PostgreSQL Tutorial

Welcome to the PostgreSQLTutorial.com website! This**PostgreSQL tutorial** helps you understand PostgreSQL quickly. You will learn PostgreSQL fast through many practical examples. We will show you not only problems but also how to solve them creatively in PostgreSQL.

If you are…

- Looking for learning PostgreSQL fast and easily.
- Developing applications using PostgreSQL as the back-end database management system.
- Migrating from other database management systems such as MySQL, Oracle, Microsoft SQL Server to PostgreSQL.

You will find all you need to know to get started with the PostgreSQL quickly and effectively here on our website.

We developed the PostgreSQL tutorial to demonstrate the unique features of PostgreSQL that make it the most advanced open source database management system.

Basic PostgreSQL Tutorial

First, you will learn how to query data from a single table using basic data selection techniques such as selecting columns, sorting result set, and filtering rows. Then, you will learn about the advanced queries such as joining multiple tables, using set operations, and constructing the subquery. Finally, you will learn how to manage database tables such as creating new a table or modifying an existing table's structure.

Section 1. Getting started with PostgreSQL

If you are new to PostgreSQL, follow 3-easy steps to get started quickly with PostgreSQL.

- First, get a brief overview of PostgreSQL to understand what PostgreSQL is.

Second, install PostgreSQL to your local computer and connect to PostgreSQL database serverfrom a client application such as psql or pgAdmin.
Third, download the PostgreSQL sample database and load it into the PostgreSQL database server.

Section 2. Querying Data

Select – shows you how to query data from a single table.
Order By – guides you how to sort the result set returned from a query.
Select Distinct – provides you a clause that removes duplicate rows in the result set.

Section 3. Filtering data

Where – filters rows based on a specified condition.
Limit – gets a subset of rows generated by a query.
Fetch– limits the number of rows returned by a query.
In – selects data that matches any value in a list of values.
Between – selects data that is a range of values.
Like – filters data based on pattern matching.

Section 4. Joining multiple tables

Inner Join – selects rows from one table that have the corresponding rows in other tables.
Left Join – selects rows from one table that may or may not have the corresponding rows in other tables.
Self-join – joins a table to itself by comparing a table to itself.
Full Outer Join – uses the full join to find a row in a table that does not have a matching row in another table.
Cross Join – produces a Cartesian product of the rows in two or more tables.
Natural Join – joins two or more tables using implicit join condition based on the common column names in the joined tables.

Section 5. Grouping data

Group By – divides rows into groups and applies an aggregate function on each.
Having – applies the condition for groups.

Section 6. Performing set operations

Union – combines result sets of multiple queries into a single result set.
Intersect – combines the result sets of two or more queries and returns a single result set that has the rows appear in both result sets.
Except – returns the rows in the first query that does not appear in the output of the second query.

Section 7. Grouping Sets

Grouping Sets – generate multiple grouping sets in reporting.
Cube – define multiple grouping sets that include all possible combinations of dimensions.
Rollup – generate reports that contain totals an subtotals.

Section 8. Subquery

Subquery – writes a query nested inside another query.
ANY – retrieves data by comparing a value with a set of values returned by a subquery.
ALL – query data by comparing a value with a list of values returned by a subquery.
EXISTS – checks for the existence of rows returned by a subquery.

Section 9. Modifying data

In this section, you will learn how to insert data into a table with the INSERT statement, modify existing data with the UPDATE statement, and remove data with the DELETE statement. In addition, you learn how to use the upsert statement to merge data.

Insert – inserts data into a table.
Update – updates existing data in a table.
Update join – updates values in a table based on values in another table.
Delete – deletes data in a table.
Upsert – inserts or update data if the new row already exists in the table.

Section 10. PostgreSQL import & export

You will learn how to import and export PostgreSQL data from and to CSV file format using copy command.

Import CSV file into Table – shows you how to import CSV file into a table.

Export PostgreSQL Table to CSV file – shows you how to export tables to a CSV file.

Section 11. Managing tables

In this section, we start exploring the PostgreSQL data types and showing you how to use the CREATE TABLE statement to create a new table. We will also cover some additional features, such as modifying table structure and deleting tables. In addition, you will learn an efficient way to delete all rows from a table by using the TRUNCATE statement.

