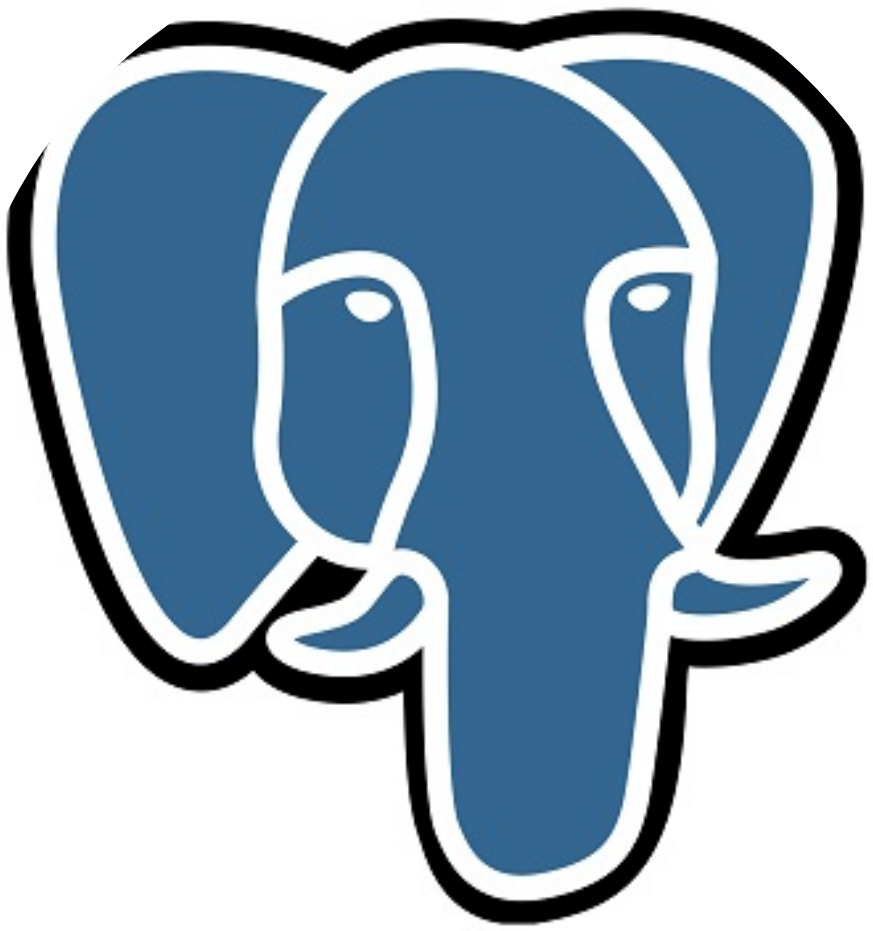

SQL Basics Part V - Creating Databases and Tables

27.10.2019

SQL Basics

OVERVIEW

★Previously we've been mainly reading from tables where the data is already in there.
★In this section, we are gonna focus on how to actually create databases and tables and insert data to those tables and manipulate the data inside of them.

- We will start by understanding data types, primary & foreign keys.
- Then we will practice creating tables and manipulating the rows inside of them.
- Then we will learn how to manipulate entire tables.
- Finally, we will end up with a more in-depth view of some constraint concepts.

DATATYPES

★PostgreSQL supports the following data types

- **Boolean**
- **Character**
- **Number**

- **Temporal i.e, date and time-related data types**
- **Special types**
- **Array.**

★Boolean:

- Can hold true, false or NULL
- We use boolean or bool keyword when we declare a column that has boolean datatype.
- When we insert data into a boolean column, PostgreSQL will convert it into the boolean value e.g, 1, yes, y, t, true are converted to **true**, 0,no,n,false,f are converted **false**.
- When we select data from a boolean column, PostgreSQL display t for true, f for false, and space character for NULL.

★Character: (PostgreSQL has 3 types of char data types:)

- **1. A single character** : char
- **2. Fixed-length character strings**: char(n). If we insert a string that is shorter than the length of the column, PostgreSQL will pad spaces. If we insert a string that is longer than the length of the column, PostgreSQL will issue an error.
- **3. Variable-length character strings**: varchar(n) : We can store up to n characters.

★Number: (PostgreSQL provides 2 distinct types of numbers)

- **1. integers**: `smallint` is 2-byte signed, `int` is 4-byte signed, `serial` as same as integer except that PostgreSQL populate value into the column automatically
- **2. floating-point numbers**: `float(n)` is a number whose precision, at least, n up to max 8 bytes. `real` or `float8` is a double-precision (8-byte) numbers. `numeric` or `numeric(p,s)` is a real number with p digits with s number after the decimal point.

★The temporal data types store date and time-related data.

- **date** : stores data data
- **time** : stores time data
- **timestamp**: stores data and time
- **interval** : stores difference in timestamps
- **timestamptz** : stores both timestamp and timezone data.

