About
This is a list of my favorite SQL Operators, Functions and Filters in PostgreSQL. Jonathan Maas www.theprototyper.design

Updated	1/6/2025

Name	Example	Description
SELECT	SELECT * FROM table_name	SELECT * chooses everything from a table.
LIMIT	SELECT * FROM table_name LIMIT 10	Limits the output by the number of rows
ORDER BY	SELECT item_id, item_amount FROM table_name ORDER BY item_ amount ASC	Orders the table by a column.
ORDER BY (Multiple)	SELECT * FROM table_name ORDER BY column1 ASC, column2 DESC;	Orders the table by two columns in case the first is a repeat.
ORDER BY RANDOM()	SELECT * FROM table_name ORDER BY RANDOM() LIMIT 10; -- Get 10 random rows	Retrieves random rows.

TABLESAMPLE SYSTEM()	SELECT * FROM table_name TABLESAMPLE SYSTEM(1) -- Get approximate 1% of rows LIMIT 10;	Retrieves a percentage of random rows

GROUP BY	SELECT
	  department,
	  COUNT(*) as items_count
	FROM table_name
	GROUP BY department;
		Groups the output by a certain set of column values.

WHERE	SELECT item_1, status, item_3 FROM table_name WHERE status = 'Good'	Filters the output by a value of a certain column
BETWEEN	SELECT * FROM table_name WHERE price BETWEEN 100 AND 200;	Filters the output to show values in a range.
AND/OR	SELECT item_1, status, item_3 FROM table_name WHERE status = 'Good' AND (amount BETWEEN 9000 AND 10000) OR (amount > 15000)	Filters the output with a few more options.
LIKE	SELECT * FROM table_name WHERE last_name LIKE 'Smith%'; -- Names starting with 'Smith' but capitalized just like this	
ILIKE	SELECT * FROM table_name WHERE last_name ILIKE 'smith%'; -- Names starting with 'smith', but can be any case	
= and IN	= is for exact matches, ie item_name = 'cake', IN is multiple values ie item_name IN 'cake', 'pie'	

% (Wildcard)	SELECT * FROM table_name
	WHERE
	  filename LIKE '%report%2024%' -- Contains report and 2024
	  OR description ILIKE '%laptop%' -- Matches laptop, LAPTOP, LaPtOp
	  OR path LIKE '%/img/%/old/%'; -- Nested path pattern
		% means can be any amount of characters

_ (Wildcard)	SELECT * FROM table_name
	WHERE
	  sku LIKE 'A_123' -- Matches A1123, AB123, etc.
	  OR code LIKE '___' -- Exactly 3 characters
	  OR phone LIKE '+1__________'; -- Exactly 11 digits after +1
		_ is a set character Wildcard

LENGTH()	/* Basic Syntax */
	SELECT
	  first_name,
	  LENGTH(first_name) as name_length
	FROM users;
	
	-- Filters by Length
	SELECT *
	FROM products
	WHERE LENGTH(product_name) > 20;
		Finds the length of a string or number, returns null if it is null

CONCATENATE or ||	/* With || */
	SELECT
	  first_name || ' ' || last_name as full_name,
	  city || ', ' || country as location
	FROM customers;
	
	-- With CONCAT()
	SELECT
	  CONCAT(first_name, ' ', last_name) as full_name,
	  CONCAT(street, ', ', city, ', ', country) as address
	FROM employees;
		Concatenates strings, numbers or dates - will change date to text.
		Two ways of saying it - || or CONCAT

EXTRACT	/* Extracting from a timestamp */
	SELECT
	  order_date_timestamp,
	  EXTRACT(YEAR FROM order_date_timestamp) as year,
	  EXTRACT(MONTH FROM order_date_timestamp) as month,
	  EXTRACT(DAY FROM order_date_timestamp) as day
	FROM orders;
		This is good for extracting part of a timestamp or other time functions.

POSITION	/* Basic syntax */
	SELECT POSITION('world' IN 'Hello world'); -- Returns 7
	
	-- Finding the @ in a string to parse out a half of the email
	
	SELECT
	  email,
	  POSITION('@' IN email) as at_symbol_position
	FROM users;
		Finds a place in a string, returns 0 if not found.

