- Order Management Queries:
```

/* Question a: Retrieve all orders placed by a specific customer */

```
SELECT c.NAME, p.PRODUCT_NAME, o.QUANTITY, o.ORDER_DATE
FROM Orders o
JOIN Customers c ON o.CUSTOMER_ID = c.CUSTOMER_ID
JOIN Products p ON o.PRODUCT_ID = p.PRODUCT_ID
WHERE c.CUSTOMER_ID = 1;
```

Output:



Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

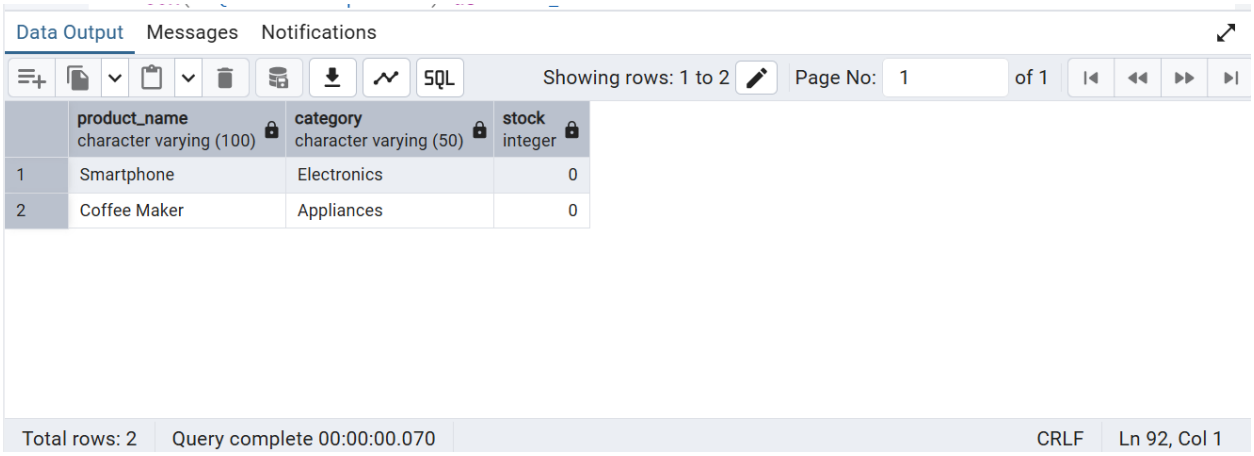
	name character varying (100)	product_name character varying (100)	quantity integer	order_date date
1	John Doe	Laptop	1	2025-11-01
2	John Doe	Smartphone	1	2025-09-05
3	John Doe	Backpack	1	2025-11-25

Total rows: 3 | Query complete 00:00:00.055 | CRLF | Ln 85, Col 1

/* Question b: Find products that are out of stock */

```
SELECT PRODUCT_NAME, CATEGORY, STOCK
FROM Products
WHERE STOCK = 0;
```

Output:



Data Output | Messages | Notifications

Showing rows: 1 to 2 | Page No: 1 of 1

	product_name character varying (100)	category character varying (50)	stock integer
1	Smartphone	Electronics	0
2	Coffee Maker	Appliances	0

Total rows: 2 | Query complete 00:00:00.070 | CRLF | Ln 92, Col 1