Data types – covers the most commonly used PostgreSQL data types.
Create table – guides you how to create a new table in the database.
Select Into & Create table as– shows you how to create a new table from the result set of a query.
Auto-increment column with SERIAL – uses SERIAL to add an auto-increment column to a table.
Alter table – changes structure of an existing table.
Rename table – change the name of the table to a new one.
Rename database – change the name of the database to a new one.
Add column – shows you how to use add one or more columns to an existing table.
Drop column – demonstrates how to drop a column of a table.
Change column data type – shows you how to change the data of a column.
Rename column – illustrates how to rename one or more column of a table.
Drop table – removes an existing table and all of its dependent objects.
Temporary table – shows you how to use the temporary table.
Truncate table – removes all data in a large table quickly and efficiently.

Section 12. PostgreSQL data types in depth

Boolean – stores TRUE and FALSE values with the Boolean data type.
CHAR, VARCHAR and TEXT – learns how to use various character types including CHAR, VARCHAR, and TEXT.
NUMERIC – shows you how to use NUMERIC type to store values that precision is required.
Integer – introduces you various integer types in PostgreSQL including SMALLINT, INT and BIGINT.
SERIAL – shows you how to create an auto-increment column using SERIAL pseudotype.
DATE – introduces the DATE data type for storing date values.
Timestamp – understands timestamp data types quickly.
Interval – shows you how to use interval data type to handle a period of time effectively.
TIME – uses the TIME data type to manage time of day values.

UUID – guides you to use UUID data type and how to generate UUID values using supplied modules.
Array – shows you how to work with the array and introduces you to some handy functions for array manipulation.
hstore – introduces you to hstore data type which is a set of key/value pairs stored in a single value in PostgreSQL.
JSON – illustrates how to work with JSON data type and shows you how to use some of the most important JSON operators and functions.
User-defined data types – shows you how to use the CREATE DOMAIN and CREATE TYPE statements to create user-defined data types.

Section 13. Understanding PostgreSQL constraints

Primary key – illustrates how to define primary key when creating a table or add primary keys to existing tables.
Foreign key – shows you how to define foreign key constraints when creating a new table or add foreign key constraints for existing tables.
CHECK constraint – adds logic to check value based on a Boolean expression.
UNIQUE constraint – makes sure that a value in a column or a group of columns unique across the table.
NOT NULL constraint – ensures values in a column are not NULL.

Section 14. Conditional expressions & operators

CASE – shows you how to form conditional queries with the CASE expression.
COALESCE – returns the first non-null argument. You can use it to substitute NULL by a default value.
NULLIF – returns NULL if the first argument equals the second one.
CAST – converts from one data type into another e.g., from a string into an integer, from a string into a date.

Section 15. PostgreSQL utilities

psql commands – shows you the most common psql commands that help you interact with psql faster and more effectively.

Section 16. PostgreSQL recipes

How to delete duplicate rows in PostgreSQL – shows you various ways to delete duplicate rows from a table.
How to copy a table – shows you how to copy a table to a new one.
How to copy a database – learns how to copy a database on the same server or from a server to another.

How to generate a random number in a range – illustrates how to generate a random number in a specific range.
How to use PostgreSQL recursive query – discusses the recursive query and learns how to apply it in various contexts.
How to use PostgreSQL Window Functions – introduces you to the most commonly used PostgreSQL window functions

---

```
create table private.db_emp(
emp_id SERIAL PRIMARY KEY,
emp_username varchar(50) UNIQUE NOT NULL,
emp_name varchar(5) NOT NULL,
emp_number integer,
emp_address varchar(100),
emp_salary integer
)

select * from private.db_emp;

insert into private.db_emp
(emp_username,emp_name,emp_number,emp_address,emp_salary) values
('srra6001','Nick','12345','usa',8000);
insert into private.db_emp
(emp_username,emp_name,emp_number,emp_address,emp_salary) values
('frra6001','Nick','12348','usa',6000);
insert into private.db_emp
(emp_username,emp_name,emp_number,emp_address,emp_salary) values
('brra6001','Nick','12345','pakistan',7000);
insert into private.db_emp
(emp_username,emp_name,emp_number,emp_address,emp_salary) values
('zrra6001','Nick','52345','india',2000);
insert into private.db_emp
(emp_username,emp_name,emp_number,emp_address,emp_salary) values
('qrra6001','Nick','62345','uk',1000)

select * from private.db_emp where emp_salary >=5000;

select * from private.db_emp where emp_salary >=5000 and emp_address = 'usa';

ALTER table private.db_emp add column emp_gender varchar;

ALTER table private.db_emp RENAME COLUMN emp_number to emp_mob_number;

select * from private.db_emp;
```

```
ALTER TABLE private.db_emp ALTER COLUMN emp_mob_number SET NOT NULL;

ALTER table private.db_emp add column emp_details jsonb;

INSERT INTO private.db_emp
(emp_username,emp_name,emp_mob_number,emp_address,emp_salary,emp_gender,em
p_details) values ('oora6001','Todd','98345','canada',5000,'male','{"name": "Cook dinner",
"tags": ["Cook", "Kitchen", "Bed"], "ingredients": ["Chesse", "Guacamole"], "finished": true}');

SELECT emp_details->>'name' FROM private.db_emp;
```

