

SQL WEEK THREE TASK REPORT

The first step is to set the search path to the schema with the tables we want to use.

```
set search_path to luxsql;
```

Next is to view all the tables needed.

```
select * from customers;
```

```
select * from books;
```

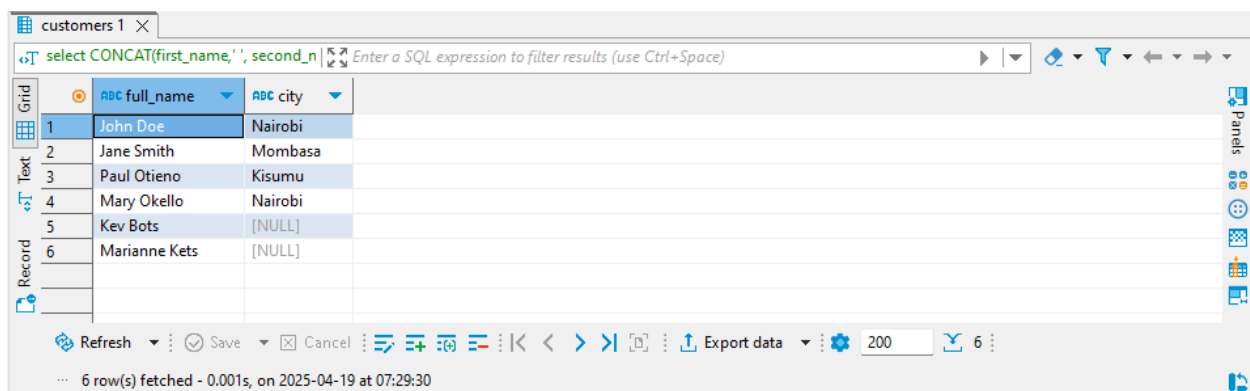
```
select * from orders;
```

Basic Queries

1. List all customers with their full name and city.

The full name is needed therefore we need to combine two or more strings into one (first_name and the second_name) therefore use the CONCAT function.

```
select CONCAT(first_name, ' ', second_name) as Full_Name, city  
from customers;
```



| | ABC full_name | ABC city |
|---|---------------|----------|
| 1 | John Doe | Nairobi |
| 2 | Jane Smith | Mombasa |
| 3 | Paul Otieno | Kisumu |
| 4 | Mary Okello | Nairobi |
| 5 | Kev Bots | [NULL] |
| 6 | Marianne Kets | [NULL] |

2. Show all books priced above 2000.

We have to use the where clause so as to filter for books priced above 2000.

```
select * from books
```

```
where price > 2000;
```

books 1 X

select * from books where price > 2000 Enter a SQL expression to filter results (use Ctrl+Space)

| | book_id | title | author | price | published_date |
|---|---------|-----------------------|---------------|-------|----------------|
| 1 | 102 | Advanced PostgreSQL | Grace Achieng | 2,500 | 2023-02-20 |
| 2 | 103 | Learning Python | James Mwangi | 3,000 | 2022-11-10 |
| 3 | 104 | Data Analytics Basics | Susan Njeri | 2,200 | 2023-03-05 |

Refresh Save Cancel Export data 200 3

3 row(s) fetched - 0.020s (0.001s fetch), on 2025-04-19 at 07:32:55

3. List customers who live in 'Nairobi'.

We have to use the where clause so as to filter for the customers whose city is recorded as nairobi.

select * from customers

where city = 'Nairobi';

customers 1 X

select * from customers where city = 'Nairobi' Enter a SQL expression to filter results (use Ctrl+Space)

| | customer_id | first_name | second_name | email | phone_number | city |
|---|-------------|------------|-------------|----------------------|--------------|---------|
| 1 | 1 | John | Doe | john.doe@gmail.com | 0712345678 | Nairobi |
| 2 | 4 | Mary | Okello | maryokello@gmail.com | 0798564321 | Nairobi |

Refresh Save Cancel Export data 200 2

2 row(s) fetched - 0.001s, on 2025-04-19 at 07:43:04

4. Retrieve all book titles that were published in 2023.

We have to filter for books within the 2023 range hence we have to use the between clause.

select title, published_date

from books

where published_date between '2023-01-01' and '2023-12-31';

| | title | published_date |
|---|-----------------------|----------------|
| 1 | Understanding SQL | 2023-01-15 |
| 2 | Advanced PostgreSQL | 2023-02-20 |
| 3 | Data Analytics Basics | 2023-03-05 |

Filtering and Sorting

5. Show all orders placed after March 1st, 2025.

We have to use the where clause so as to filter orders placed after March 1st, 2025

```
select book_id, order_date
from orders
where order_date > '2025-03-01';
```

| | book_id | order_date |
|--|---------|------------|
|--|---------|------------|

6. List all books ordered, sorted by price (descending).

In this case the price is in a different table other than the orders table.