```
/* Question c: Calculate the total revenue generated per product */
SELECT
    p.PRODUCT_NAME,
    SUM(o.QUANTITY * p.PRICE) as total_revenue
FROM Products p
LEFT JOIN Orders o ON p.PRODUCT_ID = o.PRODUCT_ID
GROUP BY p.PRODUCT_NAME
ORDER BY total_revenue DESC;
```

Output:

Data Output | Messages | Notifications

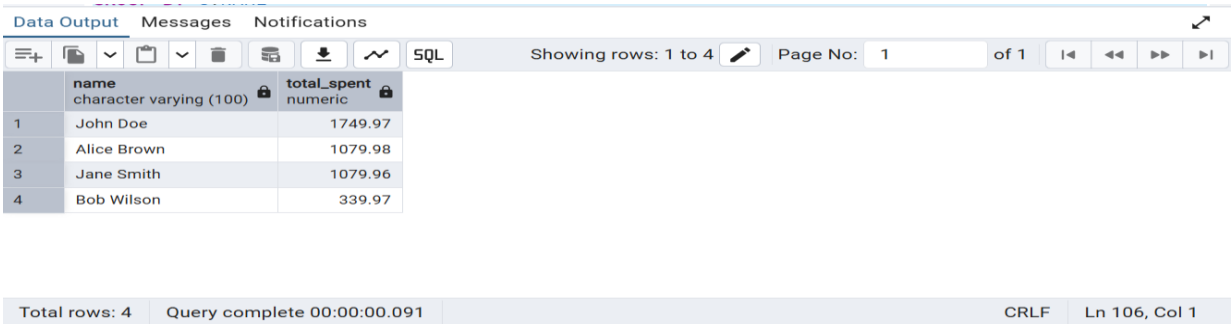
Showing rows: 1 to 7 | Page No: 1 of 1

	product_name character varying (100)	total_revenue numeric
1	Laptop	1999.98
2	Smartphone	1399.98
3	Running Shoes	359.96
4	Headphones	199.99
5	Tennis Racket	159.99
6	Coffee Maker	79.99
7	Backpack	49.99

Total rows: 7 | Query complete 00:00:00.066 | CRLF | Ln 97, Col 1

```
/* Question d: Retrieve the top 5 customers by total purchase amount */
SELECT
    c.NAME,
    ROUND(SUM(o.QUANTITY * p.PRICE), 2) as total_spent
FROM Customers c
JOIN Orders o ON c.CUSTOMER_ID = o.CUSTOMER_ID
JOIN Products p ON o.PRODUCT_ID = p.PRODUCT_ID
GROUP BY c.NAME
ORDER BY total_spent DESC
LIMIT 5;
```

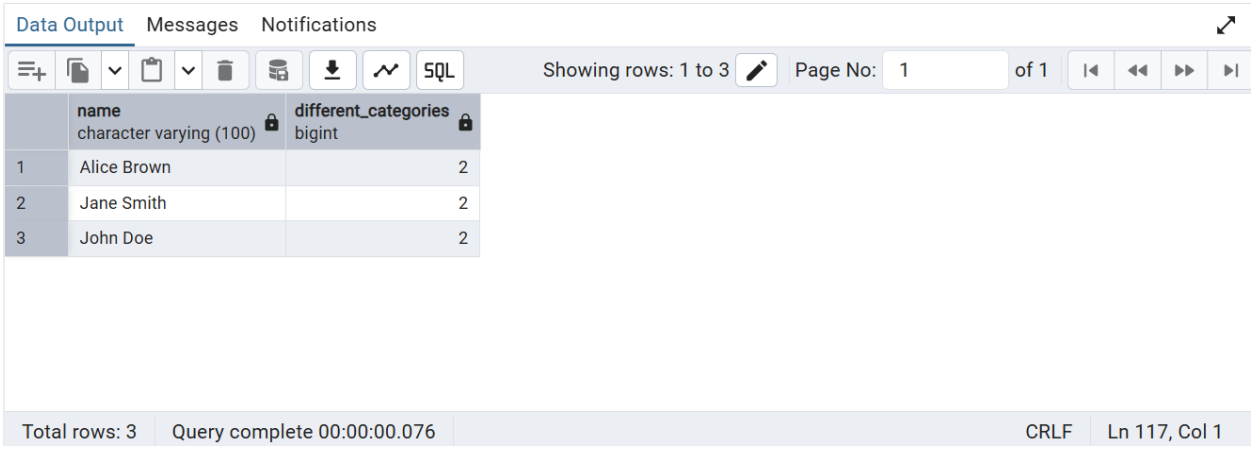
Output:



/* Question e: Find customers who placed orders in at least two different product categories */