?column? text
[null]
[null]
[null]
[null]
[null]
Cook lunch
Cook dinner

```
SELECT emp_details->>'tags' FROM private.db_emp;
```

?column? text
[null]
[null]
[null]
[null]
[null]
["Cook", "Ki...
["Cook", "Ki...

```
SELECT * from private.db_emp where emp_details->>'finished' = 'true';
```

emp_id integer	emp_username character varying (50)	emp_name character varying (5)	emp_mob_number integer	emp_address character varying (100)	emp_salary integer	emp_gender character varying	emp_details jsonb
7	oora6001	Todd	98345	canada	5000	male	{"name":"Coo...

SELECT emp_details->>'tags' AS tags FROM private.db_emp;

tags text
[null]
[null]
[null]
[null]
[null]
["Co...
["Co...

SELECT count(*) from private.db_emp where emp_details->>'finished' = 'true';

count bigint
1

-- First way to query to json data
SELECT * FROM private.db_emp WHERE emp_details @> '{"name": "Cook lunch"}';

emp_id integer	emp_username character varying (50)	emp_name character varying (5)	emp_mob_number integer	emp_address character varying (100)	emp_salary integer	emp_gender character varying	emp_details jsonb
6	prra6001	Mark	88345	canada	9000	male	{"name":"Coo...

-- Second way to query to json data
SELECT * FROM private.db_emp WHERE emp_details ->> 'name' = 'Cook lunch';

emp_id integer	emp_username character varying (50)	emp_name character varying (5)	emp_mob_number integer	emp_address character varying (100)	emp_salary integer	emp_gender character varying	emp_details jsonb
6	prra6001	Mark	88345	canada	9000	male	{"name":"Coo...

-- Creating new table with a Foreign Key with emp_test table

```
create table private.db_emp_personal(
per_id SERIAL PRIMARY KEY,
per_name varchar(5) NOT NULL,
```

```
per_number integer,
emp_address varchar(100),
FOREIGN KEY (per_id) REFERENCES private.db_emp (emp_id)
)

SELECT * FROM private.db_emp_personal;

INSERT INTO private.db_emp_personal (per_name,per_number,emp_address) VALUES
('Nick',12345,'USA');
INSERT INTO private.db_emp_personal (per_name,per_number,emp_address) VALUES
('Sam',72345,'india');
INSERT INTO private.db_emp_personal (per_name,per_number,emp_address) VALUES
('Mark',62345,'canada');
INSERT INTO private.db_emp_personal (per_name,per_number,emp_address) VALUES
('Todd',22345,'spain');

SELECT * FROM private.db_emp_personal;
```

per_id integer	per_name character varying (5)	per_number integer	emp_address character varying (100)
1	Nick	12345	USA
2	Sam	72345	india
3	Mark	62345	canada
4	Todd	22345	spain

```
-- Inner Join – selects rows from one table that have the corresponding rows in other tables.
(the colums which is required in both the tables we can use it)
-- Left Join – selects rows from one table that may or may not have the corresponding rows
in other tables.
-- Self-join – joins a table to itself by comparing a table to itself.
-- Full Outer Join – uses the full join to find a row in a table that does not have a matching
row in another table.
-- Cross Join – produces a Cartesian product of the rows in two or more tables.
-- Natural Join – joins two or more tables using implicit join condition based on the common
column names in the joined tables.(their is a disadvantage of it)

-- Inner Join Query

SELECT private.db_emp.emp_id, private.db_emp.emp_username,
private.db_emp_personal.per_name, private.db_emp_personal.per_number FROM
private.db_emp INNER JOIN private.db_emp_personal ON private.db_emp.emp_id =
private.db_emp_personal.per_id;
```