PRIMARY AND FOREIGN KEYS

PRIMARY KEY

★A primary key is a column or a group of columns that is used to identify a row uniquely in a table.
★We define primary keys through primary key constraints.

★A table can have one and only one primary key.
★It is a good practice to add a primary key to every table.
★When we add a primary key to a table, PostgreSQL creates a unique index on the column or a group of columns used to define the primary key.
★Normally, we add the primary key to a table when we define the table's structure using CREATE TABLE statement

---

Example:

```
CREATE TABLE table_name (
column_name data_type PRIMARY KEY,
column_name data_type,
...                  );
```

FOREIGN KEY

★Field or group of fields in a table that ununiquely identifies a row in another table.
★In other words, a foreign key is defined in a table that refers to the primary key of the other table.
★In PostgreSQL we define foreign key through a foreign key constraint.

CREATE TABLE

★ To create a new table in PostgreSQL, we use the **CREATE TABLE** statement.
★ Here is the syntax:

Example:

★First, we specify the name of the new table after the CREATE TABLE clause:

```
CREATE TABLE table_name
```

★Next, we will list the column name, its data type, and column constraint.
★We can have multiple columns in a table, each column is seperated by a comma (,).
★The column constraint defines the rules for the column e.g., NOT NULL.
★After the column list, we define a table level constraint that defines rules for the data in the table.

```
(column_name TYPE column_constraint, table constraint)
```

★After that, we specify existing table from which the table inherits. It means the new table contains all columns of the existing table and the columns defined in the CREATE TABLE statement.

```
INHERIT existing_table_name;
```

★Note that the INHERIT part is optional. We don't need to inherit from another table but sometimes it is easier.

COLUMN CONSTRAINTS

★ **NOT NULL** : the value of the column cannot be NULL

★ **UNIQUE** : the value of the column must be unique across the whole table. However, the column can have many NULL values because PostgreSQL treats each NULL value to be unique.

★ Notice that SQL standard only allows one NULL value in the column that has the UNIQUE constraint.

★ **PRIMARY KEY** : this constraint is the combination of NOT NULL and UNIQUE constraints.

★ We can define one column as PRIMARY KEY by using column-level constraint. In case the primary key contains multiple columns, we must use the table-level constraint.

★ **CHECK** : enables to check a condition when we insert or update data. For example, the values in the price column of the product table must be positive values.

★ **REFERENCES**: constrains the value of the column that exists in column in another table. We basically use references to define the foreign key constraint.

TABLE CONSTRAINTS

★ They are similar to column constraints except that they are applied to the entire table, rather than to individual column.

★ **UNIQUE (column_list)** : to force the the value stored in the columns listed inside the parantheses to be unique.

★ **PRIMARY KEY (column_list)** : to define the primary key that consists of multiple columns.

★ **CHECK (condition)** : to check a condition when inserting or updating data.

★ **REFERENCES**: to constrain the value stored in the column that must exist in a column in another table.

Actually Creating Tables in pgAdmin

★ First, we build a new database. It will be empty. Select the new database and open a query tool. Now I am gonna write the SQL query to create the tables inside this database.

Example:

★Let's create the account table:

```
CREATE TABLE account
( user_id serial PRIMARY KEY,
  user_name VARCHAR(50) UNIQUE NOT NULL,
  password VARCHAR(50) NOT NULL,
  email VARCHAR(355) UNIQUE NOT NULL,
  created_on TIMESTAMP NOT NULL,
  last_login TIMESTAMP );
```

★Now I have created an empty table with some constraints.

Example:

★Let's create the roll table consists of 2 columns:

```
CREATE TABLE role
( role_id serial PRIMARY KEY,
  role_name VARCHAR(255) UNIQUE NOT NULL);
```

Example:

★Here is a use of table constraint:

```
CREATE TABLE account_role
( user_id integer NOT NULL,
  role_id integer NOT NULL,
  grant_date timestamp without time zone,
  PRIMARY KEY(user_id,role_id),
  .
  .
  .
);
```

INSERT

★ When we create a new table, it does not have any data.
★ SQL provides **INSERT** statement that allows us to insert one or more rows into a table at a time.

★ Adding a single row to a table:

Example:

★First, we specify the name of the table that we want to insert a new row followed by a comma-separated column list.
★Then we list a comma-separated value list after the VALUES clause. Value list must be in the same order as the columns list specified.

```
INSERT INTO table_name(column1, column2, ...)
VALUES (value1, value2, ...)
```

★ Adding multiple rows into a table:

Example:

```
INSERT INTO table_name(column1, column2, ...)
VALUES (value1, value2, ...),
       (value1, value2, ...),
       .
       .                    ;
```

★ To insert data that comes from another table:

Example:

```
INSERT INTO table_name
SELECT  column1, column2, ...
FROM  another_table
WHERE condition ;
```

★ Complete example:

Example:

