**Prompt : Give me a brief information about a create statement in Postgres SQL by using various datatypes with examples and give me a 2 example of create statement by using various Data Types.**

* **PostgreSQL CREATE TABLE Command**

The CREATE TABLE command is the primary tool you'll use to define the structure of your data. Think of it as creating a blueprint for a spreadsheet. You define the name of the sheet (the **table name**), what each column will be called (the **column name**), and what kind of information can go into each column (the **data type**).

This is a fundamental skill because a well-designed table is the foundation of a reliable and efficient database.

**Core Purpose**

* **Define Structure:** To create a new, empty table in your database.
* **Specify Columns:** To name each column that will hold a piece of data.
* **Enforce Data Types:** To declare what kind of data is allowed in each column (e.g., numbers, text, dates).
* **Set Rules (Constraints):** To apply rules that protect the integrity and consistency of your data.
* **General Syntax**

The basic structure of the command looks like this:

SQL

CREATE TABLE table\_name (  
 column\_1\_name data\_type [column\_constraints],  
 column\_2\_name data\_type [column\_constraints],  
 ...  
 [table\_constraints]  
);

Let's break down these components.

* **Key Components Explained**

**1. table\_name**

This is the unique name for your table within the schema. It's best practice to use a descriptive, plural name (e.g., users, products, orders). Table names are typically lowercase with underscores separating words (e.g., order\_details).

**2. column\_name**

This is the name of a specific column in your table (e.g., first\_name, price, order\_date). Choose clear and descriptive names.

**3. data\_type**

This is crucial. The data type tells PostgreSQL what kind of data the column will store. Choosing the correct data type is important for data integrity and performance.

**Most Common Data Types for Beginners:**

|  |  |  |
| --- | --- | --- |
| Data Type | What it Stores | Common Use Case |
| SERIAL | Auto-incrementing integer (1, 2, 3, ...) | Perfect for a unique id column (Primary Key). |
| INT or INTEGER | Whole numbers (positive or negative). | age, stock\_quantity, year |
| VARCHAR(n) | Variable-length text, up to n characters. | username (VARCHAR(50)), title (VARCHAR(255)) |
| TEXT | Text of any length. | A blog post, a long product description. |
| NUMERIC(p, s) | Exact decimal numbers. p is total digits, s is digits after the decimal. | Price: NUMERIC(10, 2) can store 12345678.99 |
| BOOLEAN | true or false values. | is\_active, is\_published |
| DATE | A date (YYYY-MM-DD). | birth\_date, start\_date |
| TIMESTAMP | A date and time (YYYY-MM-DD HH:MI:SS). | created\_at, last\_login\_time |

**4. constraints (The Rules)**

Constraints enforce rules on the data in your columns. They are essential for preventing bad data from entering your database.

* NOT NULL: This column must have a value; it cannot be left empty.
* UNIQUE: Every value in this column must be unique across all rows. No duplicates allowed.
* PRIMARY KEY: This uniquely identifies each row in the table. It's a combination of NOT NULL and UNIQUE. A table should have **exactly one** primary key. The SERIAL data type is commonly used for primary keys.
* FOREIGN KEY: This creates a link between this table and another table, ensuring that a value in this column matches a value in the primary key of the other table. This is the foundation of relational databases.
* CHECK (condition): A custom rule. For example, you can ensure that a price is always greater than zero: CHECK (price > 0).
* DEFAULT value: If a new row is added without a value for this column, this default value will be used automatically. For example, DEFAULT 0 or DEFAULT NOW().

**Putting It All Together: A Detailed Example**

Let's create a users table for a web application. We'll include comments in the code to explain each line.

SQL

CREATE TABLE users (  
 -- The 'id' column is an auto-incrementing integer. It is the  
 -- primary key, which means it must be unique and not null.  
 -- This is the standard way to identify a user.  
 id SERIAL PRIMARY KEY,  
  
 -- The 'username' is a text field with a max length of 50 characters.  
 -- It must be unique (no two users can have the same username) and  
 -- cannot be empty (NOT NULL).  
 username VARCHAR(50) UNIQUE NOT NULL,  
  
 -- The 'email' field also has to be unique and cannot be empty.  
 email VARCHAR(255) UNIQUE NOT NULL,  
  
 -- The user's age is an integer. The CHECK constraint ensures  
 -- that we only accept ages 18 or older.  
 age INT CHECK (age >= 18),  
  
 -- 'is\_active' is a boolean to track if the account is enabled.  
 -- By default, a new user is active (DEFAULT true).  
 is\_active BOOLEAN DEFAULT true,  
  
 -- 'created\_at' stores the exact date and time the user account was created.  
 -- The default value is the current timestamp when the row is inserted.  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);

**After Running the Command**

Once you execute the CREATE TABLE users; command, PostgreSQL will:

* Register a new, empty table named users in your database.
* Define the columns id, username, email, age, is\_active, and created\_at.
* Set up all the constraints (PRIMARY KEY, UNIQUE, NOT NULL, CHECK, DEFAULT) to be automatically enforced for any data you later add to this table.

You are now ready to use the INSERT command to add data to your newly created users table!

**📘 Example 1: Basic Employee Table**

CREATE TABLE employees (

emp\_id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email TEXT,

salary NUMERIC(10, 2),

is\_active BOOLEAN DEFAULT TRUE,

join\_date DATE

);

**📘 Example 2: Product Table with Binary and Timestamp**

CREATE TABLE products (

product\_id SERIAL PRIMARY KEY,

product\_name VARCHAR(150),

description TEXT,

price REAL,

available BOOLEAN,

added\_on TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

image BYTEA

);