A join is needed so as to get data from the two tables

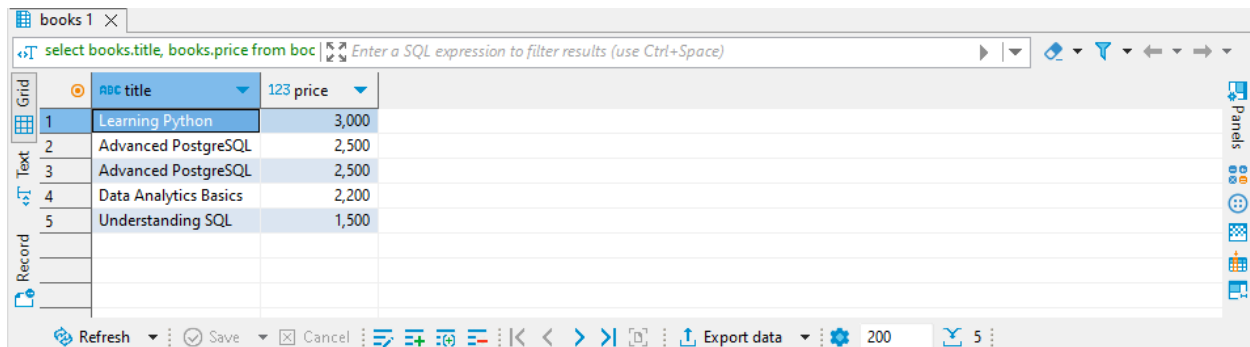
I will use the Inner join to get only books that were ordered by pulling records from the books and orders table.

```
select books.title, books.price from books
inner join orders
on books.book_id = orders.book_id
order by books.price desc;
```

In this case the price is in a different table other than the orders table.

A join is needed so as to get data from the two tables

I will use the Inner join to get only books that were ordered by pulling records from the books and orders table.



The screenshot shows a database application window titled 'books 1'. The SQL query bar contains 'select books.title, books.price from books'. The results are displayed in a grid with two columns: 'title' and 'price'. The data is as follows:

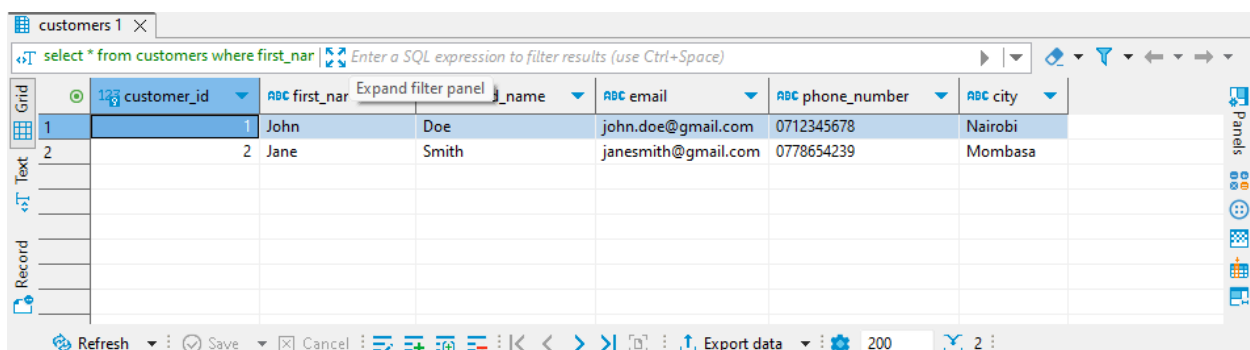
| | title | price |
|---|-----------------------|-------|
| 1 | Learning Python | 3,000 |
| 2 | Advanced PostgreSQL | 2,500 |
| 3 | Advanced PostgreSQL | 2,500 |
| 4 | Data Analytics Basics | 2,200 |
| 5 | Understanding SQL | 1,500 |

7. Show all customers whose names start with 'J'.

The Like clause will be used to pattern match by looking for the first name starting with J

select * from customers

where first name like 'J%';



The screenshot shows a database application window titled 'customers 1'. The SQL query bar contains 'select * from customers where first_name'. The results are displayed in a grid with columns: 'customer_id', 'first_name', 'last_name', 'email', 'phone_number', and 'city'. The data is as follows:

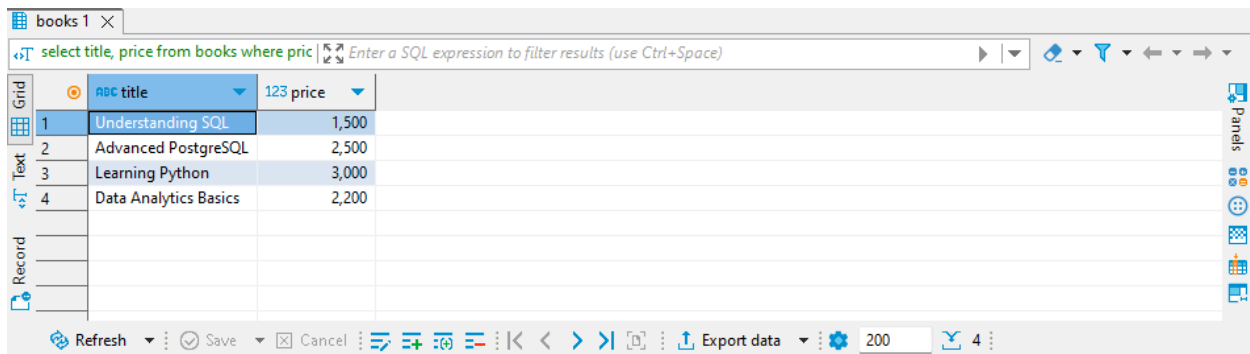
| | customer_id | first_name | last_name | email | phone_number | city |
|---|-------------|------------|-----------|---------------------|--------------|---------|
| 1 | 1 | John | Doe | john.doe@gmail.com | 0712345678 | Nairobi |
| 2 | 2 | Jane | Smith | janesmith@gmail.com | 0778654239 | Mombasa |

