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*;

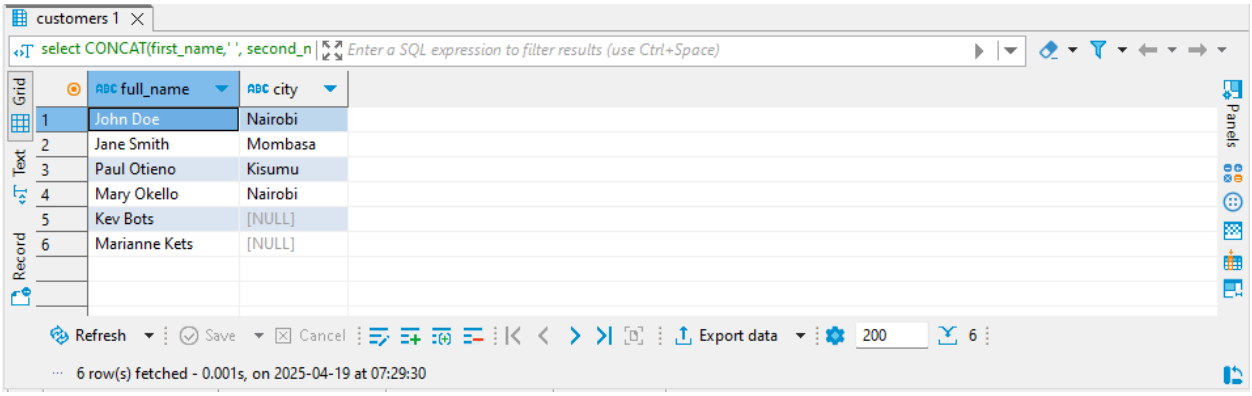
Basic Queries

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

The full name is needed therefore we need to combines 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*;

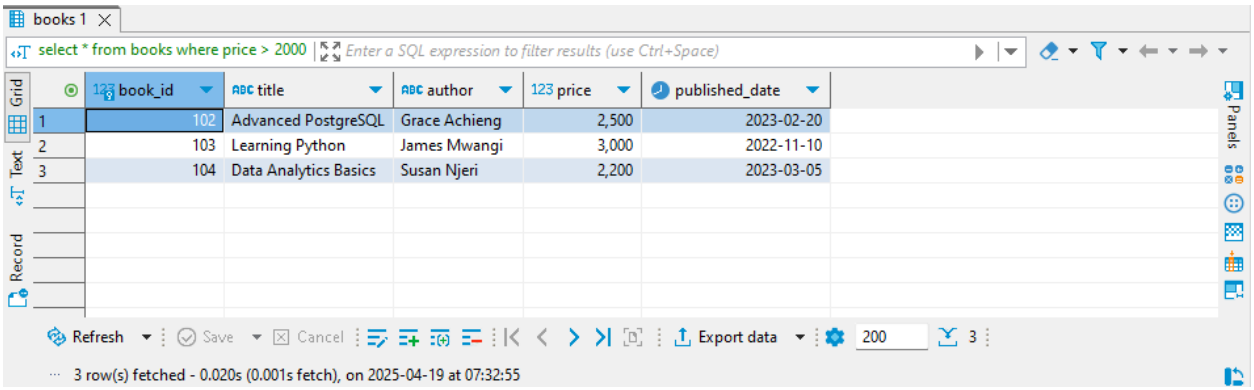


2. Show all books priced above 2000.

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

select * from *books*

where *price* > 2000;



