Database Management Systems

Dr Peadar Grant

October 19, 2021

1 Database management systems (DBMS)

For overall history see (**?**).

1.1 Client-server

Most database management systems run in a client-server model, Figure 1.

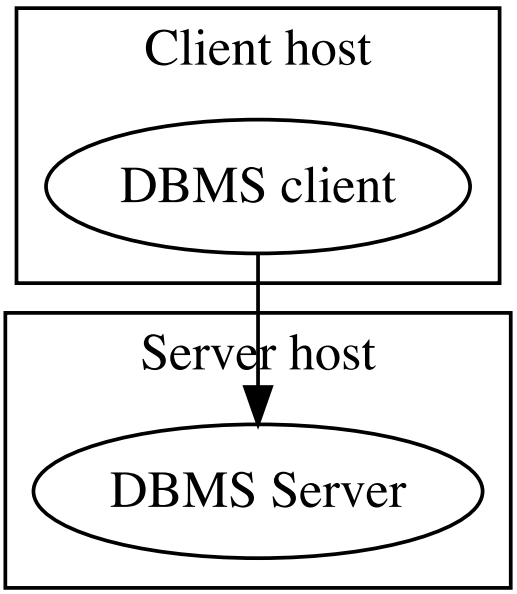

Figure 1: Client-server DBMS

The server process manages the data store and processes requests from clients. The server can be running on any of the following *hosts*:

- Standard laptop / desktop computer
- Dedicated server computer (in a data centre environment)
- Cloud-based virtual host, called a compute instance. (e.g. Amazon EC2)
- A managed database service provided by a cloud service provider (e.g. Amazon RDS, Azure, Google Cloud, IBM Cloud)

The client program accesses the server using a server-specific protocol. Clients normally access through IP networks using TCP on a specified port number. Examples of clients:

- Most databases have a simple command-line client that can send requests to the database and display results
- Apps can be written to access database servers using a client library.
  - Generally the text-mode client uses this library internally too!

Two things to note about the client:

- The client may in some cases be running on the same host as the server.
- Software that is the client of a DBMS may itself be a server.
  - Example: a web application is written in Java using the Spring framework and provides a web server using an embedded Tomcat server. The web application is itself a client of the DBMS it accesses.

This also implies that there is a degree of concurrency, where multiple clients access the same database at the same time.

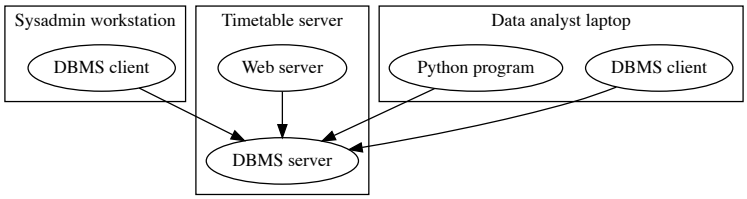

Figure 2: Concurrent access to a DBMS hosting a college timetable

2 Key terms

3 Structured query language (SQL)

For a general overview see (**?**).

4 PostgreSQL

We will focus on PostgreSQL as our primary database. Later on we will introduce other technologies. Reasons:

- Support exists for geospatial data, JSON, XML, full-text search etc.

- It is free software and can be installed on any operating system.

You should bookmark the PostgreSQL documentation.

As we continue we will refer to PostgreSQL as Postgres for brevity.