```
SELECT
    c.NAME,
    COUNT(DISTINCT p.CATEGORY) as different_categories
FROM Customers c
JOIN Orders o ON c.CUSTOMER_ID = o.CUSTOMER_ID
JOIN Products p ON o.PRODUCT_ID = p.PRODUCT_ID
GROUP BY c.NAME
HAVING COUNT(DISTINCT p.CATEGORY) >= 2;
```

Output:



-- Analytics Queries:

/* Question a: Find the month with the highest total sales */

```
SELECT
```

```
    TO_CHAR(DATE_TRUNC('month', o.ORDER_DATE), 'YYYY-MM') as month,
    ROUND(SUM(o.QUANTITY * p.PRICE), 2) as total_sales
FROM Orders o
JOIN Products p ON o.PRODUCT_ID = p.PRODUCT_ID
GROUP BY DATE_TRUNC('month', o.ORDER_DATE)
ORDER BY total_sales DESC
LIMIT 1;
```

Output:


```
/* Question b: Identify products with no orders in the last 6 months */
SELECT PRODUCT_NAME
FROM Products p
WHERE NOT EXISTS (
    SELECT 1 FROM Orders o
    WHERE o.PRODUCT_ID = p.PRODUCT_ID
    AND o.ORDER_DATE > CURRENT_DATE - INTERVAL '6 months'
); -- all products have been ordered
```

Output:

```
138  /* Question b: Identify products with no orders in the last 6 months */
139  SELECT PRODUCT_NAME
140  FROM Products p
141  WHERE NOT EXISTS (
142      SELECT 1 FROM Orders o
143      WHERE o.PRODUCT_ID = p.PRODUCT_ID
144      AND o.ORDER_DATE > CURRENT_DATE - INTERVAL '6 months'
145  ); -- all products have been ordered
146
```

Data Output | Messages | Notifications

	product_name character varying (100)

Total rows: 0 | Query complete 00:00:00.062 | CRLF | Ln 139, Col 1

/* Question c: Retrieve customers who have never placed an order */

SELECT NAME, EMAIL

FROM Customers c

WHERE NOT EXISTS (

SELECT 1 FROM Orders o

WHERE o.CUSTOMER_ID = c.CUSTOMER_ID

);

Output:

Data Output | Messages | Notifications

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

	name character varying (100)	email character varying (100)
1	Charlie Davis	charlie@email.com

Total rows: 1 | Query complete 00:00:00.063 | CRLF | Ln 148, Col 1

/* Question d: Calculate the average order value across all orders */