books 1

select * from books where price > 2000

	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

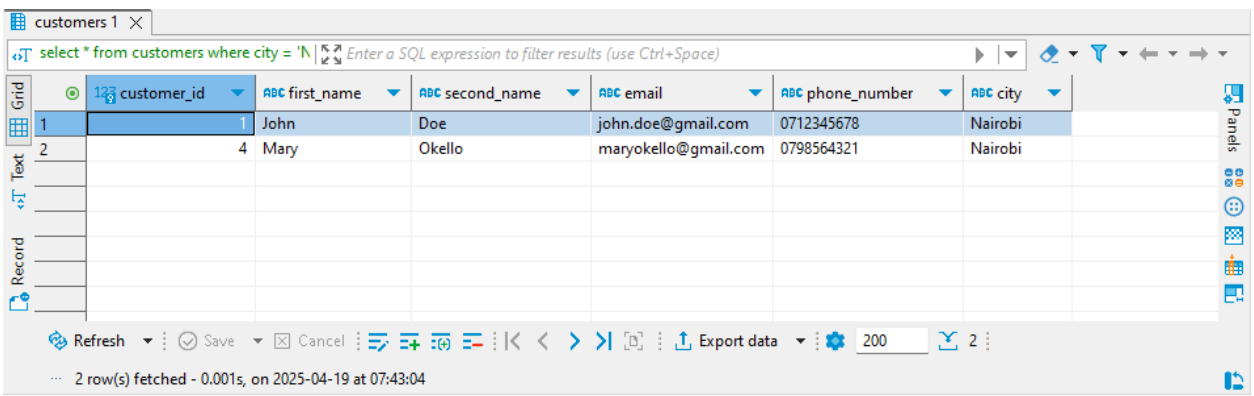
3 row(s) fetched - 0.020s (0.001s fetch), on 2025-04-19 at 07:32:55

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

select * from customers where city = 'N

	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

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

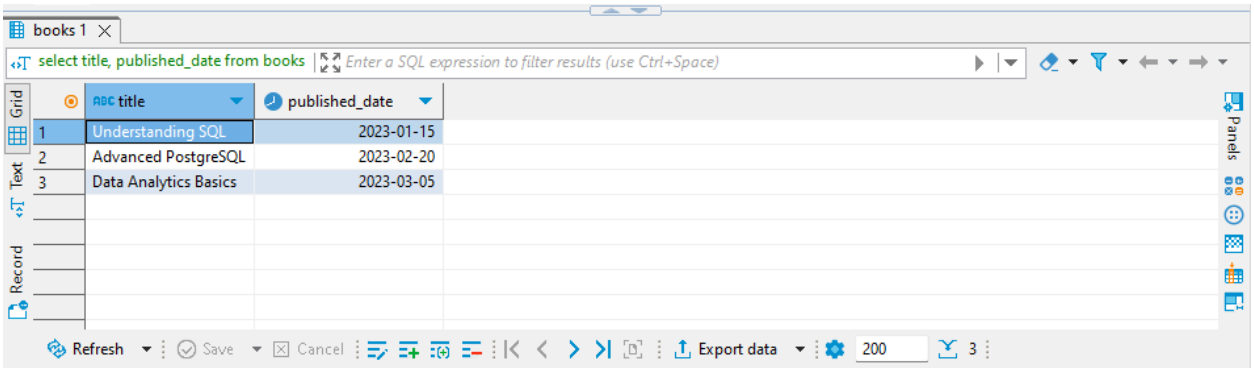
4. Retrieve all book titles that were published in 2023.

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';



Filtering and Sorting

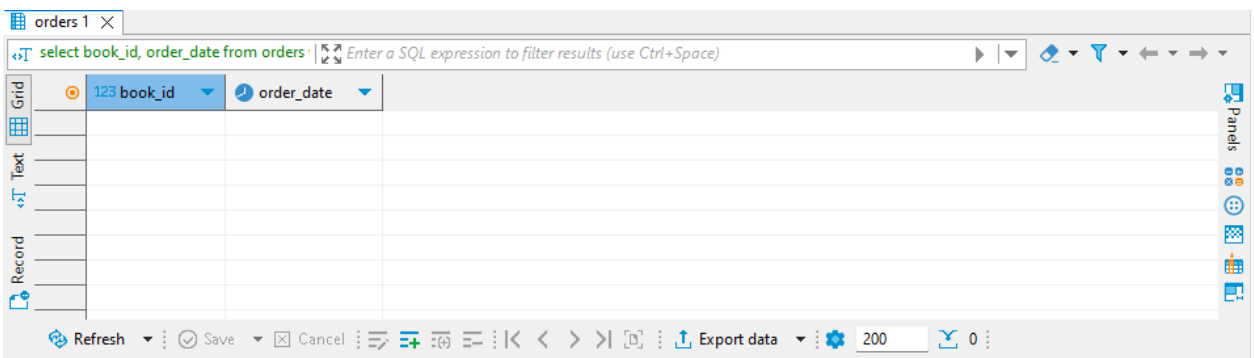
5. Show all orders placed after March 1st, 2025.

