A Brief Introduction to Using

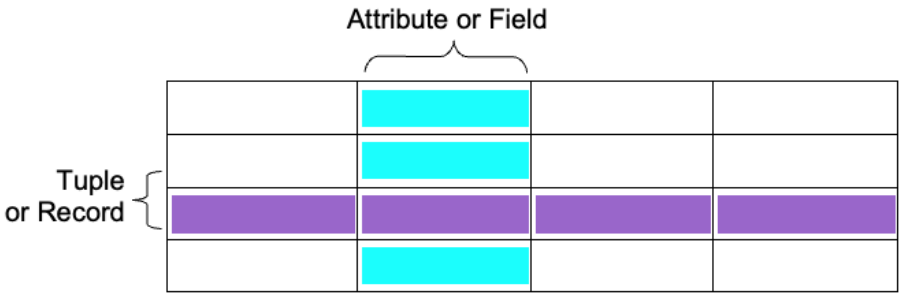
PostgreSQL

by Derek C. Schuurman
Calvin University
Department of Computer Science
Version 0.99
January 2023

Contents

1 Introduction to Databases

A database is a structured collection of logically related data. One common type of database is the relational database, a term that was originally defined and coined by Edgar Codd in 1970. In a relational database the data is stored in 2-dimensional tables of rows (called tuples or records) and columns (called attributes or fields). A table or relation is defined as a collection of records or tuples that have the same fields or attributes.



A Relational Database Management System (RDMS) is software used to manage and use a relational database. There are numerous commercial and open source RDMS software systems available. Some examples of open source RDMS's include MySQL and PostgreSQL.

1.1 Structured Query Language (SQL)

Structured Query Language or SQL is a language used to interact with a relational database system and is maintained as an ISO standard. SQL provides a convenient level of abstraction to interact with a database to define, manage, and query data. SQL is a declarative programming language as opposed to an imperative programming language such as the C programming language in which all the computations are explicitly stated step-by-step. A declarative language is one that defines what the program should accomplish, rather than describing *how* to go about accomplishing it. An SQL statement is processed by the RDMS which determines the best way to return the requested data. Although there are small differences in SQL syntax between RDMS systems, the syntax is largely the same across many systems. For the remainder of this tutorial will assume the use of PostgreSQL.

2 Using PostgreSQL

Although there are some graphical user interfaces available for PostgreSQL, this tutorial will focus on the command line interface for PostgreSQL. To enter the command line front-end to a local PostgreSQL server, type:

```
psql dbname username
```

Where `dbname` is the database name and `username` is the username that has been assigned to you by your system administrator. If you are using cloud-based database service, type the following:

```
psql postgres://username:password@hostname/database
```

where `postgres://username:password@hostname/database` is the URL provided by your cloud database provider. An interactive prompt should appear when you press enter. When you connect using psql you are connected to a particular database. To access a different database, you must initiate a new connection.

A database is a collection of tables. Once you are connect to a database, you can create, modify, and access tables in the database.

2.1 Creating Tables

To create a table within the current database, you use the CREATE TABLE statement. For example, to create a table of students names type the following:

```
CREATE TABLE students (
             studentnumber int NOT NULL,
             lastname varchar(50) NOT NULL,
             firstname varchar(50) NOT NULL
           );
```

The table name is specified after CREATE TABLE statement and then the columns names are given followed by data type, size, NOT NULL or not. A field which is specified as NOT NULL must contain a value. Identifiers such as column names are converted to lowercase in PostgreSQL unless they are enclosed in double quotes. In order to view details about a table, including information about fields and data types, use the DESCRIBE statement. For example, type:

```
\d students;
```

This will return information about the table we just created and information about the fields that comprise it. The output from PostgreSQL should resemble the following:

```
    Column     |          Type          | Collation | Nullable |
---------------+------------------------+-----------+----------+
 studentnumber | integer                |           | not null |
 lastname      | character varying(50)  |           | not null |
 firstname     | character varying(50)  |           | not null |
```

To modify the structure or type of data in existing tables, PostgreSQL provides an ALTER command. The data type of each of PostgreSQL fields or attributes must also be specified. PostgreSQL supports a variety of numeric, character and binary data types. Some of the common data types supported in PostgreSQL are summarized in the following table:

Data Type	Description
integer	A signed integer (4 bytes)
double precision	Double precision floating-point number (8 bytes)
date	Date in the format YYYY-MM-DD
timestamp	Date and time
char [(n)]	A fixed-length string with a length of n characters
varchar [(n)]	A variable-length string with a length of up to n characters
text	variable-length character string

In order to show all the tables in a database, you can use the SHOW TABLES statements as follows:

```
\dt
```

To add records to the students database, use the INSERT statement. For example, to add "John Calvin" with a student ID number of 12345 into the students table, do the following:

```
INSERT INTO students
(studentnumber,firstname,lastname) VALUES
(12345, 'John', 'Calvin');
```