8. List books with prices between 1500 and 3000.

-- In this scenario we are filtering a column based on a range hence the need to use the between clause.

select title, price from books

where *price* between '1500' and '3000';



| | ABC title | 123 price |
|---|-----------------------|-----------|
| 1 | Understanding SQL | 1,500 |
| 2 | Advanced PostgreSQL | 2,500 |
| 3 | Learning Python | 3,000 |
| 4 | Data Analytics Basics | 2,200 |

Aggregate Functions and Grouping

9. Count the number of customers in each city.

select city, count(*) as *total_customers*

from customers

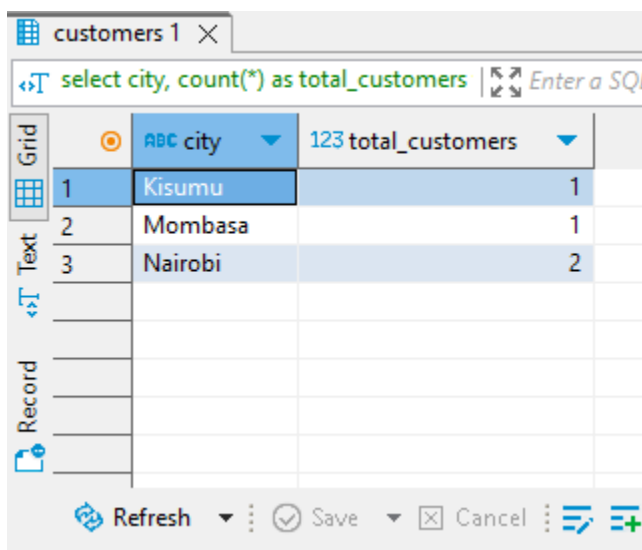
where city is not null

group by city;

count(*) counts how many customers are in each city.

as total_customers: This is the alias given to the count column

The group by in this scenario groups the data by city so the count can be calculated per city.



| | ABC city | 123 total_customers |
|---|----------|---------------------|
| 1 | Kisumu | 1 |
| 2 | Mombasa | 1 |
| 3 | Nairobi | 2 |

10. Show the total number of orders per customer.

```
select customer_id, count(*) as orders_per_customer
```

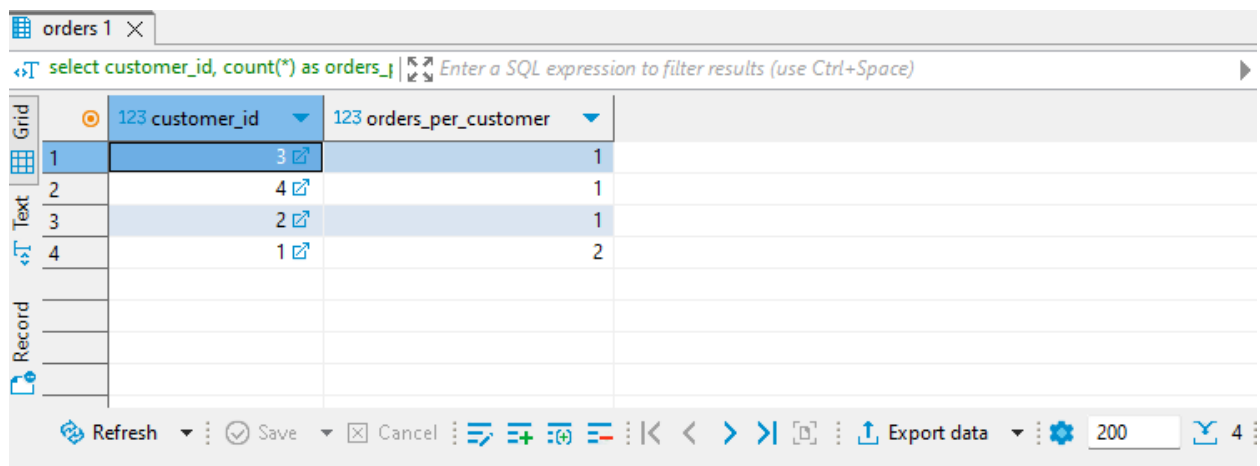
```
from orders
```

```
group by customer_id;
```

-- count(*) counts how many orders were made by every customer_id.