★First, we will create a new table:

```
CREATE TABLE link
( ID serial PRIMARY KEY,
  url VARCHAR(255) NOT NULL,
  name VARCHAR(255) NOT NULL,
  description VARCHAR(255),
   rel VARCHAR(50) );
```

★Now let's insert a row:

```
INSERT INTO link(url, name)
VALUES ('www.erdoganyildiz.com', 'Personal Web Page');
```

★Now let's insert another row:

```
INSERT INTO link(url, name)
VALUES ('www.google.com', 'Google');
```

★Now let's insert multiple rows

```
INSERT INTO link(url, name)
VALUES ('www.bing.com', 'bing'),
       ('www.amazon.com', 'amazon');
```

★ Finally let's insert data to a table from another table:

Example:

★First, we will create another table that is structured like the link table.

```
CREATE TABLE link_copy (LIKE link);
```

★Now let's insert data from the link table to the link_copy:

```
INSERT INTO link_copy
SELECT  *
FROM  link
WHERE name = 'bing' ;
```

UPDATE

★ To change the values of the columns in a table, we use the UPDATE statement.

Syntax:

```
UPDATE table_name
SET column1 = value1,
    column2 = value2,
    ...
WHERE condition;
```

★Where condition allows us to update only certain rows instead of all rows in specified columns.
★ Yani Where clause kullanmazsak tüm column1 values = value1 ve tüm column2 values = value2 olacak.

★ Let's update the link table created in the previous section:

Example:

```
UPDATE link
SET description = 'Name starts with an A'
WHERE name LIKE 'A%';
```

★We fill the empty description column.

DELETE

★ To delete rows in a table, we use DELETE statement as follows:

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

★If we omit a WHERE clause then all the rows in the table will be deleted.

★ The delete statement returns the number of rows deleted.

Syntax:

```
DELETE FROM link
WHERE name LIKE 'B%';
```

★So the rows that name column starts with the letter B are deleted.

ALTER TABLE

★ To change the existing table structure, we use ALTER TABLE statement.

Syntax:

```
ALTER TABLE table_name action;
```

★ PostgreSQL provides many actions that allow us to:

- Add, remove, rename column
- Set default value for the column
- Add CHECK constraint to a column
- Rename tableUN

★ The keywords to use will be:

- **ADD COLUMN**
- **DROP COLUMN**
- **RENAME COLUMN**
- **ADD CONSTRAINT**
- **RENAME TO**

Example:

★First, let's create a table:

```
CREATE TABLE link
(link_id serial PRIMARY KEY,
 title VARCHAR(512) NOT NULL,
 url VARCHAR(1024) NOT NULL UNIQUE);
```

★Now let's play around with this table using ALTER TABLE statement. First, we will add a new boolean type column called isActive:

```
ALTER TABLE link ADD COLUMN isActive boolean;
```

★Let's drop a column:

```
ALTER TABLE link DROP COLUMN isActive;
```

★Let's rename a column:

```
ALTER TABLE link RENAME COLUMN title TO new_title;
```

★Let's rename the whole table:

```
ALTER TABLE link RENAME TO link_table;
```

DROP TABLE

★ To remove existing table from the database, we use the DROP TABLE statement.

Syntax:

```
ALTER TABLE [IF EXISTS] table_name;
```

★ The [IF EXISTS] term is an optional statement to avoid errors if a table does not exist.

CHECK Constraint

★ A CHECK constraint is a kind of constraint that allows us to specify if a value in a column must meet a specific requirement.
★ The CHECK constraint uses boolean expression to evaluate the values of a column.
★ If the values of the column pass the check, PostgreSQL will insert or update those values, otherwise, it will throw an error.

Example:

★So while creating the table we put CHECK constraints to some columns:

```
CREATE TABLE new_users
(id serial PRIMARY KEY,
 first_name VARCHAR(50),
 birth_date DATE CHECK(birth_date > '1900-01-01'),
 join_date DATE CHECK(join_date > birth_date),
 salary int CHECK(salary > 0) );
```

NOT NULL Constraint

★ In database theory, NULL is unknown or missing information.
★ The NULL value is different from empty or zero.
★ If a value is NULL it means we don't know if it is zero or hundred or empty. But if we have zero or empty as a value it means we know this value is empty or zero.

★ So by using the NOT NULL constraint. Whenever we insert or update data, we must specify a value that is different from the NULL value.

Example:

```
CREATE TABLE learn_null
(first_name VARCHAR(50),
 sales integer NOT NULL);
```

UNIQUE Constraint

★ The UNIQUE constraint allows us to be sure that the value in a column or a group of columns is unique in a table.
★ Sometimes we want to ensure that the value in a column or a group of columns is unique across the whole table such as an email address, username, employee id, etc.

★ With the UNIQUE constraint, every time we insert a new row, PostgreSQL checks if the value is already in the table.

Example:

```
CREATE TABLE people
(id serial PRIMARY KEY,
 first_name VARCHAR(50),
 email VARCHAR(100) UNIQUE);
```

★Every single value under the email column is going to be unique.

More blogs