PostgreSQL Cheat Sheet

LearnSQL.com

PostgreSQL is an open-source relational database management system. Known for its robust features, extensibility, and adherence to standards, it is a powerful and widely used database solution for storing, managing, and processing data across diverse environments.

Check out the official PostgreSQL site here: https://www.postgresql.org/

CONNECTING TO A POSTGRESQL SERVER

Connect to a PostgreSQL server using the PostgreSQL command-line client (psql) and a username. It will prompt you for the password:
```
psql -U username
```

To connect to a specific database on a PostgreSQL server with a username:
```
psql -U username -h host_name -d
database_name
```

To exit the client:
```
\q
```

For a full list of commands:
```
\h
```

For a list of `psql` commands:
```
\?
```

To export data using the pg_dump tool:
```
pg_dump -U username -h host_name
  -d database_name > data_backup.sql
```

CREATING AND DISPLAYING DATABASES

To create a database:
```
CREATE DATABASE zoo;
```

To delete a specific database:
```
DROP DATABASE zoo;
```

To list all the databases on a server:
```
\l;
```

To connect to a specific database:
```
\c zoo;
```

To list all tables in a database:
```
\dt;
```

To get information about a specific table:
```
\d animal;
```
It outputs column names, data types, default values, and more about the table.

CREATING TABLES

To create a table:
```
CREATE TABLE habitat (
  id INT,
  name VARCHAR(64)
);
```

To increment the ID automatically with each new record, use the `SERIAL` data type:
```
CREATE TABLE habitat (
  id INT SERIAL PRIMARY KEY,
  name VARCHAR(64)
);
```

To create a table with a foreign key:
```
CREATE TABLE animal (
  id SERIAL PRIMARY KEY,
  name VARCHAR(64),
  species VARCHAR(64),
  age INT,
  habitat_id INT,
  FOREIGN KEY (habitat_id)
      REFERENCES habitat(id)
);
```

MODIFYING TABLES

Use the `ALTER TABLE` to modify a table structure.

To change a table name:
```
ALTER TABLE animal RENAME TO pet;
```

To add a column to the table:
```
ALTER TABLE animal
ADD COLUMN name VARCHAR(64);
```

To change a column name:
```
ALTER TABLE animal
RENAME COLUMN id TO identifier;
```

To change a column data type:
```
ALTER TABLE animal
ALTER COLUMN name TYPE VARCHAR(128);
```

To delete a column:
```
ALTER TABLE animal
DROP COLUMN name;
```

To delete a table:
```
DROP TABLE animal;
```

QUERYING DATA

To select data from a table, use `SELECT`.
An example of a single-table query:
```
SELECT species, AVG(age) AS average_age
FROM animal
WHERE id != 3
GROUP BY species
HAVING AVG(age) > 3
ORDER BY AVG(age) DESC;
```

An example of a multiple-table query:
```
SELECT city.name, country.name
FROM city
[INNER | LEFT | RIGHT | FULL] JOIN country
  ON city.country_id = country.id;
```

AGGREGATION AND GROUPING

- **`AVG(expr)`** – average value of `expr` for the group.
- **`COUNT(expr)`** – count of `expr` values within the group.
- **`MAX(expr)`** – maximum value of `expr` values within the group.
- **`MIN(expr)`** – minimum value of `expr` values within the group.
- **`SUM(expr)`** – sum of `expr` values within the group.

To count the rows in the table:
```
SELECT COUNT(*)
FROM animal;
```

To count the non-NULL values in a column:
```
SELECT COUNT(name)
FROM animal;
```

To count unique values in a column:
```
SELECT COUNT(DISTINCT name)
FROM animal;
```

GROUP BY

To count the animals by species:
```
SELECT species, COUNT(id)
FROM animal
GROUP BY species;
```

To get the average, minimum, and maximum ages by habitat:
```
SELECT habitat_id, AVG(age),
       MIN(age), MAX(age)
FROM animal
GROUP BY habitat_id;
```

INSERTING DATA

To insert data into a table, use `INSERT`:
```
INSERT INTO habitat VALUES
(1, 'River'),
(2, 'Forest');
```

You may specify the columns in which the data is added. The remaining columns are filled with default values or `NULL`s.
```
INSERT INTO habitat (name)
VALUES ('Savanna');
```

UPDATING DATA

To update the data in a table, use `UPDATE`:
```
UPDATE animal
SET
  species = 'Duck',
  name = 'Quack'
WHERE id = 2;
```

DELETING DATA

To delete data from a table, use `DELETE`:
```
DELETE FROM animal
WHERE id = 1;
```
This deletes all rows satisfying the `WHERE` condition.

