

#PostgreSQL#



Section 12. Managing Tables

In this section, you will begin exploring the PostgreSQL data types and learn how to create new tables and modify the structure of the existing ones.

Data types – cover the most commonly used PostgreSQL data types.

Create a table – guide you on 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.

Sequences – introduce you to sequences and describe how to use a sequence to generate a sequence of numbers.

Identity column – show you how to use the identity column.

Alter table – modify the structure of an existing table.

Rename table – change the name of the table to a new one.

Add column – show you how to add one or more columns to an existing table.

Drop column – demonstrate how to drop a column of a table.

Change column data type – show you how to change a column's data.

Rename column – illustrate how to rename one or more table columns.

Drop table – remove an existing table and all of its dependent objects.

Truncate table – remove all data in a large table quickly and efficiently.

Temporary table – show you how to use the temporary table.

Copy a table – show you how to copy a table to a new one.

Can You Teach Me above all the topics with Realtime example (create a simple table and insert some values) and show output in Table Format and Also Explain Each topic why this needed, what happen, like syntax and so on.



Let's go through these topics step by step with examples, syntax, and explanations.

1. PostgreSQL Data Types

Why needed: To define the type of data each column can store, ensuring consistency and optimizing storage.

Example:

```
sql
```

```
CREATE TABLE data_types_demo (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  balance NUMERIC(10, 2),
```

```
);
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

Explanation:

- `SERIAL`: Auto-incrementing integer.
- `VARCHAR(50)`: Variable-length character string, max 50 chars.
- `INT`: Integer data type.
- `NUMERIC(10, 2)`: Decimal number with up to 10 digits, 2 after the decimal.
- `TIMESTAMP`: Stores date and time.

Inserting values:

```
sql

INSERT INTO data_types_demo (name, age, balance)
VALUES ('Alice', 30, 5000.75), ('Bob', 40, 3000.50);
```

Query:

```
sql

SELECT * FROM data_types_demo;
```

Output:

id	name	age	balance	created_at
1	Alice	30	5000.75	2024-12-23 12:00:00
2	Bob	40	3000.50	2024-12-23 12:01:00

2. Creating a Table

Why needed: To organize and store data in a structured format.

Syntax:

```
sql

CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

Example:

```
sql

CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
```

```
emp_name VARCHAR(100),
department VARCHAR(50),
salary NUMERIC(12, 2)
);
```

3. SELECT INTO & CREATE TABLE AS

Why needed: To create a new table based on the result of a query.

Example using SELECT INTO:

```
sql

SELECT * INTO employees_backup FROM employees;
```

Example using CREATE TABLE AS:

```
sql

CREATE TABLE high_salary_employees AS
SELECT * FROM employees WHERE salary > 50000;
```

Would you like to proceed with the remaining topics, or focus more on these initial concepts?



Above all the topics with example Give me ?



Let's cover all the topics with examples, syntax, detailed explanations, and outputs in table format.

1. PostgreSQL Data Types

Why needed: Define the type of data each column stores, ensuring consistency and avoiding invalid data entries.

Example:

```
sql

CREATE TABLE data_types_demo (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  balance NUMERIC(10, 2),
  joined_on DATE DEFAULT CURRENT_DATE
);
```

Explanation:

- `SERIAL`: Auto-increment integer.
- `VARCHAR(50)`: Variable-length string, max 50 chars.
- `INT`: Integer data type.
- `NUMERIC(10, 2)`: Decimal number (10 digits, 2 after decimal).
- `DATE`: Stores dates.

Insert values:

```
sql

INSERT INTO data_types_demo (name, age, balance)
VALUES
('Alice', 30, 5000.75),
('Bob', 40, 3000.50);
```

Query:

```
sql

SELECT * FROM data_types_demo;
```

Output:

id	name	age	balance	joined_on
1	Alice	30	5000.75	2024-12-23
2	Bob	40	3000.50	2024-12-23

2. Creating a Table

Why needed: To define a structure for storing data.

Syntax:

```
sql

CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

Example:

```
sql

CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(100),
    department VARCHAR(50),
```

```
); salary NUMERIC(12, 2)
```

3. SELECT INTO & CREATE TABLE AS

Why needed: To create new tables from existing data.

Example using `SELECT INTO`:

```
sql

SELECT * INTO employees_backup FROM employees;
```

Example using `CREATE TABLE AS`:

```
sql

CREATE TABLE high_salary_employees AS
SELECT * FROM employees WHERE salary > 50000;
```

4. Auto-increment Column with SERIAL

Why needed: Automatically generate unique numbers for rows.

Example:

```
sql

CREATE TABLE departments (
    dept_id SERIAL PRIMARY KEY,
    dept_name VARCHAR(50)
);

INSERT INTO departments (dept_name) VALUES ('HR'), ('IT'), ('Finance');

SELECT * FROM departments;
```

Output:

dept_id	dept_name
1	HR
2	IT
3	Finance

5. Sequences

Why needed: Generate unique numbers that can be used in multiple tables.

Example:

sql

```
CREATE SEQUENCE emp_id_seq START 100;

CREATE TABLE sequence_demo (
  emp_id INT DEFAULT nextval('emp_id_seq'),
  emp_name VARCHAR(50)
);

INSERT INTO sequence_demo (emp_name) VALUES ('John'), ('Jane');

SELECT * FROM sequence_demo;
```

Output:

emp_id	emp_name
100	John
101	Jane

6. Identity Column

Why needed: A replacement for `SERIAL`, offering better control.

Example:

sql

```
CREATE TABLE identity_demo (
  id INT GENERATED ALWAYS AS IDENTITY,
  description TEXT
);

INSERT INTO identity_demo (description) VALUES ('Identity Example');

SELECT * FROM identity_demo;
```

7. Alter Table

Why needed: Modify the structure of an existing table.

Example:

sql

```
ALTER TABLE employees ADD COLUMN email VARCHAR(100);

ALTER TABLE employees DROP COLUMN department;

ALTER TABLE employees RENAME COLUMN salary TO monthly_salary;
```

8. Rename Table

Why needed: Rename an existing table for clarity or consistency.

Example:

```
sql

ALTER TABLE employees RENAME TO staff;
```

9. Add Column

Why needed: Add new fields to store additional information.

Example:

```
sql

ALTER TABLE staff ADD COLUMN hire_date DATE;
```

10. Drop Column

Why needed: Remove unused or irrelevant columns.

Example:

```
sql

ALTER TABLE staff DROP COLUMN hire_date;
```

11. Change Column Data Type

Why needed: Change the type of data stored in a column.

Example:

```
sql

ALTER TABLE staff ALTER COLUMN emp_name TYPE TEXT;
```

12. Rename Column

Why needed: Rename columns for better understanding.

Example:

sql

```
ALTER TABLE staff RENAME COLUMN emp_name TO full_name;
```

13. Drop Table

Why needed: Permanently delete a table.

Example:

sql

```
DROP TABLE staff;
```

14. Truncate Table

Why needed: Quickly remove all data while retaining the structure.

Example:

sql

```
TRUNCATE TABLE employees;
```

15. Temporary Table

Why needed: Store data temporarily for a session.

Example:

sql

```
CREATE TEMP TABLE temp_employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(50)
);

INSERT INTO temp_employees (emp_name) VALUES ('Temp User');
SELECT * FROM temp_employees;
```

16. Copy a Table

Why needed: Duplicate the structure and/or data of an existing table.

Example:

sql

```
CREATE TABLE employees_copy AS TABLE employees WITH DATA;
```

Would you like detailed outputs or additional focus on specific topics?