-- as orders_per_customer: This is the alias given to the count column

-- The group by in this scenario groups the data by the customer_id so the count can be calculated per customer.



The screenshot shows a database query results window titled 'orders 1'. The query is 'select customer_id, count(*) as orders_per_customer'. The results are displayed in a table with two columns: 'customer_id' and 'orders_per_customer'. The table has four rows of data.

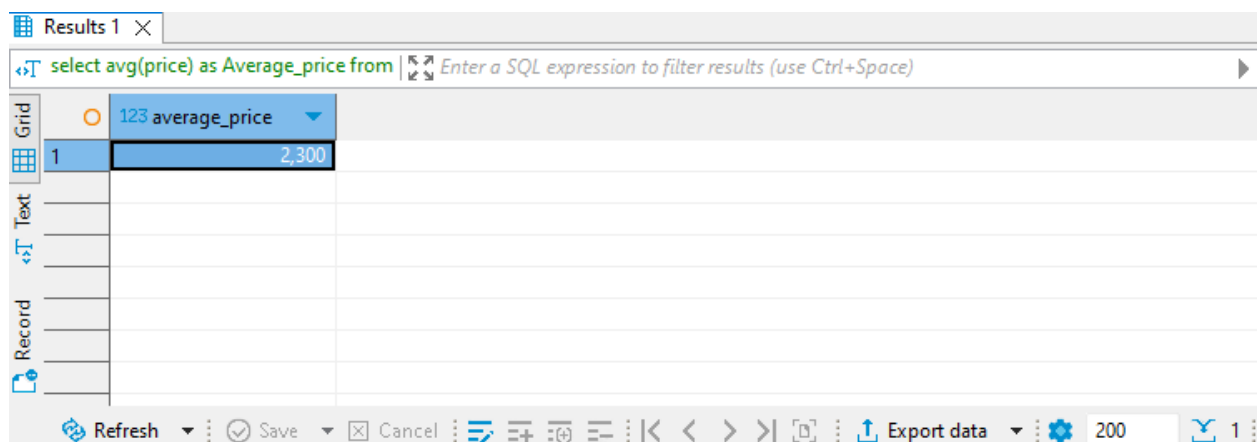
| customer_id | orders_per_customer |
|-------------|---------------------|
| 1 | 3 |
| 2 | 4 |
| 3 | 2 |
| 4 | 1 |

11. Find the average price of books in the store.

```
select avg(price) as Average_price
```

```
from books;
```

The Average aggregate function is used in this scenario to get the mean of the price column.



The screenshot shows a database query results window titled 'Results 1'. The query is 'select avg(price) as Average_price from'. The results are displayed in a table with one column: 'Average_price'. The table has one row of data.

| Average_price |
|---------------|
| 2,300 |

12. List the book title and total quantity ordered for each book

```
select books.title, sum(orders.quantity) as total_quantity
```

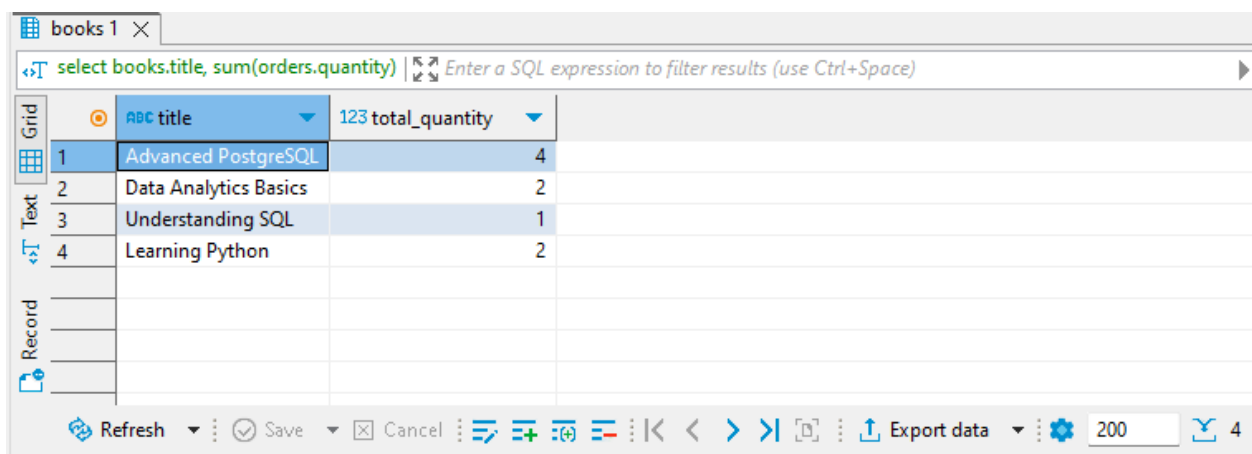
```
from books
```

```
inner join orders on books.book_id = orders.book_id
```

```
group by title;
```

sum is used to add up the total of all quantities per book

The inner join is used to combine rows from both tables needed and only includes rows where there is a matching book_id in both tables



