PostgreSQL Cheat Sheet

www.databasestar.com

SELECT Query

```
SELECT col1, col2
FROM table
JOIN table2 ON table1.col = table2.col
WHERE condition
GROUP BY column_name
HAVING condition
ORDER BY col1 ASC|DESC;
```

SELECT Keywords

DISTINCT: Removes duplicate results

```
SELECT DISTINCT product_name
FROM product;
```

BETWEEN: Matches a value between two other values (inclusive)

```
SELECT product_name
FROM product
WHERE price BETWEEN 50 AND 100;
```

IN: Matches to any of the values in a list

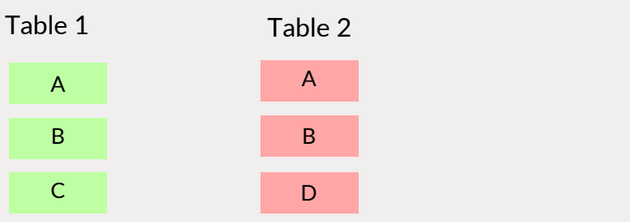
```
SELECT product_name
FROM product
WHERE category IN
('Electronics', 'Furniture');
```

LIKE: Performs wildcard matches using _ or %

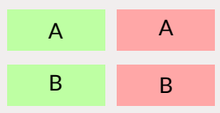
```
SELECT product_name
FROM product
WHERE product_name
LIKE '%Desk%';
```

Joins

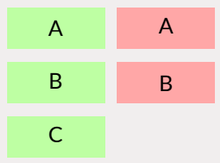
```
SELECT t1.*, t2.*
FROM t1
join_type t2 ON t1.col = t2.col;
```



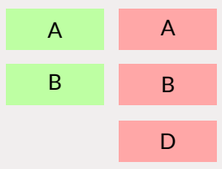
INNER JOIN: show all matching records in both tables.



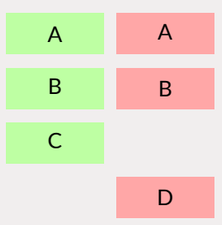
LEFT JOIN: show all records from left table, and any matching records from right table.



RIGHT JOIN: show all records from right table, and any matching records from left table.



FULL JOIN: show all records from both tables, whether there is a match or not.



CASE Statement

Simple Case

```
CASE name
  WHEN 'John' THEN 'Name John'
  WHEN 'Steve' THEN 'Name Steve'
  ELSE 'Unknown'
END
```

Searched Case

```
CASE
  WHEN name='John' THEN 'Name John'
  WHEN name='Steve' THEN 'Name Steve'
  ELSE 'Unknown'
END
```

Common Table Expression

```
WITH queryname AS (
  SELECT col1, col2
  FROM firsttable)
SELECT col1, col2..
FROM queryname...;
```

Modifying Data

Insert

```
INSERT INTO tablename
(col1, col2...)
VALUES (val1, val2);
```

Insert from a Table

```
INSERT INTO tablename
(col1, col2...)
SELECT col1, col2...
```

Insert Multiple Rows

```
INSERT INTO tablename
(col1, col2...) VALUES
(valA1, valB1),
(valA2, valB2),
(valA3, valB3);
```

Update

```
UPDATE tablename
SET col1 = val1
WHERE condition;
```

Update with a Join

```
UPDATE t
SET col1 = val1
FROM tablename t
INNER JOIN table x
ON t.id = x.tid
WHERE condition;
```

Delete

```
DELETE FROM tablename
WHERE condition;
```

Indexes

Create Index

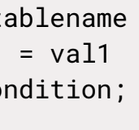
```
CREATE INDEX indexname
ON tablename (cols);
```

Drop Index

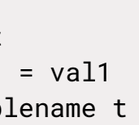
```
DROP INDEX indexname;
```

Set Operators

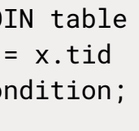
UNION: Shows unique rows from two result sets.



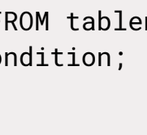
UNION ALL: Shows all rows from two result sets.



INTERSECT: Shows rows that exist in both result sets.



EXCEPT: Shows rows that exist in the first result set but not the second.



Aggregate Functions

- SUM: Finds a total of the numbers provided
- COUNT: Finds the number of records
- AVG: Finds the average of the numbers provided
- MIN: Finds the lowest of the numbers provided
- MAX: Finds the highest of the numbers provided

Common Functions

- LENGTH(string): Returns the length of the provided string
- POSITION(string IN substring): Returns the position of the substring within the specified string.
- CAST(expression AS datatype): Converts an expression into the specified data type.
- NOW: Returns the current date, including time.
- CEIL(input_val): Returns the smallest integer greater than the provided number.
- FLOOR(input_val): Returns the largest integer less than the provided number.
- ROUND(input_val, [round_to]): Rounds a number to a specified number of decimal places.
- TRUNC(input_value, num_decimals): Truncates a number to a number of decimals.
- REPLACE(whole_string, string_to_replace, replacement_string): Replaces one string inside the whole string with another string.
- SUBSTRING(string, [start_pos], [length]): Returns part of a value, based on a position and length.