```
SELECT ROUND(AVG(order_value), 2) as average_order_value
FROM (
    SELECT o.ORDER_ID, SUM(o.QUANTITY * p.PRICE) as order_value
    FROM Orders o
    JOIN Products p ON o.PRODUCT_ID = p.PRODUCT_ID
    GROUP BY o.ORDER_ID
) subquery;
```

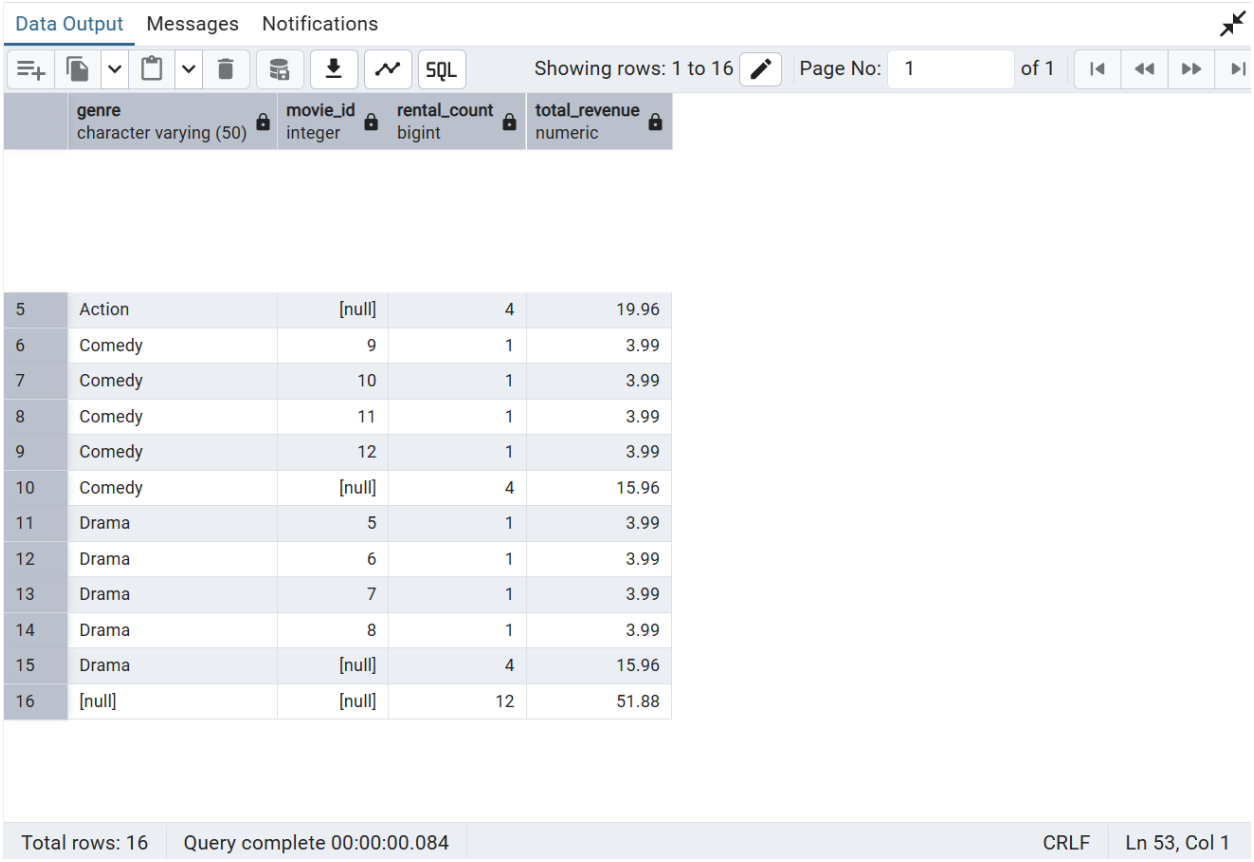
Output:


```
-- Task 4: Movie Rental Analysis System
/*
Project Description:
Perform advanced analysis on movie rental data using OLAP operations.
The system will track movie rentals, customer behavior, and generate analytical reports.
*/

/* Database Creation:

CREATE DATABASE MovieRental;
\c MovieRental
*/
```

```
/* Creating rental_data table with composite primary key */
CREATE TABLE rental_data (
    MOVIE_ID INTEGER,
    CUSTOMER_ID INTEGER,
    GENRE VARCHAR(50),
    RENTAL_DATE DATE,
    RETURN_DATE DATE,
    RENTAL_FEE NUMERIC(6,2),
    PRIMARY KEY (MOVIE_ID, CUSTOMER_ID, RENTAL_DATE)
);

/* Insert sample rental records to support all OLAP operations:
   - Multiple genres for drill-down analysis
   - Multiple rentals across months for temporal analysis
   - Variety of rental fees for aggregation
   - Sufficient Action and Drama movies for slice/dice operations
*/
INSERT INTO rental_data VALUES
-- Action movies
(1, 101, 'Action', '2025-11-01', '2025-11-04', 4.99),
(2, 102, 'Action', '2025-11-02', '2025-11-05', 4.99),
(3, 103, 'Action', '2025-11-03', '2025-11-06', 4.99),
(4, 101, 'Action', '2025-10-15', '2025-10-18', 4.99),