emp_id integer	emp_username character varying (50)	per_name character varying (5)	per_number integer
1	srra6001	Nick	12345
2	frra6001	Sam	72345
3	brra6001	Mark	62345
4	zrra6001	Todd	22345

```
-- PostgreSQL NATURAL JOIN Explained By Examples

CREATE TABLE categories (
 category_id serial PRIMARY KEY,
 category_name VARCHAR (255) NOT NULL
);

CREATE TABLE products (
 product_id serial PRIMARY KEY,
 product_name VARCHAR (255) NOT NULL,
 category_id INT NOT NULL,
 FOREIGN KEY (category_id) REFERENCES categories (category_id)
);

INSERT INTO categories (category_name)
VALUES
 ('Smart Phone'),
 ('Laptop'),
 ('Tablet');

INSERT INTO products (product_name, category_id)
VALUES
 ('iPhone', 1),
 ('Samsung Galaxy', 1),
 ('HP Elite', 2),
 ('Lenovo Thinkpad', 2),
 ('iPad', 3),
 ('Kindle Fire', 3);

SELECT * FROM products NATURAL JOIN categories;
```

category_id integer	product_id integer	product_name character varying (255)	category_name character varying (255)
1	1	iPhone	Smart Phone
1	2	Samsung Galaxy	Smart Phone
2	3	HP Elite	Laptop
2	4	Lenovo Thinkpad	Laptop
3	5	iPad	Tablet
3	6	Kindle Fire	Tablet

```
-- Disadvantage of PostgreSQL NATURAL JOIN Explained By Examples
-- if you have two or more than two colums common, then the select result will be always
null

CREATE TABLE city (
 category_id serial PRIMARY KEY,
 last_update VARCHAR (255) NOT NULL
);

CREATE TABLE country (
 product_id serial PRIMARY KEY,
 last_update VARCHAR (255) NOT NULL,
 category_id INT NOT NULL,
 FOREIGN KEY (category_id) REFERENCES categories (category_id)
);

INSERT INTO city (last_update)
VALUES
 ('Smart Phone'),
 ('Laptop'),
 ('Tablet');

INSERT INTO country (last_update, category_id)
VALUES
 ('iPhone', 1),
 ('Samsung Galaxy', 1),
 ('HP Elite', 2),
 ('Lenovo Thinkpad', 2),
 ('iPad', 3),
 ('Kindle Fire', 3);

SELECT * FROM city NATURAL JOIN country;
```

	category_id integer	last_update character varying (255)	product_id integer

---

Part 2

Postgresql tutorial

- [root@localhost pgsql]# sudo -u postgres psql => login to postgres with 'postgres' user
- postgres=# create database studentdb; => to create a database with name 'studentdb'
- postgres=# \l => listing the database names

```
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_IN.UTF-8 | en_IN.UTF-8 |
 studentdb | postgres | UTF8     | en_IN.UTF-8 | en_IN.UTF-8 |
 template0 | postgres | UTF8     | en_IN.UTF-8 | en_IN.UTF-8 | =c/postgres        +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_IN.UTF-8 | en_IN.UTF-8 | =c/postgres        +
           |          |          |             |             | postgres=CT
```

- postgres=# \c studentdb; => connect to 'studentdb' database

You are now connected to database "studentdb" as user "postgres".

- studentdb=# create schema studentdbschema; => create schema with name 'studentdbschema'

CREATE SCHEMA

- studentdb=# create table studentdbschema.studenttable_1 (id integer, password char); => create table 'studenttable_1'in schema 'studentdbschema'

CREATE TABLE

studentdb=# select * from pg_catalog.pg_tables where schemaname != 'information_schema'and schemaname != 'pg_catalog';

```
schemaname    |   tablename    | tableowner | tablespace | hasindexes | hasrules | hastriggers
----------------+----------------+------------+------------+------------+----------+----------
---
 studentdbschema | studenttable_1 | postgres   |            | f          | f        | f
(1 row)
```

- studentdb=# insert into studentdbschema.studenttable_1 values (1,'1');

```
INSERT 0 1

studentdb=# insert into studentdbschema.studenttable_1 values (2,'2');

INSERT 0 1

studentdb=# select * from studentdbschema.studenttable_1;

 id | password
----+----------
  1 | 1
  2 | 2
(2 rows)

- studentdb=# truncate studentdbschema.studenttable_1; => delete the records in the table
```