The screenshot shows a database application window titled 'books 1'. The SQL query entered is 'select books.title, sum(orders.quantity)'. The results are displayed in a table with two columns: 'title' and 'total_quantity'. The table contains four rows of data.

| | title | total_quantity |
|---|-----------------------|----------------|
| 1 | Advanced PostgreSQL | 4 |
| 2 | Data Analytics Basics | 2 |
| 3 | Understanding SQL | 1 |
| 4 | Learning Python | 2 |

13. Show customers who have placed more orders than customer with ID = 1.

```
SELECT customer_id, COUNT(*) AS total_orders
```

```
FROM orders
```

```
GROUP BY customer_id
```

```
HAVING COUNT(*) > (
```

```
    SELECT COUNT(*)
```

```
    FROM orders
```

```
    WHERE customer_id = 1
```

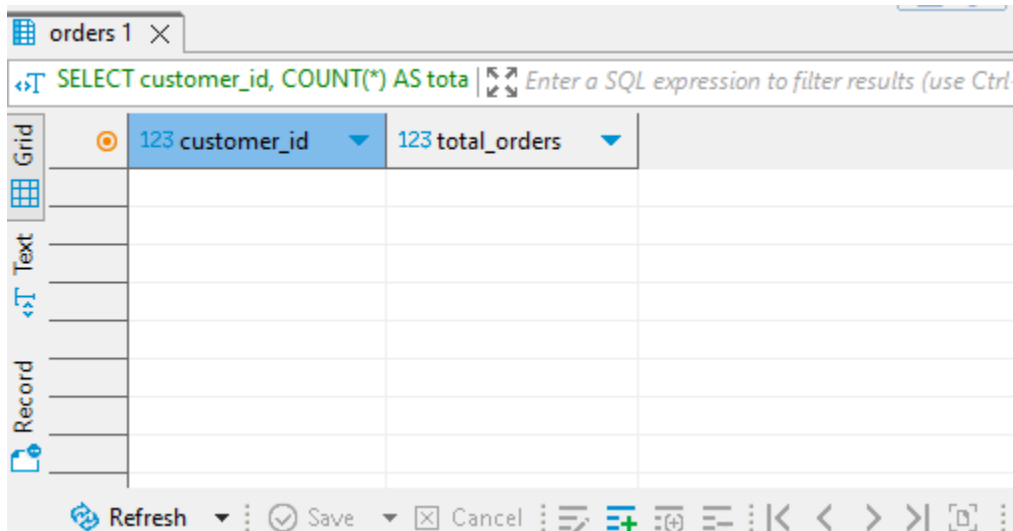
```
);
```

A subquery is a query nested within another query

The inner query fetches the count of orders associated with customer_id 1

The outer query fetches the count of all customers grouping them by their customer ID

The having clause filters based on the count (*) aggregate of the outer query.



14. List books that are more expensive than the average book price.

select title, price

from books

where price > (

select avg(price) as average_price from books

)

order by price **desc**;

The inner query fetches for the average price of all books

The outer query fetches for the books that are more expensive than the average of the inner query then orders by desc

books 1 X

select title, price from books where price > 123 Enter a SQL expression to filter results (use Ctrl+Space)

| | ABC title | 123 price |
|---|---------------------|-----------|
| 1 | Learning Python | 3,000 |
| 2 | Advanced PostgreSQL | 2,500 |
| | | |
| | | |
| | | |
| | | |
| | | |

Refresh Save Cancel

15. Show each customer and the number of orders they placed using a subquery in SELECT.

select customer_id, **concat**(first_name,'-', second_name) **as** full_name,

(

select count(*)

from orders

where orders.customer_id = customers.customer_id) **as** count_of_total_orders

from customers;

concat(first_name,'-', second_name) as full_name shows full name as first_name-second_name

The subquery in select counts how many orders each customer has placed

customers 1 X

select customer_id, concat(first_name,'-', second_name) as full_name, (select count(*) from orders where orders.customer_id = customers.customer_id) as count_of_total_orders Enter a SQL expression to filter results (use Ctrl+Space)

| | 123 customer_id | ABC full_name | 123 count_of_total_orders |
|---|-----------------|---------------|---------------------------|
| 1 | 1 | John-Doe | 2 |
| 2 | 2 | Jane-Smith | 1 |
| 3 | 3 | Paul-Otieno | 1 |
| 4 | 4 | Mary-Okello | 1 |
| 5 | 5 | Kev-Bots | 0 |
| 6 | 6 | Marianne-Kets | 0 |
| | | | |
| | | | |

Refresh Save Cancel Export data

16. Show full name of each customer and the titles of books they ordered.

```
select concat(customers.first_name, ' ', second_name) as full_name, books.title
from customers
inner join orders on orders.customer_id = customers.customer_id
inner join books on books.book_id = orders.book_id
order by full_name asc;
```

| | full_name | title |
|---|-------------|-----------------------|
| 1 | Jane Smith | Understanding SQL |
| 2 | John Doe | Learning Python |
| 3 | John Doe | Advanced PostgreSQL |
| 4 | Mary Okello | Data Analytics Basics |
| 5 | Paul Otieno | Advanced PostgreSQL |