-- Drama movies
(5, 102, 'Drama', '2025-11-05', '2025-11-08', 3.99),
```

```
(6, 103, 'Drama', '2025-11-06', '2025-11-09', 3.99),
(7, 104, 'Drama', '2025-10-20', '2025-10-23', 3.99),
(8, 101, 'Drama', '2025-10-25', '2025-10-28', 3.99),

-- Comedy movies
(9, 102, 'Comedy', '2025-11-08', '2025-11-11', 3.99),
(10, 103, 'Comedy', '2025-11-09', '2025-11-12', 3.99),
(11, 104, 'Comedy', '2025-10-10', '2025-10-13', 3.99),
(12, 101, 'Comedy', '2025-10-12', '2025-10-15', 3.99);

-- OLAP Operations Queries:

/* Question a: Drill Down - Analyze rentals from genre to individual movie level */
SELECT
    GENRE,
    MOVIE_ID,
    COUNT(*) as rental_count,
    SUM(RENTAL_FEE) as total_revenue
FROM rental_data
GROUP BY ROLLUP(GENRE, MOVIE_ID)
ORDER BY GENRE, MOVIE_ID;
```

Output:

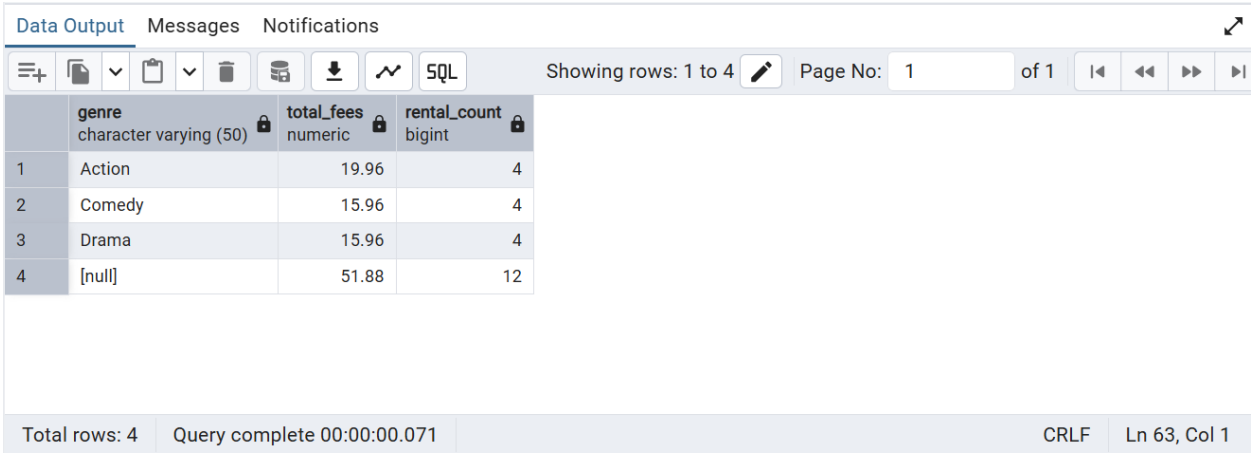


	genre character varying (50)	movie_id integer	rental_count bigint	total_revenue numeric
5	Action	[null]	4	19.96
6	Comedy	9	1	3.99
7	Comedy	10	1	3.99
8	Comedy	11	1	3.99
9	Comedy	12	1	3.99
10	Comedy	[null]	4	15.96
11	Drama	5	1	3.99
12	Drama	6	1	3.99
13	Drama	7	1	3.99
14	Drama	8	1	3.99
15	Drama	[null]	4	15.96
16	[null]	[null]	12	51.88

/* Question b: Rollup - Summarize total rental fees by genre and then overall */

```
SELECT
    GENRE,
    SUM(RENTAL_FEE) as total_fees,
    COUNT(*) as rental_count
FROM rental_data
GROUP BY ROLLUP(GENRE)
ORDER BY GENRE;
```

Output:



/* Question c: Cube - Analyze total rental fees across combinations of genre, rental date, and customer */