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';



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*

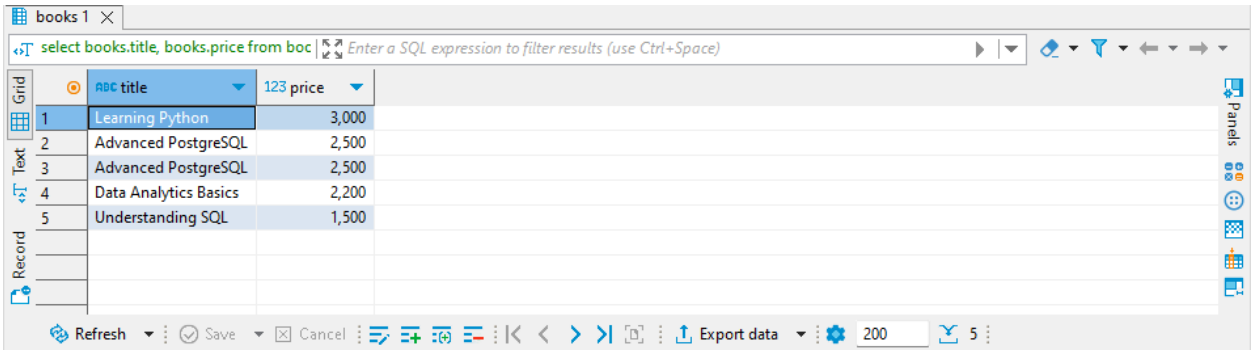
on *books*.book_id = *orders*.book_id

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.

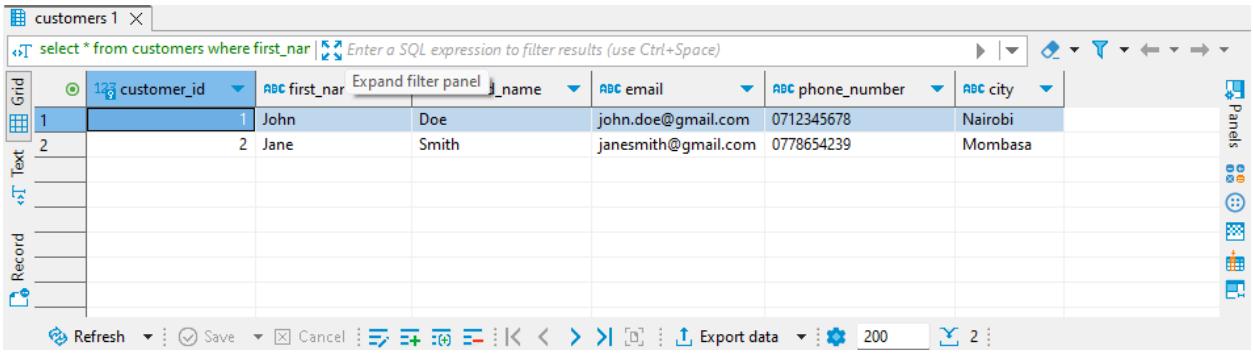


7. Show all customers whose names start with 'J'.

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

select * from *<u>customers</u>*

where *<u>first_name</u>* like 'J%';