17. List all orders including book title, quantity, and total cost (price × quantity).

```
select * from customers;

select * from orders;

select orders.order_id, books.title, books.price, orders.quantity, books.price*orders.quantity as
total_cost
from books
inner join orders on books.book_id = orders.book_id;
```

| orders(+) 1 X | | | | | |
|--|--------------|-----------------------|-----------|--------------|----------------|
| select orders.order_id, books.title, books Enter a SQL expression to filter results (use Ctrl+Space) | | | | | |
| Grid | 123 order_id | ABC title | 123 price | 123 quantity | 123 total_cost |
| 1 | 6 | Learning Python | 3,000 | 2 | 6,000 |
| 2 | 7 | Understanding SQL | 1,500 | 1 | 1,500 |
| 3 | 8 | Advanced PostgreSQL | 2,500 | 3 | 7,500 |
| 4 | 9 | Data Analytics Basics | 2,200 | 2 | 4,400 |
| 5 | 10 | Advanced PostgreSQL | 2,500 | 1 | 2,500 |
| Record | | | | | |
| Refresh Save Cancel Export data | | | | | |

18. Show customers who haven't placed any orders (LEFT JOIN).

```
select concat(first_name,',',second_name) as customers_without_orders
from customers
left join orders on orders.customer_id = customers.customer_id
where order_id is null;
```

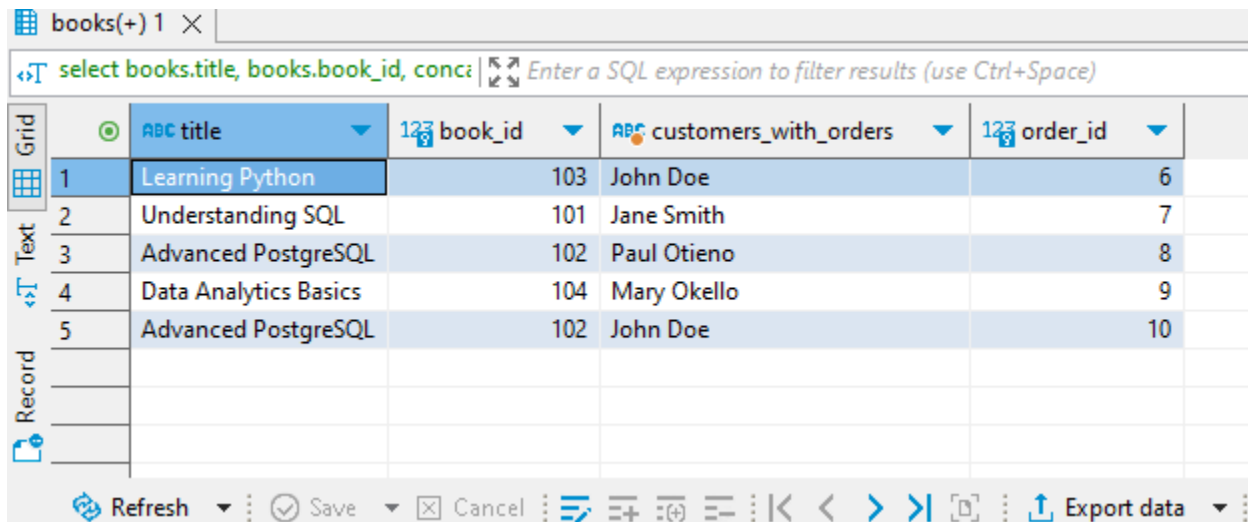
| Results 1 X | |
|--|------------------------------|
| select concat(first_name,',',second_name Enter a SQL expression to filter results (use Ctrl+Space) | |
| Grid | ABC customers_without_orders |
| 1 | Kev Bots |
| 2 | Marianne Kets |
| Text | |
| Record | |
| Refresh Save Cancel | |

19. List all books and the names of customers who ordered them, if any (LEFT JOIN).

```
select books.title, books.book_id, concat(first_name,',',second_name) as
customers_with_orders, orders.order_id
from orders
```

left join customers on orders.customer_id = customers.customer_id

left join books on books.book_id = orders.book_id;



The screenshot shows a database application interface. At the top, there's a tab labeled 'books(+) 1'. Below it, a SQL query is entered: 'select books.title, books.book_id, conc:'. A tooltip提示 says 'Enter a SQL expression to filter results (use Ctrl+Space)'. The main area displays a table with 5 rows and 4 columns. The columns are labeled 'ABC title', '123 book_id', 'ABC customers_with_orders', and '123 order_id'. The rows contain data about books and their orders. On the left, there are icons for 'Grid', 'Text', and 'Record' views. At the bottom, there's a toolbar with buttons for 'Refresh', 'Save', 'Cancel', and 'Export data'.

| | ABC title | 123 book_id | ABC customers_with_orders | 123 order_id |
|---|-----------------------|-------------|---------------------------|--------------|
| 1 | Learning Python | 103 | John Doe | 6 |
| 2 | Understanding SQL | 101 | Jane Smith | 7 |
| 3 | Advanced PostgreSQL | 102 | Paul Otieno | 8 |
| 4 | Data Analytics Basics | 104 | Mary Okello | 9 |
| 5 | Advanced PostgreSQL | 102 | John Doe | 10 |

