Introduction to SQL

Pieter Joubert

March 12, 2022

Introduction to SQL

Preliminaries

Disclaimer

These lectures will be focused on *PostgreSQL* and, more importantly, the *psql* commandline tool. The skills are transferable to other Database Management Systems as well as other front ends to PostgreSQL. Try not to focus too much on the tools themselves but rather focus on the concepts, and *SQL* as a language.

Note also that we will not be covering ALL possible commands that exist in *psql* and *SQL*. It is up to the student to explore additional commands as necessary.

Preliminaries

Remark

Please make sure to refer back to the content covered in the past two weeks as that forms the *theoretical* foundation of everything we are doing practically in the next three weeks.

Table of Contents

Section 1

PostgreSQL

Installing PostgreSQL

Video Tutorials

Please remember to check on Canvas for Video Tutorials on installing PostgreSQL

- From this point onward we are assuming that the PostgreSQL DBMS (Database Management Server) is installed on the machine you are working on.
- We will be using a combination of *psql* and *SQL* commands in this class.
- Please note that the *psql* commands will only work the PostgreSQL, while the *SQL* commands should work with any DBMS.

Superusers

Warning

When installing PostgreSQL a *Superuser* account should have been created. This account has full access to all the server features, which can be a security risk. For this reason we are going to setup a user that we can use for a specific Database.

Using psql

- To be able to add a new user (and perform all the commands we want to on the database), we need to use the *psql* commandline tool.
- Open up your commandline or shell tool (Windows: cmd or Powershell, Mac: Terminal, Linux: any installed terminal tool)
- Type in the following command to open psql with the postgres database:

```
01 |  psql postgres
```

Setting up users for PostgreSQL

- Creating a new user named *testuser* with a password *password*. Note the use of single quotes for string literals:

```
01 |  CREATE ROLE testuser
02 |      NOINHERIT LOGIN PASSWORD 'password';
```

- Viewing all the users in a database:

```
01 |  \du
```

- Notice how our new *testuser* has no attributes (i.e. permissions associated with it)

psql vs SQL

Syntax

Note the difference in syntax between SQL commands and psql instructions. SQL commands end in a semi-colon, while psql commands start with a backslash. Reserved words in SQL (e.g. CREATE, SELECT etc.) are generally written in all upper case to make them easier to read. Most DBMSes are not case sensitive but it is always good practise to make sure your case keeps matching.

Section 2

Creating a Database and Tables

Creating a Database

- We cannot grant database specific permissions to our new user if we don't have a database that we want them to use.
- We first need to create a data base as follows:

```
01 |  CREATE DATABASE todo_list
02 |      WITH
03 |      OWNER = testuser
04 |      ENCODING = 'UTF8'
05 |      CONNECTION LIMIT = -1;
```

- We can now connect to our new database with our new user as follows:

```
01 |  \c todo_list testuser
```

Setting up user permissions

- Because we made *testuser* the owner of *todo_list*, *testuser* can do almost everything we want with the *todo_list* database.
- We can *GRANT* and *REVOKE* more fine-grained permissions using code similar to the examples below (assuming a table named *todo_item* exists):

```
01 |  REVOKE SELECT ON public.todo_item FROM testuser;
02 |  GRANT SELECT ON public.todo_item TO testuser;
```

Creating a Table

- To create a table we use the SQL command shown below. We'll discuss this command in more detail in the next couple of slides:

```
01 |  CREATE TABLE public.todo_item
02 |  (
03 |      todo_item_id integer NOT NULL,
04 |      description text,
05 |      owner_id integer,
06 |      priority_id integer,
07 |      context_id integer,
08 |      status_id integer,
09 |      project_id integer,
10 |      due_date date,
11 |      completion_date date,
12 |      PRIMARY KEY (todo_item_id)
13 |  );
```

Schemas

- In PostgreSQL we can organise tables within a Database according to *schemas*.
- A schema allows us more control over access to a Database.
- For example we can have a schema setup for management and another for employees.
- The *management* schema could include more tables (e.g. payroll information), while the *employee* scheme would be limited to tables the employees need for daily work only (e.g. sales information).
- In our example we are going to be using the default *public* schema, that includes all tables in a database, and is created automatically when we create a database.