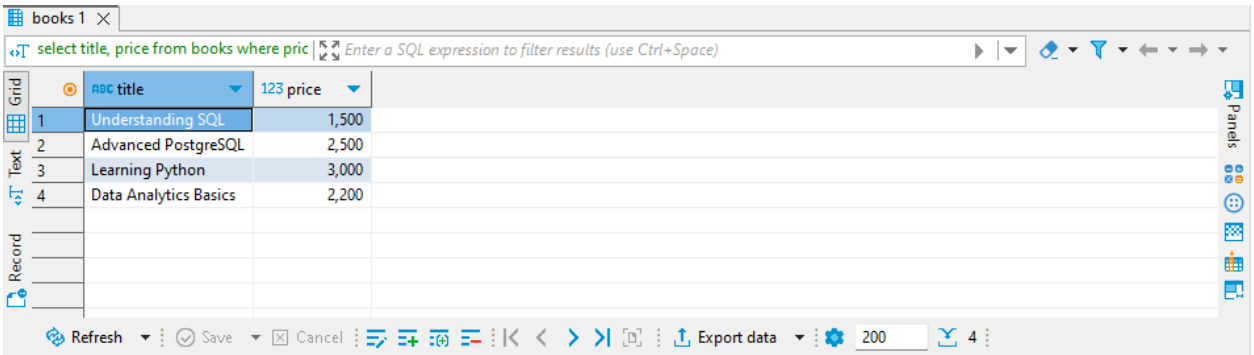


8. List books with prices between 1500 and 3000.

-- 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';



Aggregate Functions and Grouping

9. Count the number of customers in each city.

select *city*, count(*) as *total_customers*

from *customers*

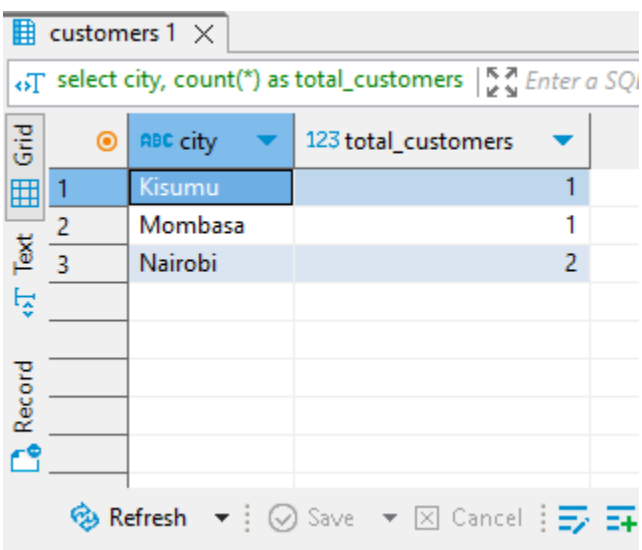
where *city* is not null

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.



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.

orders 1

select customer_id, count(*) as orders_p | Enter a SQL expression to filter results (use Ctrl+Space)

	customer_id	orders_per_customer
1	3	1
2	4	1
3	2	1
4	1	2

Refresh Save Cancel Export data 200 4

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.

Results 1

select avg(price) as Average_price from | Enter a SQL expression to filter results (use Ctrl+Space)