To delete all data from a table, use `TRUNCATE TABLE`:
```
TRUNCATE TABLE animal;
```

COPYING DATA

To import data from a CSV file into a table:
```
\copy animal FROM 'animal.csv' CSV HEADER
```

To export data from a query to a CSV file:
```
\copy (SELECT * FROM animal)
  TO 'animal.csv' CSV HEADER
```

CASTING

To change the type of a value, use the `::` operator:
```
SELECT 25.5::INTEGER; -- result: 26
```

You may also use `CAST()`. This is useful when the name of the type contains spaces, e.g., `double precision`:
```
SELECT CAST(column AS DOUBLE PRECISION);
```

TEXT FUNCTIONS

FILTERING THE OUTPUT

To fetch the city names that are not Berlin:
```
SELECT name
FROM city
WHERE name != 'Berlin';
```

TEXT OPERATORS

To fetch the city names that start with a `'P'`:
```
SELECT name
FROM city
WHERE name LIKE 'P%';
```

To fetch the city names that start with any letter followed by `'ublin'` (like Dublin in Ireland or Lublin in Poland):
```
SELECT name
FROM city
WHERE name LIKE '_ublin';
```

CONCATENATION

To concatenate two strings, use the `||` operator or the `CONCAT()` function:
```
SELECT 'Hi ' || 'there!';
-- result: Hi there!
SELECT CONCAT('Hello ', 'there!');
-- result: Hello there!
```

Note that with `||`, the result is `NULL` if any of the strings is `NULL`:
```
SELECT 'Great ' || 'day' || NULL;
-- result: NULL
```

In contrast, `CONCAT()` ignores `NULL`:
```
SELECT CONCAT('Good ', 'day', NULL);
-- result: Good day
```

OTHER USEFUL TEXT FUNCTIONS

To get the count of characters in a string:
```
SELECT LENGTH('LearnSQL.com');
-- result: 12
```

To convert all letters to lowercase:
```
SELECT LOWER('LEARNSQL.COM');
-- result: learnsql.com
```

To convert all letters to uppercase:
```
SELECT UPPER('LearnSQL.com');
-- result: LEARNSQL.COM
```

To capitalize the first letter of each word in a string, use `INITCAP()`:
```
SELECT INITCAP('hello world');
-- result: 'Hello World'
```

To get a part of a string:
```
SELECT SUBSTRING('LearnSQL.com', 9);
-- result: .com
SELECT SUBSTRING('LearnSQL.com', 1, 5);
-- result: Learn
```

To replace a part of a string:
```
SELECT REPLACE('LearnSQL.com', 'SQL',
'Python'); -- result: LearnPython.com
```

NUMERIC FUNCTIONS

Use `+`, `-`, `*`, `/` for basic math.

To get the number of seconds in a week:
```
SELECT 60 * 60 * 24 * 7; -- result: 604800
```

In PostgreSQL, the division operator `/` performs an integer division on integer arguments. For example:
```
SELECT 25 / 4; -- result 6
```
Avoid integer division by including at least one non-integer argument:
```
SELECT 25::numeric / 4; -- result 6.25
SELECT 25.0 / 4; -- result 6.25
```

To get the remainder of a division:
```
SELECT MOD(13, 2); -- result: 1
SELECT 13 % 2; -- result: 1
```

To round a number to its nearest integer:
```
SELECT ROUND(1234.56789); -- result: 1235
```

To round a number to three decimal places (`NUMERIC` arguments only):
```
SELECT ROUND(1234.56789, 3);
-- result: 1234.568
```