Create Table

Create Table

```
CREATE TABLE tablename (
  column_name data_type
);
```

Create Table with Constraints

```
CREATE TABLE tablename (
  column_name data_type NOT NULL,
  CONSTRAINT pkname PRIMARY KEY (col),
  CONSTRAINT fkname FOREIGN KEY (col)
REFERENCES other_table(col_in_other_table),
  CONSTRAINT ucname UNIQUE (col),
  CONSTRAINT ckname CHECK (conditions)
);
```

Create Temporary Table

```
CREATE TEMP TABLE tablename (
  colname datatype
);
```

Drop Table

```
DROP TABLE tablename;
```

Alter Table

Add Column

```
ALTER TABLE tablename ADD COLUMN
columnname datatype;
```

Drop Column

```
ALTER TABLE tablename DROP COLUMN
columnname;
```

Modify Column

```
ALTER TABLE tablename ALTER COLUMN
columnname TYPE newdatatype;
```

Rename Column

```
ALTER TABLE tablename RENAME COLUMN
currentname TO newname;
```

Add Constraint

```
ALTER TABLE tablename ADD CONSTRAINT
constraintname constrainttype
(columns);
```

Drop Constraint

```
ALTER TABLE tablename DROP
constraint_type constraintname;
```

Rename Table

```
ALTER TABLE tablename
RENAME TO newtablename;
```

Window/Analytic Functions

```
function_name ( arguments ) OVER (
[query_partition_clause]
[ORDER BY order_by_clause
[windowing_clause] ] )
```

Example using RANK, showing the student details and their rank according to the fees_paid, grouped by gender:

```
SELECT
student_id, first_name, last_name, gender, fees_paid,
RANK() OVER (
  PARTITION BY gender ORDER BY fees_paid
) AS rank_val
FROM student;
```

Subqueries

Single Row

```
SELECT id, last_name, salary
FROM employee
WHERE salary = (
  SELECT MAX(salary)
  FROM employee
);
```

Multi Row

```
SELECT id, last_name, salary
FROM employee
WHERE salary IN (
  SELECT salary
  FROM employee
  WHERE last_name LIKE 'C%'
);
```

PostgreSQL Data Types

www.databasestar.com

Numeric

SMALLINT	A small integer number Range -32,768 to +32,767
INTEGER	An integer number Range -2,147,483,648 to +2,147,483,647,
BIGINT	A large integer number Range -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
DECIMAL	A decimal number with precision. Range: up to 131,072 digits before the decimal point; up to 16,383 digits after the decimal point
NUMERIC (p, s)	A decimal number with precision of “p” and scale of “s” Range: up to 131,072 digits before the decimal point; up to 16,383 digits after the decimal point
REAL	A floating-point variable-precision number. 6 decimal digits precision
DOUBLE PRECISION	A floating-point variable-precision number. 15 decimal digits precision
SMALLSERIAL	A small automatically incrementing integer. Range: 1 to 32,767
SERIAL	An automatically incrementing integer. Range: 1 to 2,147,483,647
BIGSERIAL	A large automatically incrementing integer. Range: 1 to 9,223,372,036,854,775,807
MONEY	A currency amount. Range: -92,233,720,368,547,758.08 to +92,233,720,368,547,758.07

Date

TIMESTAMP (p)	A date and time value with no time zone. Precision “p” can be specified which is the number of fractional seconds. Range: 4713 BC to 294276 AD
TIMESTAMP (p) WITH TIME ZONE	A date and time value with time zone. Precision “p” can be specified which is the number of fractional seconds. Range: 4713 BC to 294276 AD
DATE	A date but no time. Range: 4713 BC to 5874897 AD
TIME (p)	A time of day with no date Precision “p” can be specified which is the number of fractional seconds. Range: 00:00:00 to 24:00:00
TIME (p) WITH TIME ZONE	A time of day with no date and a time zone Precision “p” can be specified which is the number of fractional seconds. Range: 00:00:00+1459 to 24:00:00-1459
INTERVAL [fields] (p)	An interval of time. Precision “p” can be specified which is the number of fractional seconds. The parameter “fields” can be used to specify the type of data (e.g. YEAR, MONTH, DAY TO HOUR) Range: -178,000,000 years to 178,000,000 years

Character

CHARACTER VARYING (n)	A variable-length string up to “n” characters. Range: Up to 10,485,760 characters (1GB)
VARCHAR (n)	A variable-length string up to “n” characters. Range: Up to 10,485,760 characters (1GB)
CHARACTER (n)	A fixed-length string, padded to a length of “n” characters. Range: Up to 10,485,760 characters (1GB)
CHAR (n)	A fixed-length string, padded to a length of “n” characters. Range: Up to 10,485,760 characters (1GB)
TEXT	A variable length string
BYTEA	A variable-length binary string. Similar to BLOB
ENUM	A set of values that can be used for a column.
JSON	Stores JSON data
JSONB	Stores JSON data in binary format, and can support indexing.