	average_price
1	2,300

Refresh Save Cancel Export data 200 1

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

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

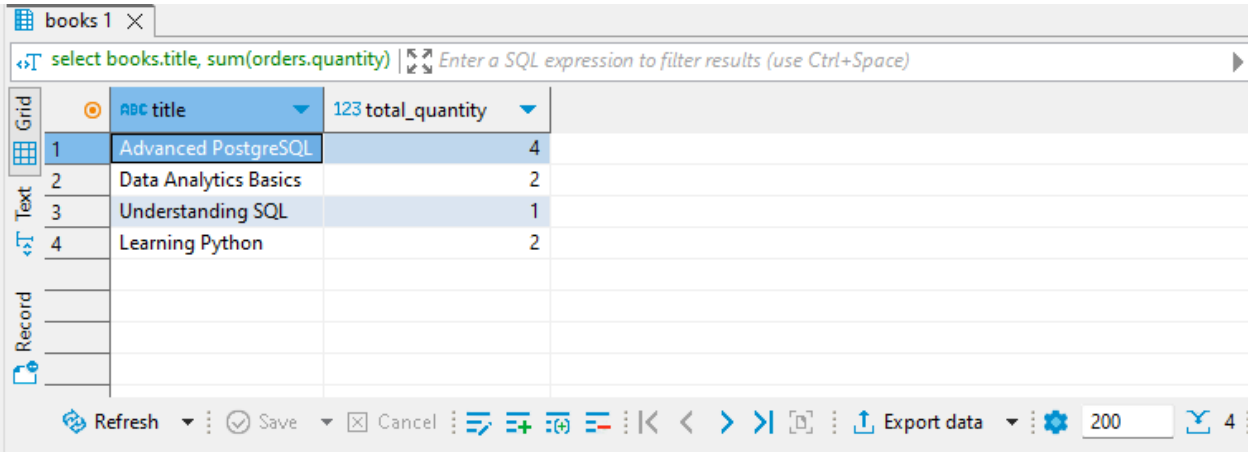
from *books*

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



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

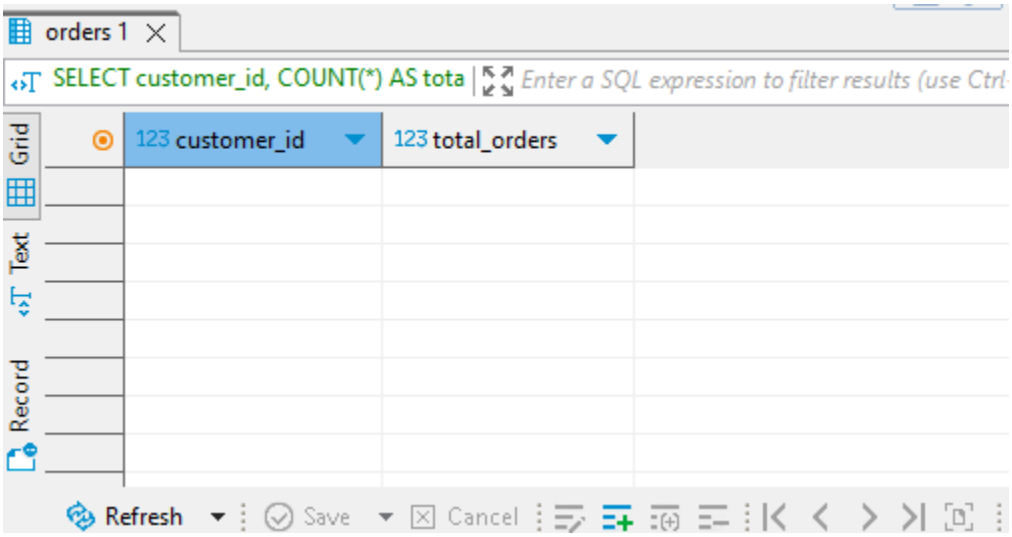
);

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

select title, price from books where pric | Enter a SQL expression to filter results (use Ctrl

	title	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

select customer_id, concat(first_name,' | Enter a SQL expression to filter results (use Ctrl+Space)

	customer_id	full_name	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.

