**a. Use SELECT, WHERE, ORDER BY, GROUP BY**

Here’s how to use these fundamental SQL clauses to query your data.

* **SELECT**: To choose specific columns you want to see.

SQL

-- Selects the city and state for all customers

SELECT customer\_city, customer\_state

FROM customers;

**WHERE**: To filter your data based on a condition.

SQL

-- Selects all customers from the state of 'SP' (São Paulo)

SELECT \*

FROM customers

WHERE customer\_state = 'SP';

 **ORDER BY**: To sort your results.

SQL

-- Selects all customers from São Paulo, ordered by their city

SELECT \*

FROM customers

WHERE customer\_state = 'SP'

ORDER BY customer\_city ASC; -- ASC for ascending, DESC for descending

 **GROUP BY**: To group rows that have the same values into summary rows.

SQL

-- Counts the number of customers in each state and orders the result

SELECT customer\_state, COUNT(customer\_id) AS number\_of\_customers

FROM customers

GROUP BY customer\_state

ORDER BY number\_of\_customers DESC;

* **INNER JOIN**: Returns records that have matching values in both tables.

SQL

-- Selects customers who have placed an order

SELECT c.customer\_unique\_id, o.order\_id, o.amount

FROM customers AS c

INNER JOIN orders AS o ON c.customer\_id = o.customer\_id;

* **LEFT JOIN**: Returns all records from the left table (customers), and the matched records from the right table (orders).

SQL

-- Selects all customers and their orders, if they have any

SELECT c.customer\_unique\_id, o.order\_id, o.amount

FROM customers AS c

LEFT JOIN orders AS o ON c.customer\_id = o.customer\_id;

* **RIGHT JOIN**: Returns all records from the right table (orders), and the matched records from the left table (customers). (Note: SQLite does not directly support RIGHT JOIN, but you can achieve the same result by swapping the tables and using a LEFT JOIN).

SQL

-- This is how you would write it in standard SQL

-- SELECT c.customer\_unique\_id, o.order\_id, o.amount

-- FROM customers AS c

-- RIGHT JOIN orders AS o ON c.customer\_id = o.customer\_id;

-- In SQLite, you would do this:

SELECT c.customer\_unique\_id, o.order\_id, o.amount

FROM orders AS o

LEFT JOIN customers AS c ON c.customer\_id = o.customer\_id;

**c. Write subqueries**

A subquery is a query nested inside another query.

SQL

-- Selects all customers from cities that have more than 2000 customers

SELECT customer\_unique\_id, customer\_city, customer\_state

FROM customers

WHERE customer\_city IN (

SELECT customer\_city

FROM customers

GROUP BY customer\_city

HAVING COUNT(customer\_id) > 2000

);

**d. Use aggregate functions (SUM, AVG)**

Aggregate functions perform a calculation on a set of values and return a single value.

* **AVG**: Calculates the average value.

SQL

-- Calculates the average customer\_zip\_code\_prefix for each state

SELECT customer\_state, AVG(customer\_zip\_code\_prefix) AS avg\_zip\_prefix

FROM customers

GROUP BY customer\_state;

* **SUM**: Calculates the sum of values. Using the hypothetical orders table:

SQL

-- Calculates the total amount of all orders

SELECT SUM(amount) AS total\_revenue

FROM orders;

You can also use SUM with GROUP BY:

SQL

-- Calculates the total order amount for each customer

SELECT c.customer\_unique\_id, SUM(o.amount) AS total\_spent

FROM customers AS c

JOIN orders AS o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_unique\_id

ORDER BY total\_spent DESC;

**e. Create views for analysis**

A view is a virtual table based on the result-set of an SQL statement. It’s useful for simplifying complex queries.

SQL

-- Creates a view that only contains customers from Rio de Janeiro ('RJ')

CREATE VIEW rio\_de\_janeiro\_customers AS

SELECT \*

FROM customers

WHERE customer\_state = 'RJ';

Now you can query the view as if it were a real table:

SQL

-- Select all customers from the 'rio\_de\_janeiro\_customers' view

SELECT \* FROM rio\_de\_janeiro\_customers;

**f. Optimize queries with indexes**

An index helps to speed up the retrieval of rows from a table. It's particularly useful for columns that are frequently used in WHERE clauses or JOIN conditions.

SQL

-- Creates an index on the customer\_state column

CREATE INDEX idx\_customer\_state ON customers (customer\_state);

With this index, a query like the one below will be much faster because the database can quickly find all rows where customer\_state is 'SP' without scanning the entire table.

SQL

-- This query will now be faster

SELECT \*

FROM customers

WHERE customer\_state = 'SP';

You can also create an index on the customer\_id which is a primary key and likely to be used in joins.

-- Creates an index on the customer\_id column

CREATE INDEX idx\_customer\_id ON customers (customer\_id);