SUBSTRING	/* Basic Syntax */
	SELECT
	  SUBSTRING('Hello World' FROM 1 FOR 5); -- Returns 'Hello'
	SELECT
	  SUBSTRING('Hello World', 7, 5); -- Returns 'World'
	
	-- Returns parts of phone number
	  SELECT
	  phone_number,
	  SUBSTRING(phone_number FROM 1 FOR 3) as area_code,
	  SUBSTRING(phone_number FROM 4 FOR 3) as prefix
	FROM contacts;
		Returns parts of strings or numbers, position starts at 1 and not 0.

TO_CHAR	/* Basic Syntax */ SELECT order_date_timestamp, TO_CHAR(order_date_timestamp,, 'MM/DD/YYYY') as us_date, TO_CHAR(order_date_timestamp,, 'DD Month YYYY') as full_date, TO_CHAR(order_date_timestamp,, 'Dy, DD Mon YYYY') as custom_date FROM orders; -- With Current Timestamp SELECT CURRENT_TIMESTAMP, TO_CHAR(CURRENT_TIMESTAMP, 'MM/DD/YYYY') as date1, TO_CHAR(CURRENT_TIMESTAMP, 'DD Month YYYY') as date2, TO_CHAR(CURRENT_TIMESTAMP, 'Dy, DD Mon YYYY') as date3	A way of turning timestamp numbers as a readable format
CASE WHEN	SELECT order_id, amount, CASE WHEN amount < 100 THEN 'Small' WHEN amount < 1000 THEN 'Medium' ELSE 'Large' END AS order_size FROM table_name;	

COALESCE	SELECT COALESCE(actual_arrival-scheduled_arrival, '0:00') as time_delay FROM airport_schedule;	COALESCE returns values that are not null, and returns a specified value if the value is null. In the previous example, there are null actual_arrivals, and it returned those instances as 0:00 to keep in line with the time value.
CAST	/* String to Integer */ SELECT CAST('100' AS INTEGER), -- Or using :: syntax '100'::INTEGER, -- Integer to String CAST(100 AS VARCHAR)	CAST changes values from one type to another
CAST and COALESCE	SELECT rental_date, COALESCE(CAST(return_date AS VARCHAR), 'not returned') FROM rentals	Sometimes you have to combine CAST and COALESCE. If a return_date is a Time stamp, you can turn it into a VARCHAR so that you can return a VARCHAR value like 'not returned.'
INNER JOIN	SELECT * FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;	Takes two tables and returns a single table with only the values where a single shared key table are identical. All NULL values will be excluded.

FULL OUTER JOIN	/* Basic Syntax */
	SELECT * FROM table1
	FULL OUTER JOIN table2
	ON table1.column = table2.column;
	
	/* FULL OUTER JOIN to find Mismatches */
	SELECT
	  customers.customer_name,
	  orders.order_id
	FROM customers
	FULL OUTER JOIN orders
	ON customers.customer_id = orders.customer_id
	WHERE customers.customer_id IS NULL
	  OR orders.customer_id IS NULL;
		Takes two tables and returns a single table with all values, uniting the tables with the single shared key table. All NULL values are included.

LEFT JOIN	SELECT * FROM customers -- This is the LEFT table LEFT JOIN orders -- This is the RIGHT table ON customers.id = orders.customer_id;	Takes two tables, returns ALL records from the LEFT table, and only the records with a matching KEY value from the RIGHT table.
RIGHT JOIN	SELECT * FROM customers -- This is the LEFT table RIGHT JOIN orders -- This is the RIGHT table ON customers.id = orders.customer_id;	Takes two tables, returns ALL records from the RIGHT table, and only the records with a matching KEY value from the LEFT table.
MULTIPLE JOINs (two tables, for specificity	SELECT * FROM orders INNER JOIN shipments ON orders.order_id = shipments.order_id AND orders.customer_id = shipments.customer_id;	This adds specificity to the filtering. It is good for names as well - there might be multiple first_name s, and multiple last_name s, but a first_name + last_name combination is good

MULTIPLE JOINs (multiple tables)	/* This is a query with four tables, customer is linked to address, address is linked to city, and city is linked to country */ SELECT first_name, last_name, email, country.country FROM customer LEFT JOIN address a ON customer.address_id = a.address_id LEFT JOIN city ON a.city_id = city.city_id LEFT JOIN country ON city.country_id = country.country_id WHERE country.country = 'Brazil'	
UNION	SELECT first_name, last_name, 'source_1' AS origin FROM table1 UNION SELECT first_name, last_name, 'source_2' FROM table2	This combines matching tables, or matching columns into one big table or set of big columns.
SUBQUERY (technique)	SELECT * FROM payment /* Below making a subquery to make sure amount is bigger than average */ WHERE (amount > (SELECT AVG(amount) from payment))	A query within a query, ie making another SELECT query to compare a value against an average.