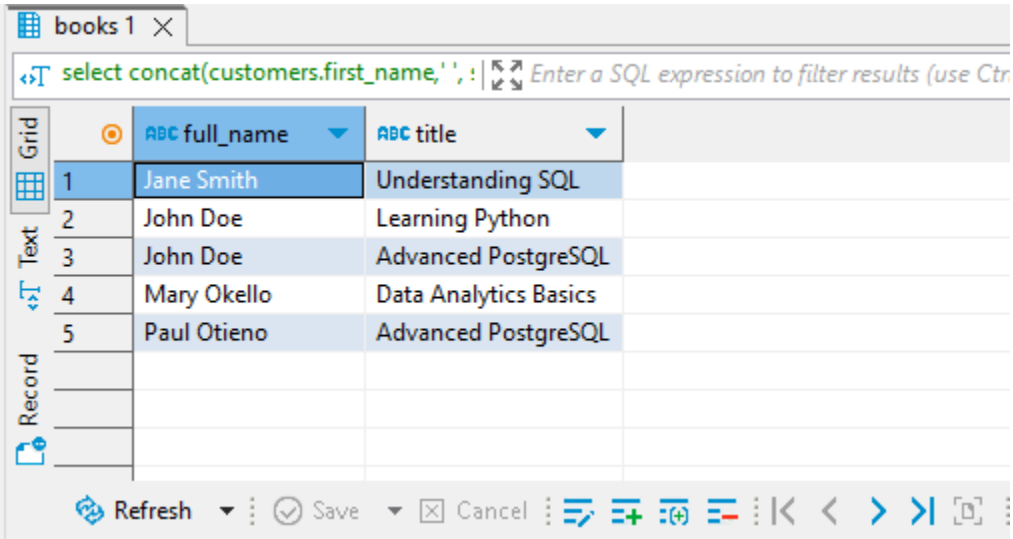
select concat(***customers***.first_name,**' '**, ***second_name***) **as** *full_name*, ***books***.title

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**;



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

select orders.order_id, books.title, books | Enter a SQL expression to filter results (use Ctrl+Space)

	order_id	title	price	quantity	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

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

select concat(first_name,' ',second_nan | Enter a SQL expression to filter results (use Ctrl

	customers_without_orders
1	Kev Bots
2	Marianne Kets

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;

books(+) 1

select books.title, books.book_id, conca | Enter a SQL expression to filter results (use Ctrl+Space)

	title	book_id	customers_with_orders	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

Refresh Save Cancel Export data

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***;

customers 1

select A.customer_id as customer_1, A.l | Enter a SQL expression to filter results (use Ctrl+Space)

	customer_1	name_1	customer_2	customer_2	city
1	1	John	4	Mary	Nairobi
2	1	John	1	John	Nairobi
3	2	Jane	2	Jane	Mombasa
4	3	Paul	3	Paul	Kisumu
5	4	Mary	4	Mary	Nairobi
6	4	Mary	1	John	Nairobi

Refresh Save Cancel Export data

Combined Logic

21. Show all customers who placed more than 2 orders for books priced over 2000.

select ***customers***.customer_id, ***customers***.first_name, **count**(***orders***.order_id) **as** *total_orders*

from ***orders***

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

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

where ***books***.price > 2000

group by ***customers***.customer_id, ***customers***.first_name

having count(***orders***.order_id) > 2;

select customers.customer_id, custome | Enter a SQL expression to filter results (use Ctrl