To get the absolute value of a number:
```
SELECT ABS(-12); -- result: 12
```

To get the square root of a number:
```
SELECT SQRT(9); -- result: 3
```

USEFUL NULL FUNCTIONS

To fetch the names of the cities whose rating values are not missing:
```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

COALESCE(x, y, ...)

To replace `NULL` in a query with something meaningful:
```
SELECT domain,
       COALESCE(domain, 'domain missing')
FROM contacts;
```
`COALESCE()` takes any number of arguments and returns the value of the first non-`NULL` argument.

NULLIF(x, y)

To save yourself from *division by 0* errors:
```
SELECT last_month, this_month,
  this_month * 100.0
    / NULLIF(last_month, 0)
    AS better_by_percent
FROM video_views;
```
`NULLIF(x, y)` returns `NULL` if x equals y; else it returns the value of x.

DATE AND TIME

There are 5 main time-related types in PostgreSQL:

`DATE` – a date with a resolution of one day; stores the year, month, and day in the `YYYY-MM-DD` format.

`TIME` – a time of day with a resolution of one microsecond; stores the hours, minutes, seconds, and fractional seconds in the `HH:MM:SS.SSSSSS` format.

`TIMESTAMP WITH TIME ZONE` – a timestamp with the time zone; stores the date and the time along with the corresponding time zone information. The range is from `'4713-11-24 00:00:00'` BC to `'294276-12-31 23:59:59'` AD.

`TIMESTAMP` – a timestamp without the time zone; stores the date and the time. PostgreSQL handles `TIMESTAMP` values automatically with time zone conversion.

`INTERVAL` – a duration of time, such as 3 days, 4 hours, and 30 minutes.

WHAT TIME IS IT?

To answer this question, use:
- `CURRENT_TIME` – to get the current time.
- `CURRENT_DATE` – to get the current date.
- `CURRENT_TIMESTAMP` – to get the current timestamp with both of the above.

CREATING DATE/TIME VALUES

To create a date, time, or datetime value, write it as a string and cast it to the desired type.
```
SELECT '2023-12-31'::date;
SELECT '15:31'::time;
SELECT '2023-12-31 23:59:29'::timestamp;
```
You may also use `CAST()` or `DATE()`.

You may skip casting in simple conditions. The database knows what you mean.
```
SELECT airline, flight_number,
departure_time
FROM airport_schedule
WHERE departure_time < '12:00';
```

INTERVALS

An interval is the duration between two points in time.
To define an interval: `INTERVAL '3 days';`

This syntax consists of the `INTERVAL` keyword, a value, and a time part keyword (`YEAR`, `QUARTER`, `MONTH`, `WEEK`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `MICROSECOND`).
You may combine different `INTERVAL`s using the + or – operator:
```
INTERVAL '1 year' + INTERVAL '3 months'
```

EXTRACTING PARTS OF DATES

To extract a part of a date, use `EXTRACT()`:
```
SELECT
  EXTRACT(MONTH FROM '2023-12-31'::DATE);
-- result: 12
```

You may also use `DATE_PART()`. It extracts specific components from a date or timestamp.
```
SELECT DATE_PART('day', '2023-12-
31'::DATE); -- result: 31
```
Common arguments include `'day'`, `'month'`, `'year'`, `'quarter'`, `'hour'`, `'minute'`, and `'second'`, among others.

DATE ARITHMETICS

To add or subtract an `INTERVAL` from a `date`, `time`, or `timestamp`:
```
SELECT '2023-10-31'::DATE
       + INTERVAL '2 months';
-- result: '2023-12-31'
SELECT '2024-04-05'::DATE
       + INTERVAL '-3 days';
-- result: '2024-04-02'
SELECT '2023-06-10 07:55:00'::TIMESTAMP
       + INTERVAL '2 months';
-- result: '2023-08-10 07:55:00'
SELECT '2023-02-12 10:20:24'::TIMESTAMP
       + INTERVAL '-12:43:02';
-- result: '2023-02-11 21:37:22'
```

To find the difference between two dates in days:
```
SELECT '2024-01-01'::date
       - '2023-01-02'::date AS date_diff;
-- result: 364
```

`DATE_TRUNC()` in PostgreSQL truncates date or timestamp values to the specified time units.
```
SELECT DATE_TRUNC('hour',
       '2023-01-15 14:38:00'::TIMESTAMP);
-- result: '2023-01-15 14:00'
SELECT DATE_TRUNC('month',
       '2023-12-30'::DATE);
-- result: '2023-12-01'
```

`DATE_TRUNC()` is often used to group by year, month, week, etc.
```
SELECT
  DATE_TRUNC('month', birth_date) AS
month,
  COUNT(*)
FROM animal
GROUP BY DATE_TRUNC('month', birth_date)
ORDER BY DATE_TRUNC('month', birth_date);
```

Check out the **SQL from A to Z in PostgreSQL** learning track or explore all courses at **LearnSQL.com**.