2.2 PostgreSQL Queries

You can also ask PostgreSQL to search for data by submitting a query asking for all the records or rows that match a specific criteria. The SQL SELECT statement is used to perform queries on an SQL table. For example, to list all the students in the table, type: SELECT * FROM students; The * indicates that all columns should be returned. The SELECT query returns the requested data as text in a tabular format like follows:

```
  studentNumber | lastName | firstName |
+---------------+----------+-----------+
          12345 | Calvin   | John
```

To display specific columns, replace the * with a comma-separated list of columns that you would like to see displayed. For example, to display just the lastName and firstName columns, type:

```
SELECT lastname, firstname FROM students;
```

To list the students in alphabetical order using the ORDER BY clause as follows:

```
SELECT * FROM students ORDER BY lastname, firstname;
```

The order can be explicitly set to be ascending or descending by placing the ASC or DESC keywords at the end of the query. To prevent the SELECT statement from returning duplicate values in the results, the DISTINCT keyword can be used. For example, to list all the distinct first names in the students table, type:

```
SELECT DISTINCT firstname FROM students;
```

It is also possible to restrict the search to meet specific criteria using the WHERE keyword. For example, to list all the students who have the first name of "John", type:

```
SELECT * FROM students WHERE firstname = 'John';
```

The conditions to restrict search results can be further combined with boolean operators such as AND and OR operators to express search based on different conditions. In addition to WHERE, there are other keywords such as LIKE and BETWEEN which can be used to restrict the results of a search. For example, to list all the students whose first name starts with a "J", type:

```
SELECT * FROM students WHERE firstname LIKE 'J%';
```

where % is a wildcard which matches any character or sequence of characters. The BETWEEN keyword will restrict a search to values that fall in a range between some minimum and maximum value. For example, to return all the last names that lie alphabetically between "Calvin" and "Luther", type:

```
SELECT * FROM students WHERE lastname
BETWEEN 'Calvin' AND 'Luther';
```

One can also query the number of rows that match a certain condition using the COUNT keyword:

```
SELECT COUNT(*) FROM students WHERE firstname LIKE 'J%';
```

To delete data from a table, use the DELETE statement. For example, to delete all the students with the last name of "Calvin", do the following:

```
DELETE FROM students WHERE lastname = 'Calvin';
```

To make another table called courses that stores a course code and student number for each course a student is enrolled in, use the CREATE statement again as follows:

```
CREATE TABLE courses (
             studentnumber int NOT NULL,
             coursecode varchar(7) NOT NULL );
```

In this table the studentnumber field will not necessarily be unique since it will be appear once for each course in which a student is enrolled. The courseCode will also not necessarily be unique since it will be repeated for each student in the course. Note that the students names do not need to be stored again; they can be retrieved if required by looking up the studentNumber in the students table. To add some records to the courses database, use the INSERT statement once again:

```
INSERT INTO courses (studentnumber,courseCode)
VALUES (12345, 'CSC101A');
```

To change or modify data in a table, use the UPDATE keyword as follows:

```
UPDATE students SET lastname = 'knox'
WHERE studentnumber = 12345;
```

This statement will modify the students table and replace the studentNumber for the student with the name John Calvin. It is possible to modify multiple field values with an UPDATE statement using a comma-separated list of assignments. The WHERE clause in this case uses a boolean AND operator to make a more complex condition. You can also ask PostgreSQL to search data from multiple tables by using a JOIN operation. The JOIN keyword relates two or more tables, typically by using values that are common between them. The students and courses database have a common value of studentNumber that can be used to join them. The ON keyword can be used to specify a condition with which to join tables. For example, to list all the first names and last names of students enrolled in CSC101A, you can use a join operation based on the condition of matching a studentNumber in a query as follows:

```
SELECT firstname, lastname
FROM students
   JOIN courses
   ON students.studentnumber = courses.studentnumber
WHERE courseCode = 'CS101A';
```

It is also possible to delete a table and all it contents using the DROP command. This command should be used with care since it permanently deletes your table and cannot be undone.

```
DROP TABLE students;
```

To close a database connection and quit `psql`, type the following:

```
\q
```

3 Exercises

Here are some exercises to help you practice using PostrgreSQL:

1. If you have not done so already, create a new table called students to store student information including: first name, last name, and student ID number. Select appropriate data type for each of the fields in the table.
2. In the same database, create a table called courses that can store: course names, room number (a 3 digit number), and a course ID number (use a 6 character alpha-numeric course code such as "CS101A"), and a teacher ID.
3. Create a table called "teachers" that has a first name, last name, and teacher ID.
4. Finally, create a table of course enrollments in a course called `courses_students` that stores course IDs and student Ids (both of these are foreign keys). Each course that a student is enrolled in will be stored in one row.
5. Using a suitable graphical tool, draw the database *schema* for your tables.
6. Populate your tables with some "fake data" with 12 students, 6 courses, 3 teachers, and then populate the registrations such that several different students are

enrolled in each course.

7. Devise SQL queries to determine the following:
    - all the students in the school
    - all the students in the school in descending alphabetical order
    - the number of students enrolled in a particular course
    - the list of courses for a given student in alphabetical order
    - the teacher name for a given course
    - all the student names taught by a certain teacher in alphabetical order and such that no student names are repeated
    - the number of courses taught by a given teacher