customer_id	first_name	total_orders

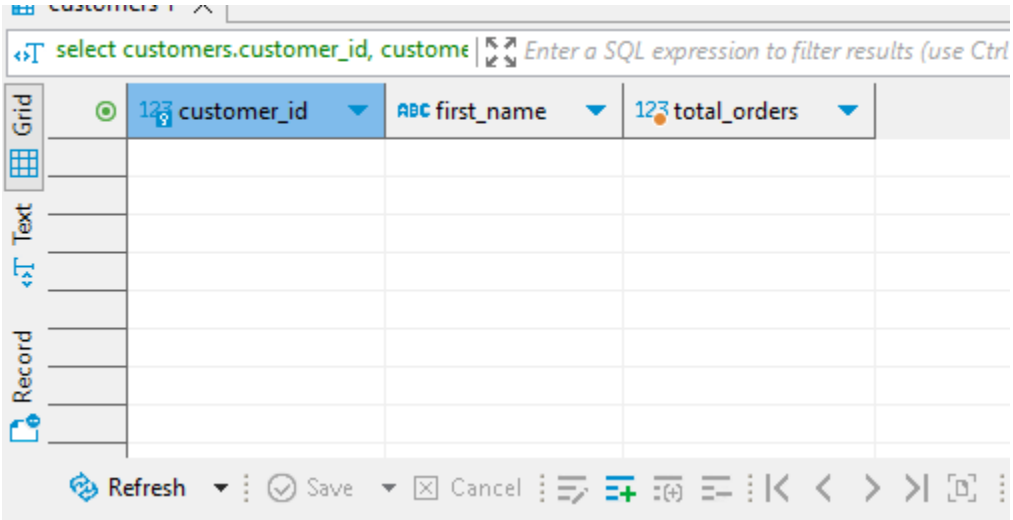
Refresh Save Cancel

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

select ***customer_id***, ***book_id***, ***quantity***

from ***orders***

where ***quantity*** > 1;



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

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

	full_name	total_quantity	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

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

title	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*

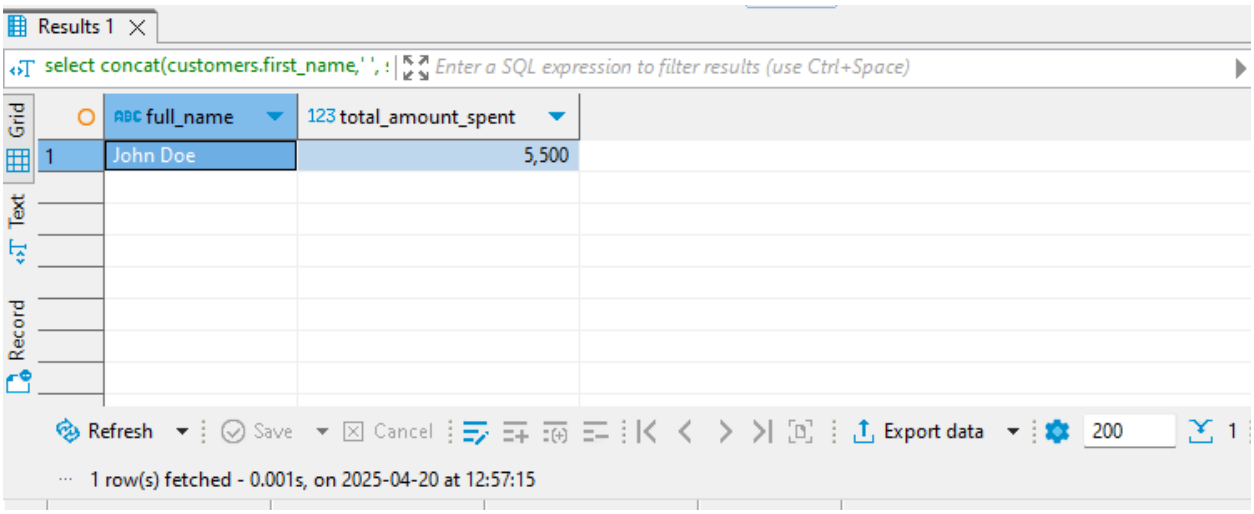
inner join *orders* on *orders*.customer_id = ***customers***.customer_id

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

group by *full_name*

order by *total_amount_spent* **desc**

limit 1;