Table Columns

- Each table we create needs to define columns that store the various attributes of one row in our table.
- For example, we need to define what attributes (e.g. description, due date etc.) we need to store per todo item in our table.
- Each column definition includes the name of the column, the datatype, and a number of other optional parameters
- The names of the column follow a similar set of rules to naming variables in a language like Java, e.g. no spaces, no special characters, cannot start with a number etc.

Data types

- Each column we add to a Table needs to have a Datatype.
- In the same way we declare variables in Java as *String*, *int* etc. we need to define data types for the columns in our Tables.
- The example above has the *integer*, *text* and *date* datatypes.
- There are many more datatypes available in PostgreSQL so please take some time to research what they are and when they should be used.

Defining constraints

- In any database we need to define various relationships and constraints.
- One of the main constraints we will define is the *PRIMARY KEY* constraint.
- In the example about we set the *todo_item_id* as the *PRIMARY KEY* for that table.
- we also define that the *todo_item_id* cannot be null, using the *NOT NULL* optional definition.
- We need both of these definitions in place to ensure that our *PRIMARY KEY* constraint will work correctly.

Foreign Keys

Note

The *todo_item* table mostly contains Ids that will link to primary keys in other tables that we will still create. This means for the moment our data will mostly be a bunch of Ids but we will look at ways to link everything together in the upcoming weeks.

Section 3

Basic SQL commands

CRUD

- The basic actions we can perform on a table are Creating new data, Reading existing data, Updating (or Editing) existing data and, Deleting existing data.
- These actions are known by the acronym CRUD (Created, Read, Update Delete)
- In SQL these actions are performed by the INSERT, SELECT, UPDATE and DELETE commands.

INSERT into a table

- To insert into a Table we need to specify the values that we want to insert as a comma separated list in the same order that the columns are organised in the table:

```
01 |  INSERT INTO public.todo_item
02 |          VALUES (0, 'New Task', 0, 0, 0, 0, '
       2022-02-02', null);
```

- We can also explicitly specify the column order to insert into:

```
01 |  INSERT INTO public.todo_item (todo_item_id,
       description, owner_id, priority_id,
       context_id, project_id, due_date,
       completion_date)
02 |          VALUES (1, 'Task 2', 0, 0, 0, 0, '
       2022-02-02', null);
```

UPDATE existing data

- To update data in a table, we need to define the table in which we want to make the update, the column(s) that we want to update, the new value we want use, and finally, *where* in the table we want to make the update(s).

```
01 |  UPDATE public.todo_item SET
02 |  completion_date = now()
03 |  WHERE todo_item_id = 0;
```

- In this example we are using a built-in SQL function named *now* that returns the current date, but we could update a column to any valid value:

```
01 |  UPDATE public.todo_item SET
02 |  completion_date = '2022-03-01'
03 |  WHERE todo_item_id = 0;
```

SELECT from a table

- When we want to read data from a Table we use the SELECT command:

```
01 |  SELECT * from public.todo_item;
```

- In this example we are selecting all the columns and rows in a table. To select a specific column we use:

```
01 |  SELECT description from public.todo_item;
```

- Finally to specify a specific row (or rows) we use a WHERE clause:

```
01 |  SELECT description from public.todo_item WHERE
         todo_item_id = 1;
```

DELETE from a table

- If we want to delete data from a table we use the DELETE command:

```
01 |  DELETE FROM public.todo_item WHERE due_date < '
       2022-02-03';
```

- We don't need to specify a column as we can only delete an entire row.
- We've also define a specific WHERE clause that deletes all items whose complete dates are older than a specified date.

DELETING

Warning

Please note that you need to specify a WHERE clause when deleting or else the DELETE command will delete all the rows in a table.

WHERE clause

Note

We've been using the WHERE clause to specify a specific column (or columns) in various ways. There are many more permutations we can use to have fine-grained control over exactly which rows we want to specify. We will be looking at these in more detail in the next two weeks, but please feel free to do some research regarding the WHERE clause so long.

Saving your SQL

Hint

It might be a good idea to save the sql scripts you run in a text file. If a script runs successfully, copy and paste it into a file and save it so that you can re-use it later. We will look at running sql scripts from a file in a later lecture.

Section 4

Lecture summary

Lecture summary

- Installing and setting up PostgreSQL
- Setting up users and permissions
- Basic SQL queries

Questions

Thank you! Questions?