TRUNCATE TABLE

```
studentdb=# select * from studentdbschema.studenttable_1;

 id | password
----+----------
(0 rows)

- studentdb=# insert into studentdbschema.studenttable_1 values (1,'1');
INSERT 0 1
studentdb=# insert into studentdbschema.studenttable_1 values (2,'2');
INSERT 0 1
studentdb=# select * from studentdbschema.studenttable_1;
 id | password
----+----------
  1 | 1
  2 | 2
(2 rows)

studentdb=# ^C
studentdb=# truncate studentdbschema.studenttable_1;
TRUNCATE TABLE
studentdb=# select * from studentdbschema.studenttable_1;
 id | password
----+----------
(0 rows)

studentdb=# ^C
studentdb=# insert into studentdbschema.studenttable_1 values (1,'1');
INSERT 0 1
studentdb=# insert into studentdbschema.studenttable_1 values (2,'2');
INSERT 0 1
studentdb=# select * from studentdbschema.studenttable_1;
```

```
 id | password
----+----------
  1 | 1
  2 | 2
(2 rows)

studentdb=# delete from studentdbschema.studenttable_1;
DELETE 2
studentdb=# select * from studentdbschema.studenttable_1;
 id | password
----+----------
(0 rows)
```

- studentdb=# drop table studentdbschema.studenttable_1;

DROP TABLE

- \d => to describe the databaseLogical Operators

- postgres=# \du => listing users
- postgres=# create user devops with password 'redhat'; => create new user with a password
- postgres=# alter user devops with superuser; => add superuser to devops user
- postgres=# alter user postgres with password 'redhat'; => to change password of a user
- postgres=# drop user devops; => delete the user

- studentdb=# select * from studentdbschema.emp;

```
 id | name  | age | address | salary
----+-------+-----+---------+--------
  1 | ram   |  20 | delhi   |  10000
  2 | shyam |  25 | chennai |  20000
  3 | rocky |  30 | kolkata |  30000
  4 | happy |  35 | mumbai  |  40000
  5 | mark  |  40 | newyork |  50000
(5 rows)

studentdb=# select * from studentdbschema.emp where salary <= 10000;
 id | name | age | address | salary
----+------+-----+---------+--------
  1 | ram  |  20 | delhi   |  10000
(1 row)

studentdb=# select * from studentdbschema.emp where salary > 10000;
 id | name  | age | address | salary
----+-------+-----+---------+--------
  2 | shyam |  25 | chennai |  20000
  3 | rocky |  30 | kolkata |  30000
  4 | happy |  35 | mumbai  |  40000
  5 | mark  |  40 | newyork |  50000
(4 rows)
```

- Logical Operators

```
studentdb=# select * from studentdbschema.emp where salary > 10000 and age > 30;
 id | name  | age | address | salary
----+-------+-----+---------+--------
  4 | happy |  35 | mumbai  |  40000
  5 | mark  |  40 | newyork |  50000
(2 rows)

studentdb=# select * from studentdbschema.emp where salary > 10000 or age > 30;
 id | name  | age | address | salary
----+-------+-----+---------+--------
  2 | shyam |  25 | chennai |  20000
  3 | rocky |  30 | kolkata |  30000
  4 | happy |  35 | mumbai  |  40000
  5 | mark  |  40 | newyork |  50000
(4 rows)

studentdb=# select * from studentdbschema.emp where salary is null;
 id | name | age | address | salary
----+------+-----+---------+--------
  6 | nick |  45 | newyork |
(1 row)

studentdb=# select * from studentdbschema.emp where name like 'r%';
 id | name  | age | address | salary
----+-------+-----+---------+--------
  1 | ram   |  20 | delhi   |  10000
  3 | rocky |  30 | kolkata |  30000
(2 rows)

studentdb=# select * from studentdbschema.emp where name like '%m';
 id | name  | age | address | salary
----+-------+-----+---------+--------
  1 | ram   |  20 | delhi   |  10000
  2 | shyam |  25 | chennai |  20000
(2 rows)

studentdb=# select * from studentdbschema.emp where name like 's%m';
 id | name  | age | address | salary
----+-------+-----+---------+--------
  2 | shyam |  25 | chennai |  20000
(1 row)

studentdb=# select * from studentdbschema.emp where age in ('25','20');
 id | name  | age | address | salary
----+-------+-----+---------+--------
  1 | ram   |  20 | delhi   |  10000
  2 | shyam |  25 | chennai |  20000
(2 rows)
```

```
studentdb=# select * from studentdbschema.emp where age between 25 and 35;
 id | name  | age | address | salary
----+-------+-----+---------+--------
  2 | shyam |  25 | chennai |  20000
  3 | rocky |  30 | kolkata |  30000
  4 | happy |  35 | mumbai  |  40000
(3 rows)
```