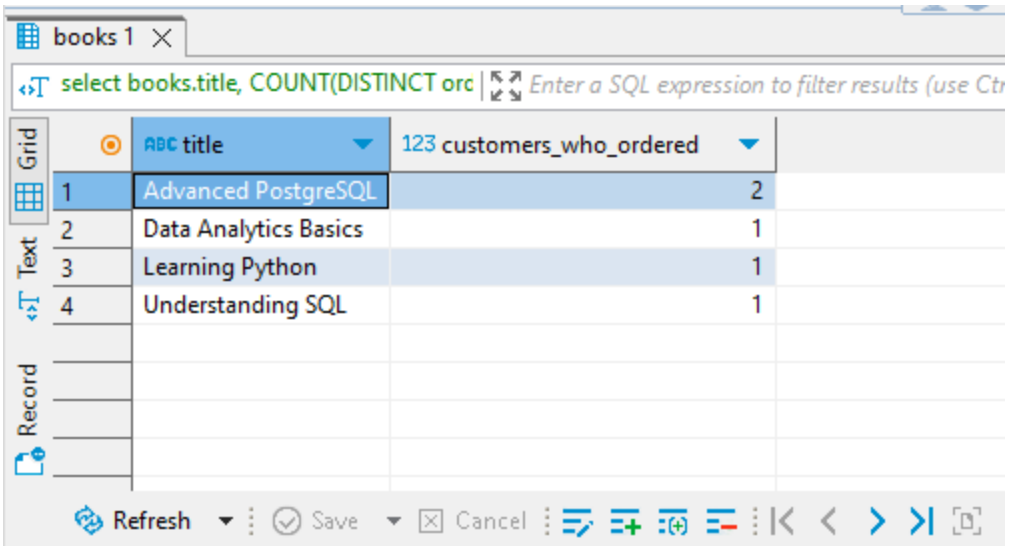
26. Write a query that shows, for each book, the number of different customers who have ordered it.

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 ;



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

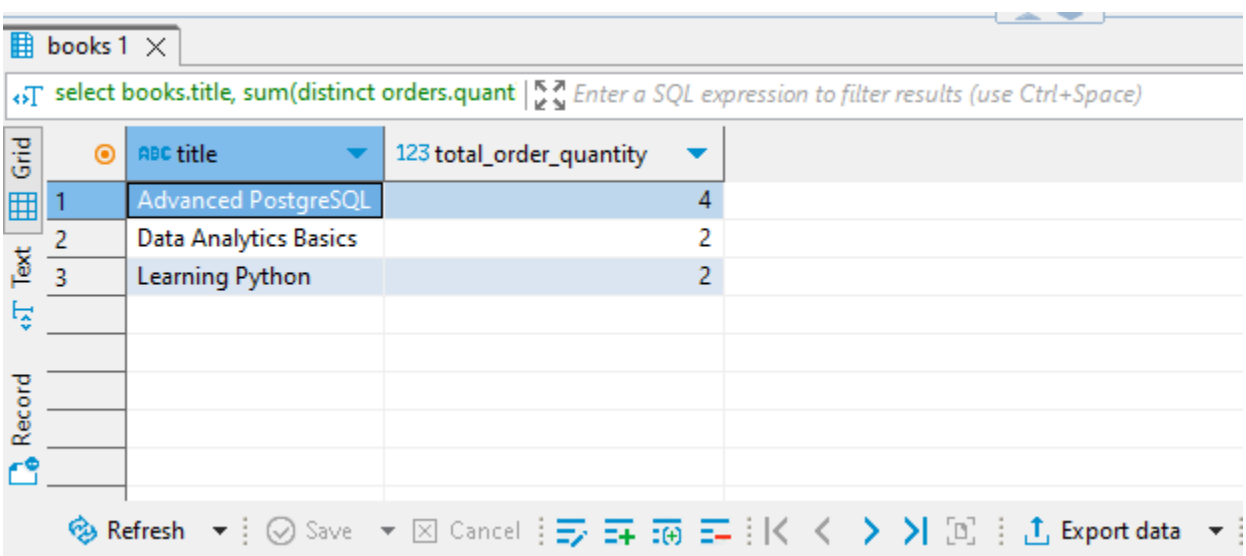
from orders

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);



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**;

Results 1

select concat(customers.first_name,' ', secor | Enter a SQL expression to filter results (use

	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

Refresh Save Cancel