20. Show customers who live in the same city (SELF JOIN).

select A.customer_id as customer_1,

A.first_name as name_1,

B.customer_id as customer_2,

B.first_name as customer_2,

A.city

from customers A

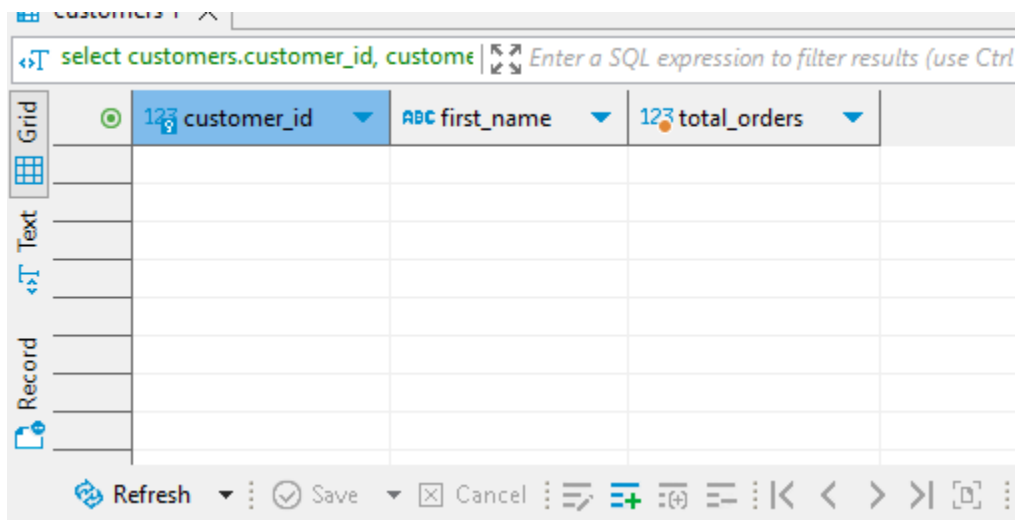
join customers B on A.city = B.city;

22. List customers who ordered the same book more than once.

```
select customer_id, book_id, quantity
```

```
from orders
```

```
where quantity > 1;
```



The screenshot shows a database query interface. At the top, a SQL query is entered: `select customers.customer_id, customer`. Below the query, there is a table grid with the following columns: `customer_id`, `first_name`, and `total_orders`. The grid is currently empty. The interface includes a toolbar with buttons for `Refresh`, `Save`, `Cancel`, and navigation controls.

| | customer_id | first_name | total_orders |
|--|-------------|------------|--------------|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

23. Show each customer's full name, total quantity of books ordered, and total amount spent.

```
select concat(customers.first_name,'-', second_name) as full_name, sum(orders.quantity) as  
total_quantity, sum(books.price) as total_amount_spent
```

```
from customers
```

```
inner join orders on orders.customer_id = customers.customer_id
```

```
inner join books on books.book_id = orders.book_id
```

```
group by full_name;
```

Results 1 X

SQL: `select concat(customers.first_name,'-',` *Enter a SQL expression to filter results (use Ctrl+.*

| | ABC full_name | 123 total_quantity | 123 total_amount_spent |
|---|---------------|--------------------|------------------------|
| 1 | Paul-Otieno | 3 | 2,500 |
| 2 | John-Doe | 3 | 5,500 |
| 3 | Mary-Okello | 2 | 2,200 |
| 4 | Jane-Smith | 1 | 1,500 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Refresh Save Cancel

24. List books that have never been ordered.

select books.title, books.book_id, books.published_date

from books

inner join orders **on** orders.book_id = books.book_id

where orders.order_date is null;

books 1 X

SQL: `select books.title, books.book_id, book:` *Enter a SQL expression to filter results (use Ctrl+.*

| | ABC title | 123 book_id | published_date |
|--|-----------|-------------|----------------|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Refresh Save Cancel

No data- 0.000s, on 2025-04-20 at 12:54:09

25. Find the customer who has spent the most in total (JOIN + GROUP BY + ORDER BY + LIMIT).

```
select concat(customers.first_name,' ',second_name) as full_name, sum(books.price) as  
total_amount_spent
```

```
from customers
```

```
inner join orders on orders.customer_id = customers.customer_id
```

```
inner join books on books.book_id = orders.book_id
```

```
group by full_name
```

```
order by total_amount_spent desc
```

```
limit 1;
```

The screenshot shows a database query results window titled "Results 1". The SQL query entered is: `select concat(customers.first_name,' ',second_name) as full_name, sum(books.price) as total_amount_spent`. The results are displayed in a table with two columns: "full_name" and "total_amount_spent". The first row shows "John Doe" with a total amount spent of 5,500. The window also includes a sidebar with "Grid", "Text", and "Record" views, and a bottom toolbar with "Refresh", "Save", "Cancel", "Export data", and a pagination control showing "1 row(s) fetched - 0.001s, on 2025-04-20 at 12:57:15".

