PostGreSQL Commands and Usage

Table of Contents

Database Creation

a. postgres=# CREATE DATABASE testdb;

CreateDB commandline

Option	Description
-D tablespace	Specifies the default tablespace for the database.
-e	Echo the commands that createdb generates and sends server.
-E encoding	Specifies the character encoding scheme to be used in t database.
-l locale	Specifies the locale to be used in this database.
-T template	Specifies the template database from which to build this
--help	Show help about dropdb command line arguments, and
-h host	Specifies the host name of the machine on which the se running.
-p port	Specifies the TCP port or the local Unix domain socket fi on which the server is listening for connections.
-U username	User name to connect as.
-w	Never issue a password prompt.
-W	Force createdb to prompt for a password before connect database.

createdb -h localhost -p 5432 -U postgres testdb

Connect to a database

b. At the PSQL prompt
   i. \c databasename
   ii. \d to list all databases
   iii. \l command is used to list all dbs
   iv. \d tablename to describe the table

Create Table

```
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
```

```
        ADDRESS      CHAR(50),
        SALARY       REAL
    );
```

Creating a table in a schema

c. **CREATE SCHEMA myschema;**

d.
```
create table myschema.company(
ID   INT            NOT NULL,
NAME VARCHAR (20)     NOT NULL,
AGE  INT            NOT NULL,
ADDRESS  CHAR (25) ,
SALARY   DECIMAL (18, 2),
PRIMARY KEY (ID)
);
```

Insert

e. INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY,JOIN_DATE) VALUES (1, 'Paul', 32, 'California', 20000.00 ,'2001-07-13');

f. To insert into columns with graphic elements

i. insert into "TESTSCHEMA"."myxyz" values ('one','T','(1,1),(3,2)')

AutoIncrmented Columns

```
CREATE TABLE names (
id SERIAL,
name varchar);

INSERT INTO "names" (name) values ('t1');
INSERT INTO "names" (name) values ('t2');

SELECT * FROM names;
```

Method 2

```
CREATE SEQUENCE mserial START 11;
INSERT INTO distributors VALUES
(nextval('mserial'), 'nothing');
```

Functions

Create OR REPLACE function totrows()

```
RETURNS integer AS $total$
declare
```

```
        total integer;
BEGIN
   SELECT count(*) into total FROM COMPANY;
   RETURN total;
END;
$total$ LANGUAGE plpgsql;
```

select totrows();

Triggers

Triggers are table callback functions

```
CREATE TRIGGER example_trigger AFTER INSERT ON COMPANY
FOR EACH ROW EXECUTE PROCEDURE testfunc();

CREATE OR REPLACE FUNCTION testfunc() RETURNS TRIGGER AS
$example_table$
BEGIN
  INSERT INTO AUDIT(EMP_ID, ENTRY_DATE) VALUES (new.ID,
  current_timestamp);
RETURN NEW;
END;
$example_table$ LANGUAGE plpgsql;
```

tables used for the trigger

```
CREATE TABLE COMPANY(
        ID INT PRIMARY KEY     NOT NULL,
        NAME           TEXT    NOT NULL,
        AGE            INT     NOT NULL,
        ADDRESS        CHAR(50),
        SALARY         REAL
        );

CREATE TABLE AUDIT(
         EMP_ID INT NOT NULL,
         ENTRY_DATE TEXT NOT NULL
                 );
```

References (Foreign Keys), CHECK, EXCLUDE

```
CREATE TABLE EMPS(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL CHECK(SALARY > 0),
);

CREATE TABLE DEPINFO(
```

```
ID INT PRIMARY KEY      NOT NULL,
DEPT          CHAR(50) NOT NULL,
EMP_ID        INT      references EMPS(ID)
      );
```

What happens when you insert rows into the EMP TABLE , with different salaries values > 0 and less than 0?

Expressions

g. **Boolean Expressions – Select * from test where salary=3000;**
h. **Numeric Expressions – Select (5+6) as Addition ;**
   Select Count(*) from emps AS "Total" from table;

i. Date Expressions SELECT CURRENT_TIMESTAMP

Cursors

```
CREATE OR REPLACE FUNCTION get_Info(p_year INTEGER)
   RETURNS text AS $$
DECLARE
 titles TEXT DEFAULT '';
 rec_film   RECORD;
 cur_films CURSOR(p_year INTEGER)
 FOR SELECT
 FROM film
 WHERE release_year = p_year;
BEGIN
   -- Open the cursor
   OPEN cur_films(p_year);

   LOOP
    -- fetch row into the film
      FETCH cur_films INTO rec_film;
    -- exit when no more row to fetch
      EXIT WHEN NOT FOUND;

    -- build the output
      IF rec_film.title LIKE '%ful%' THEN
         titles := titles || ',' || rec_film.title || ':'
|| rec_film.release_year;
      END IF;
   END LOOP;

   -- Close the cursor
   CLOSE cur_films;
```

```
   RETURN titles;
END; $$

LANGUAGE plpgsql;
```

Using it

```
SELECT get_film_titles(2006);
```

Returning Tables from Functions

```
CREATE OR REPLACE FUNCTION get_film (p_pattern VARCHAR)
 RETURNS TABLE (
 film_title VARCHAR,
 film_release_year INT
)
AS $$
BEGIN
 RETURN QUERY SELECT
 title,
 cast( release_year as integer)
 FROM
 film
 WHERE
 title LIKE p_pattern ;
END; $$

LANGUAGE 'plpgsql';
```

Declaring Variables

```
DO $$
DECLARE
   counter integer := 1;
   first_name varchar(50) := 'John';
   last_name varchar(50) := 'Doe';
   payment numeric(11,2) := 20.5;
BEGIN
   RAISE NOTICE '% % % has been paid % USD', counter, first_name,
last_name, payment;
END $$;
```