Other

BOOLEAN	Stores either true or false. True, yes, on, 1. False, no off, 0.
POINT	A point of geometry
LINE	A line of geometry
LSEG	A segment of a line
BOX	A rectangular box
PATH	An open path
POLYGON	A polygon or shape
CIRCLE	A circle
CIDR	Stores IPv4 and IPv6 network addresses
INET	Stores IPv4 and IPv6 hosts and network addresses
MACADDR	Stores MAC addresses using 6 bytes.
MACADDR8	Stores MAC addresses using 8 bytes (the EUI-64 format)
TSVECTOR	A sorted list of words
TSQUERY	A list of words to be searched for
UUID	Stores a Universally Unique Identifier (or GUID). A 128-bit generated value
XML	Stores XML data
PG_LSN	PostgreSQL Log Sequence Number
TXID_SNAPSHOT	A user-level transaction ID snapshot

PostgreSQL JSON Cheat Sheet

www.databasestar.com

JSON Example

```
{
  “color”: “black”,
  “drawers”: [
    {
      “side”: “left”,
      “height”: “30cm”
    },
    {
      “side”: “left”,
      “height”: “40cm”
    }
  ],
  “material”: “metal”
}
```

Data Types

JSON: regular JSON

JSONB: JSON Binary. The **recommended data type**.

Creating a JSON Field

Create Table with JSONB field:

```
CREATE TABLE product (
  id INT,
  product_name CHARACTER VARYING(200),
  attributes JSONB
);
```

Create Table with JSON field:

```
CREATE TABLE product (
  id INT,
  product_name CHARACTER VARYING(200),
  attributes JSON
);
```

Insert JSON Data

Insert statement:

```
INSERT INTO product (id, product_name, attributes)
VALUES (
1,
'Chair',
'{"color":"brown", "material":"wood",
"height":"60cm"}'
);
```

Insert array:

```
INSERT INTO product (id, product_name, attributes)
VALUES (
3,
'Side Table',
'{"color":"brown", "material":["metal", "wood"]}'
);
```

Insert with JSONB_BUILD_OBJECT:

```
INSERT INTO product (id, product_name, attributes)
VALUES (
4,
'Small Table',
JSONB_BUILD_OBJECT(
  'color', 'black', 'material', 'plastic'
)
);
```

Other functions for inserting:

- TO_JSON and TO_JSONB
- ARRAY_TO_JSON
- ROW_TO_JSON
- JSON_BUILD_ARRAY and JSONB_BUILD_ARRAY
- JSON_OBJECT and JSONB_OBJECT

Selecting

Select with key and value:
(displays a value such as "blue" **with surrounding quotes)**

```
SELECT
id,
product_name,
attributes -> 'color' AS color_key
FROM product;
```

Select with key and value:
(displays a value such as "blue" **without surrounding quotes)**

```
SELECT
id,
product_name,
attributes ->> 'color' AS color_key
FROM product;
```

Select an array value with key and value:

```
SELECT
id,
product_name,
attributes -> 'drawers' -> 1 AS drawer_value
FROM product;
```

Select an array value with key and value as object or as text

```
SELECT
id,
product_name,
attributes #> '{drawers, 1}' AS drawers_element,
attributes #>> '{drawers, 1}' AS drawers_text
FROM product;
```

Filtering

Filtering a value with key and value:

```
SELECT
id,
product_name,
attributes
FROM product
WHERE attributes ->> 'color' = 'brown';
```

Filtering where a key exists:

```
SELECT
id,
product_name,
attributes
FROM product
WHERE attributes ? 'drawers' = true;
```

Split Data into Rows

Split each element into separate rows:

```
SELECT
id,
product_name,
JSONB_EACH(attributes)
FROM product;
```

Get all keys:

```
SELECT
id,
product_name,
JSONB_OBJECT_KEYS(attributes)
FROM product;
```

Updating

Update field by concatenating:

```
UPDATE product
SET attributes =
attributes || '{"width":"100cm"}'
WHERE id = 1;
```

Update field using jSONB_SET:

```
UPDATE product
SET attributes =
JSONB_SET(attributes, '{height}', '"75cm"')
WHERE id = 1;
```

Deleting

Delete based on filter:

```
DELETE FROM product
WHERE attributes ->> 'color' = 'brown';
```

Remove attribute from field:

```
UPDATE product
SET attributes = attributes - 'height'
WHERE id = 1;
```