SUBQUERY + FROM	/* Subquery FROM - end with the top clause, refer to the alias and say it is from*/ SELECT AVG(total_payment) /* FROM says the second clause is done first */ FROM /* Subquery FROM - start with the second clause, give it an alias*/ (SELECT customer_id, SUM(amount) AS total_payment FROM payment GROUP BY customer_id)	A way to subquery another calculation.
SUBQUERY SELECT + SELECT	/* You can add a column with a Select subquery provided it is the same ie a Sum or an Avg */ SELECT *, (SELECT ROUND(AVG(amount), 2) AS avg_amount FROM payment) FROM payment /* You can add a column of varying values if you limit it to one */ SELECT *, (SELECT amount AS first_amount_in_list FROM payment LIMIT 1) FROM payment	A way to add another column of identical values to a query.
CORRELATED SUBQUERY	SELECT * FROM payment p1 WHERE amount = (SELECT MAX(amount) FROM payment p2 WHERE p1.customer_id = p2.customer_id) ORDER BY customer_id	Correlated subquery finds the comparison of a column compared to itself, ie for each item in a category purchased, which items are higher than average for that category For this example, it shows only payments that have the highest amount per customer

CORRELATED SUBQUERY AS SELECT	SELECT payment_id, customer_id, amount, (SELECT MAX(amount) FROM payment p2 WHERE p1.customer_id = p2.customer_id) AS max_amount FROM payment p1	This adds a correlated subquery as a column
CREATE DATABASE	CREATE TABLE director (/* make a Primary Key, which implies that it is Unique and Not Null, Serial adds incremental value */ director_id SERIAL PRIMARY KEY, /* make another few columns */ director_account_name VARCHAR(20) UNIQUE, first_name VARCHAR (50), /* Give a default value to this column */ last_name VARCHAR (50) DEFAULT 'Not Specified', /* This column is a date */ date_of_birth DATE, /* This acts as a Foreign key to another table */ address_id INT REFERENCES address(address_id))	Commented Create table command with a few ways of giving each column a constraint
DROP DATABASE	/* be warned that you be wary of this */ DROP DATABASE database_name;	Be warned - it is an easy command but be wary

ALTER TABLE	ALTER TABLE director -- change column to a different type ALTER COLUMN director_account_name TYPE VARCHAR(30) -- change column to have no default ALTER COLUMN last_name DROP DEFAULT -- change constraint on column so it is not null ALTER COLUMN last_name SET NOT NULL -- add a column with type ADD COLUMN email VARCHAR(40) -- rename a column RENAME COLUMN director_account_name TO account_name -- rename the table RENAME TO directors	Here is a few alterations with what each one is commented out
TRUNCATE TABLE	TRUNCATE table_name -- or TRUNCATE TABLE table_name	Keeps table, deletes what is inside the table
CHECK	/* Making a table with a few checks */ CREATE TABLE employees (id SERIAL PRIMARY KEY, -- check to see if name is greater than 1 character name TEXT CHECK (length(name)>1), -- check to see if age is between a certain range age INTEGER CHECK (age >= 18 AND age < 65), -- check to make sure salary is a positive number salary NUMERIC CHECK (salary > 0));	Check sets a few constraints when you are setting up a table for values. You can give the checks names but if not it will assign it a name.

ALTER TABLE CHECK	/* Query 1 - Drop existing constraint - if you have not made its name, you can find it through PG Admin */ ALTER TABLE songs DROP CONSTRAINT songs_price_check; /* Query 2 - Make new constraint with new value, and give it the same name */ ALTER TABLE songs ADD CONSTRAINT songs_price_check CHECK (price >= .99);	You can alter tables with checks, and also drop checks. You can find the constraint name through PG Admin, and you must do two queries - first to drop the existing constraint, and then another to make the new one, often with the same name.