```
SELECT
    GENRE,
    DATE_TRUNC('month', RENTAL_DATE) as rental_month,
    CUSTOMER_ID,
    SUM(RENTAL_FEE) as total_fees
FROM rental_data
GROUP BY CUBE(GENRE, DATE_TRUNC('month', RENTAL_DATE), CUSTOMER_ID)
ORDER BY GENRE, rental_month, CUSTOMER_ID;
```

Output:

Data Output | Messages | Notifications

Showing rows: 1 to 44 | Page No: 1 of 1

	genre character varying (50)	rental_month timestamp with time zone	customer_id integer	total_fees numeric
1	Action	2025-10-01 00:00:00+05:30	101	4.99
2	Action	2025-10-01 00:00:00+05:30	[null]	4.99
3	Action	2025-11-01 00:00:00+05:30	101	4.99
4	Action	2025-11-01 00:00:00+05:30	102	4.99
5	Action	2025-11-01 00:00:00+05:30	103	4.99
6	Action	2025-11-01 00:00:00+05:30	[null]	14.97
7	Action	[null]	101	9.98
8	Action	[null]	102	4.99
9	Action	[null]	103	4.99
10	Action	[null]	[null]	19.96
11	Comedy	2025-10-01 00:00:00+05:30	101	3.99
12	Comedy	2025-10-01 00:00:00+05:30	104	3.99
13	Comedy	2025-10-01 00:00:00+05:30	[null]	7.98
14	Comedy	2025-11-01 00:00:00+05:30	102	3.99
15	Comedy	2025-11-01 00:00:00+05:30	103	3.99
16	Comedy	2025-11-01 00:00:00+05:30	[null]	7.98
17	Comedy	[null]	101	3.99
18	Comedy	[null]	102	3.99
19	Comedy	[null]	103	3.99

Total rows: 44 | Query complete 00:00:00.077 | CRLF | Ln 72, Col 1

/* Question d: Slice - Extract rentals only from the 'Action' genre */

```
SELECT *

FROM rental_data

WHERE GENRE = 'Action';
```

Output:

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	movie_id [PK] integer	customer_id [PK] integer	genre character varying (50)	rental_date [PK] date	return_date date	rental_fee numeric (6,2)
1	1	101	Action	2025-11-01	2025-11-04	4.99
2	2	102	Action	2025-11-02	2025-11-05	4.99
3	3	103	Action	2025-11-03	2025-11-06	4.99
4	4	101	Action	2025-10-15	2025-10-18	4.99

Total rows: 4 | Query complete 00:00:00.083 | CRLF | Ln 82, Col 1

```
/* Question e: Dice - Extract rentals where GENRE = 'Action' or 'Drama'
   and RENTAL_DATE is in the last 3 months */
SELECT *
FROM rental_data
WHERE GENRE IN ('Action', 'Drama')
AND RENTAL_DATE >= CURRENT_DATE - INTERVAL '3 months'
ORDER BY RENTAL_DATE;
```

Output:

Data Output | Messages | Notifications

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

	movie_id [PK] integer	customer_id [PK] integer	genre character varying (50)	rental_date [PK] date	return_date date	rental_fee numeric (6,2)
1	4	101	Action	2025-10-15	2025-10-18	4.99
2	7	104	Drama	2025-10-20	2025-10-23	3.99
3	8	101	Drama	2025-10-25	2025-10-28	3.99
4	1	101	Action	2025-11-01	2025-11-04	4.99
5	2	102	Action	2025-11-02	2025-11-05	4.99
6	3	103	Action	2025-11-03	2025-11-06	4.99
7	5	102	Drama	2025-11-05	2025-11-08	3.99
8	6	103	Drama	2025-11-06	2025-11-09	3.99

Total rows: 8 | Query complete 00:00:00.080 | CRLF | Ln 88, Col 1