4.1 Connecting to a remote host

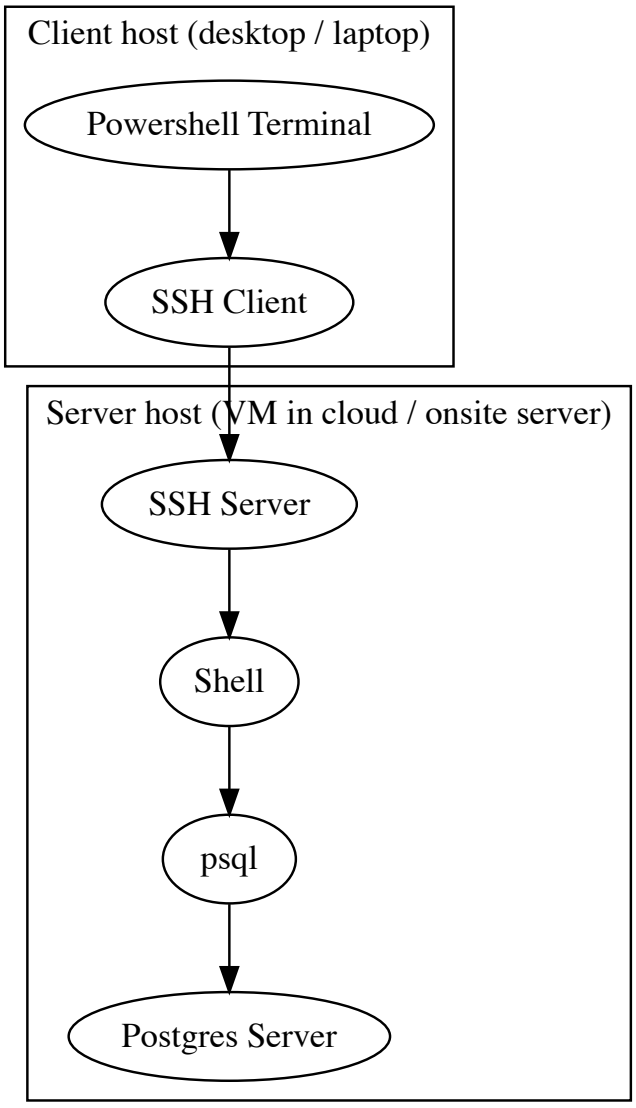

Figure 3: Using psql on a remote host over SSH

5 Database basics

Make sure you can connect to the SSH server given. (Note that this server will not be available outside the time period of this lab session.)

5.1 Running psql

Once logged in, we can use the `psql` command-line client to connect to the PostgreSQL server. Each server can host a number of databases independently of each other. Today we are going to use a database called `population`.

```
psql population
```

Note the DBMS didn't ask us for a username/password. This is because it is setup to trust local connections with the same name as the login username. Generally a DBMS will ask for a username/password.

Once logged in you should see a prompt similar to:

```
psql (9.2.24)
Type "help" for help.


population=>
```

There are two types of command we will encounter:

- SQL statements to be executed by the server
- Client commands (that sometimes get expanded into server-side SQL). These are usually prefixed with a backslash.

You can get a list of all client commands using \?.

5.2 Listing all tables

A database in its simplest form is like an Excel workbook. Just as an excel workbook has one or more sheets, a database may have one or more tables in it. To find out the tables (and other objects) in our DB we can ask the database to describe the tables:

```
\dt
```

Our population database has two tables:

```
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | directions | table | ec2-user
 public | population | table | ec2-user
(2 rows)
```

5.3 Describing a table

A table consists of a number of columns. Each row is a record. We can find out the schema definition using the client command `\tablename`. To find out more about the `directions` table we could say:

```
\d directions
```

Would give us:

```
    Table "public.directions"
   Column   |  Type   | Modifiers
------------+---------+-----------
 fullname   | text    |
 code       | text    |
 vertical   | bigint  |
 horizontal | bigint  |
```

Refer to datatypes in the PostgreSQL manual

6 SELECT queries

We will review basic querying operations in SQL. If you have experience already with relational databases (MySQL, Microsoft SQL, Oracle) you will find most of this section familiar. You should refer to the queries section of the PostgreSQL manual for further detail on the examples below.

6.1 Entire table

Selecting an entire table using the asterisk (*) to pick all columns.

```
select * from directions;
```

Note that statements need to be terminated with the semicolon (;). If you omit the semicolon the prompt will change from => to ->.

In PostgreSQL we can also use the shortened version

```
table directions;
```

6.2 Sorting

Unlike a spreadsheet, the order of rows in result sets from database queries is non-deterministic by default. They are *not* necessarily alphabetical order, insertion order or any other pattern. We use the `ORDER BY` clause to enforce ordering:

```
select * from directions order by fullname asc;
```

The order direction is either `asc` for ascending or `desc` for descending.

See sorting page of PostgreSQL manual for further details.

6.3 Where clause

The `WHERE` clause limits the output

```
select * from directions where vertical=0;
```

Text is best compared using the `LIKE` operator which allows `%` to represent any text.

```
select * from directions where code like '%th';
```

6.4 Column specification

We can specify specific column names

```
select fullname, code from directions;
```

6.5 Column expressions

PostgreSQL supports powerful expressions that combine the values of different columns. The columns themselves need not be selected in the result set.

```
select fullname, code, vertical+horizontal from directions;
```

6.6 Column labels

Columns in the result set receive automatic names. These are normally the name of the column or are derived from an expression. We can assign specific names using the `AS` keyword.

```
select fullname, code, vertical+horizontal as hey from directions;
```

See column labels in PostgreSQL manual.

7 Aggregate functions

Mastering aggregate functions will significantly improve your capabilities to query databases. You should refer to the aggregate functions section of the PostgreSQL manual as you review the following examples.

7.1 Count

The simplest aggregate function is `COUNT` to return the number of rows matched by a specific query. At its simplest, we can tell the number of rows in a table.

```
select count(*) from directions;
select count(*) from population;
```

Count is often used with where to obtain the number of rows matching a specific condition:

```
select count(*) from directions where code like '%th';
```