| | full_name | total_amount_spent |
|---|-----------|--------------------|
| 1 | John Doe | 5,500 |

26. Write a query that shows, for each book, the number of different customers who have ordered it.

```
select books.title, COUNT(DISTINCT orders.customer_id) as customers_who_ordered
```

```
from orders
```

```
inner join books on books.book_id = orders.book_id
```

```
group by books.title ;
```


books 1 X

SQL: `select books.title, COUNT(DISTINCT orc` | *Enter a SQL expression to filter results (use Ctrl+Space)*

| | ABC title | 123 customers_who_ordered |
|---|-----------------------|---------------------------|
| 1 | Advanced PostgreSQL | 2 |
| 2 | Data Analytics Basics | 1 |
| 3 | Learning Python | 1 |
| 4 | Understanding SQL | 1 |

Refresh Save Cancel

27. Using a subquery, list books whose total order quantity is above the average order quantity.

select books.title, **sum**(**distinct** orders.quantity) **as** total_order_quantity

from orders

inner join books **on** books.book_id = orders.book_id

group by books.title

Having **sum**(**distinct** orders.quantity) >(

select **avg**(quantity) **as** average_quantity **from** orders);

books 1 X

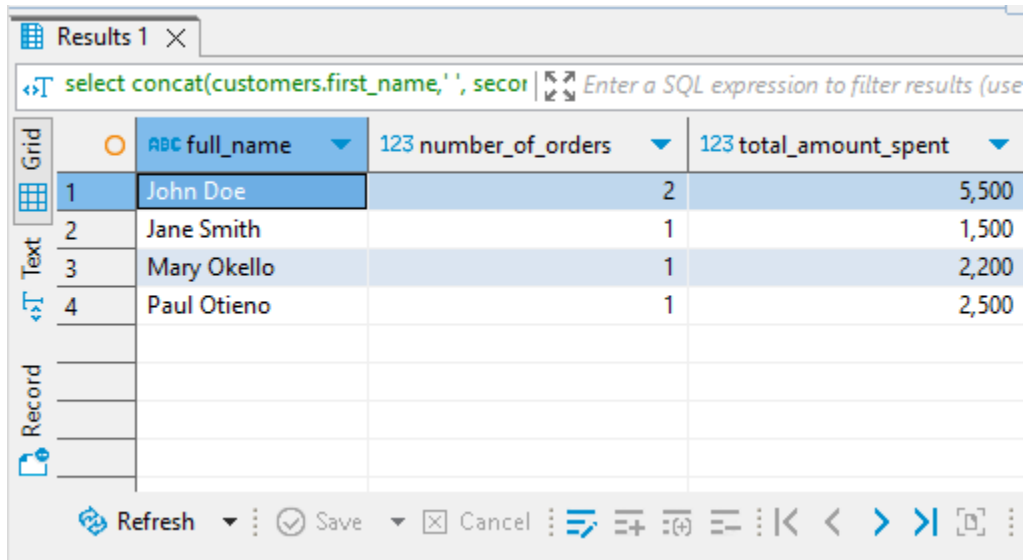
SQL: `select books.title, sum(distinct orders.quant` | *Enter a SQL expression to filter results (use Ctrl+Space)*

| | ABC title | 123 total_order_quantity |
|---|-----------------------|--------------------------|
| 1 | Advanced PostgreSQL | 4 |
| 2 | Data Analytics Basics | 2 |
| 3 | Learning Python | 2 |

Refresh Save Cancel Export data

28. Show the top 3 customers with the highest number of orders and the total amount they spent.

```
select concat(customers.first_name,' ', second_name) as full_name, count(distinct
orders.order_id) as number_of_orders, sum(distinct books.price) as total_amount_spent
from customers
inner join orders on orders.customer_id = customers.customer_id
inner join books on books.book_id = orders.book_id
group by full_name
order by number_of_orders desc;
```



The screenshot shows a database query results window titled "Results 1". The SQL query entered is: `select concat(customers.first_name,' ', second_name) as full_name, count(distinct orders.order_id) as number_of_orders, sum(distinct books.price) as total_amount_spent`. The results are displayed in a table with 4 columns: an index, `full_name`, `number_of_orders`, and `total_amount_spent`. The table shows the top 4 customers by number of orders, ordered in descending order. The first row is John Doe with 2 orders and a total amount spent of 5,500. The second row is Jane Smith with 1 order and a total amount spent of 1,500. The third row is Mary Okello with 1 order and a total amount spent of 2,200. The fourth row is Paul Otieno with 1 order and a total amount spent of 2,500. The table has a toolbar at the bottom with buttons for Refresh, Save, Cancel, and navigation icons.

| | full_name | number_of_orders | total_amount_spent |
|---|-------------|------------------|--------------------|
| 1 | John Doe | 2 | 5,500 |
| 2 | Jane Smith | 1 | 1,500 |
| 3 | Mary Okello | 1 | 2,200 |
| 4 | Paul Otieno | 1 | 2,500 |