SQL Querying

Pieter Joubert

March 20, 2022

SQL Querying

Table of Contents

Section 1

Importing Data

Existing Data

Dealing with temporary data

While we are developing a database, we might fill it with some temporary or test data. It is critically important to remove this data once the structure is finalised, both for reasons of security and data integrity.

DROP

- Occasionally we want to completely remove an existing table (or database). In this case we use the *DROP* command.

```sql
DROP TABLE public.status;
```

- Be careful when using the *DROP* command as that will completely remove whatever database structures you set, as well as all data that was stored in that structure.

TRUNCATE

- More often we would prefer to keep the database structure, but remove all the data in the database. In this case we can use the *TRUNCATE* command:

```
TRUNCATE TABLE public.status;
```

- In general we prefer to use *DELETE* when we need to remove a subset of rows from a table. *DELETE* performs the remove row-by-row, so *TRUNCATE* will be much faster than *DELETE* for removing all the rows from a table.

Comma separated file

- There are a number of different formats we can use to store data in a raw file.
- You used a *tab-separated* file in Assignment 1. Another very common format is a *comma-separated* file:

```
0,Carl,Sagan
1,Neil,de Grasse Tyson
2,Andrea,Ghez
```

Import from csv - create table

- To be able to import from a csv file we first need an empty table to import into. Let's create a new *owner* table. This will store data about the users of the program who will *own* various todo items.

```sql
CREATE TABLE public.owner
(
    owner_id integer NOT NULL,
    name character varying(50),
    surname character varying(50),
    PRIMARY KEY (owner_id)
);
```

Import from csv - COPY command

- Once we have a *.csv* file with data and an empty table to import the data into, we canuse the *COPY* command to import the data:

```
\copy owner(owner_id, name, surname)
      FROM '~joubertp/Documents/owners.csv'
      DELIMITER ',';
```

- We need to define the table we want to copy into, the exact file location where the *.csv* file is stored, and what character is used to separate values in the file (in this case a comma).

\copy vs. COPY

- The standard SQL command to import data from a file is *COPY*.
- To be able to use it we need superuser permissions.
- We could connect to our database using our superuser and use the *COPY* command.
- Instead, to keep our permissions more clearly defined, we are rather going to use the \copy psql command.

Section 2

Creating Constraints

Constraints

- We've only created a single type of constraint, the *PRIMARY KEY* constraint.
- We've also only created a constraint when we create the table.
- We're now going to look at *altering* an exist table to include a constraint.

ALTER existing structure

- We can alter the existing structure of a table using the *ALTER* command.
- We are only adding a foreign key in this example but the *ALTER* command can be used to change almost any part of the structure of a table:

```sql
ALTER TABLE IF EXISTS public.todo_item
    ADD FOREIGN KEY (owner_id)
    REFERENCES public.owner (owner_id) MATCH
   SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE
    NOT VALID;
```

- NOT VALID does not validate existing data.

ALTER Create FK constraint

In the example code on the previous slide:

- We first check if the table we want to alter exists.
- Then we add a foreign key, specifying the column in the table that will be the foreign key.
- We then need to reference the foreign table (owner) and the column in that table that we want to link to (owner_id)
- We need to define what happens when a linked key is *Updated* or *Deleted*.
- In this example we *CASCADE* the changes, meaning any changes we make in the foreign table will also change in the primary table. For example if we *DELETE* an owner, all the rows in the todo_item table will also be deleted.

Relationship Design Aproach

When to create FK Relationships

Most of the time it makes more sense to design as much of your database, in as much detail as possible before you implement it. In this way you can define the Relationships when you create the Table, and not have to alter the table afterwards. In our example we have already created our primary table (todo_item), so we will just be altering this table going forward.

Default Constraint

- We can also alter (or add when creating the database) a default constraint:

```
ALTER TABLE public.todo_item ALTER COLUMN
        due_date SET DEFAULT now() + '7days';
```

- This constraint will make the default due_date 7 days from the current date, if no due_date is specified.

Checking existing constraints and relationships

- We can check the relationships and constraints on a specific table using the \d command in psql.

```
\d todo_item
```

Section 3

SQL Functions

Functions

- There are a number of SQL functions that we can use to perform additional analysis on the data that we query using the select statement.
- All of these functions return either additional information, (e.g. the Sum() function), or they format the result of the query in some way (e.g. the SORT command)

Count()

- We can use the *Count()* function to return the number of rows a specific query returns:

```sql
SELECT COUNT(*)
    FROM public.todo_item
    WHERE owner_id = 1;
```

- This query would return the number of todo_item rows that belong to owner_id 1.

Sum()

- The *Sum()* function returns the SUM of all the numeric values in the selected columns:

```
SELECT SUM(completion_date - due_date)
    FROM public.todo_item;
```

- This query returns the total number of days *all* tasks currently in the todo_item table took to complete.
- We actual perform a subtraction on each row (to get the number of days a single task took), and then use the *Sum()* function to add all those individual durations together.

Min() and Max()

- The *Sum()* function is can be quite useful but in our example it does not make as much sense.
- It would be more interesting to see what the *minimum* and *maximum* number of days a task took would be:

```sql
SELECT
    MAX(completion_date - due_date),
    MIN(completion_date - due_date)
    FROM public.todo_item;
```

- Note that we can include multiple different functions in a single *SELECT*.

Query Result Formatting

- The result from the previous query works but it is not that useful in it's presentation:

```
  max | min
 -----+-----
   57 |   5
 (1 row)
```

- We can improve the legibility of our displayed data using the *AS* command.

The AS Command

- The *AS* command renames the column in a result.
- This is useful if we need to make our results more clear, or if we need to display values from a foreign key instead of an ID (which we will look at later when we do JOINs)
- Updating our previous query to include the *AS* command looks as follows:

```
SELECT
    MAX(completion_date - due_date) AS "Max
    completion time",
    MIN(completion_date - due_date) AS "Min
    completion time"
    FROM public.todo_item;
```

- Note the use of the *double quotes* when defining the column name, instead of the *single quote* when defining a string literal.

AS Result

- The result from the previous would now look as follows:

```
  Max completion time | Min completion time
---------------------+---------------------
                  57 |                   5
(1 row)
```

- Remember also that the renamed column will also be sent to any programming code that is running this querying, which can make presenting the result in an application much easier.

Avg()

- Another useful way to examine the todo_item table might be in the context of the *average* number of days a tasks takes:

```
SELECT
    AVG(completion_date - due_date)
    AS "Average number of days per task"
    FROM public.todo_item;
```

- HINT: You can use the *Round()* function to round off the number of days to a more readable format.

ORDER BY ASC and DESC

- A critical part of presenting queried data is *Sorting* it in a useful manner.
- We can sort (or order) our results by a specific column, either in *ascending* or *descending* order:

```
SELECT * FROM todo_item
    ORDER BY description ASC;
```

- We can change whether to order in *ascending* or *descending* order by using the *ASC* or *DESC* command.

Section 4

Lecture summary

Lecture summary

- Importing Data
- Creating Constraints
- SQL Functions

Questions

Thank you! Questions?