Appendix A
Getting Started

A.1 Downloading and Installing Postgres and MySQL

To download and install Postgres, go to https://www.postgresql.org/download/. In this page, there are a variety of options for different operating systems. Postgres is available for Windows, Mac OS, and Unix/Linux. The windows version has an installer that downloads and installs the database server as well as pgAdmin, a well-known and commonly used GUI (Graphical User Interface). Likewise, Mac OS also has an installer that downloads and installs a bundle with the server and pgAdmin on it. On top of that, Postgres is the default database for Mac OS since version 10.7 and so a version may already be installed on your Mac. However, this version may not be the latest available. There are other ways to download and install Postgres, all of them described at https://www.postgresql.org/download/macosx/.

To download and install MySQL, go to http://dev.mysql.com/download. In this page, you will see a list of options. You want to use the link to the MySQL Community Server, which is the free database server we have been using in this book.[1] On the page that the Community Server links to, you are given a set of choices depending on the operating systems your computer runs. MySQL has versions for Windows, Mac OS, and Unix/Linux. If you choose the Windows version, you will be recommended to use the MySQL Installer for Windows. If you choose this option (and you should, unless you know what you are doing) you will be taken to another page where there will be two choices: a 'full' installer that downloads about 400+ Mbs for a bundle that contains the server and a bunch of other

[1]MySQL AB, the company behind the software, was bought by Oracle, a large and famous database software company, in 2008. However, the Community Server version remains open-source and free of charge. Another open-source and free database system, MariaDB, has been developed from MySQL and independently of Oracle. You should be able to download and install MariaDB from https://mariadb.com. The MariaDB Community Server is a highly compatible replacement for MySQL and has new features added periodically.

A. Badia, *SQL for Data Science*, Data-Centric Systems and Applications,
https://doi.org/10.1007/978-3-030-57592-2

packages, and the 'Web' installer that only downloads about 25 Mbs. This package allows the user to choose which pieces of software to install. Only the server is needed if one is going to use the CLI (Command Line Interface) to interact with the database. However, it is recommended (especially for first-timers) that a GUI (Graphical User Interface) is also installed (see next section). The most popular GUI for MySQL is MySQL Workbench.

On either systems, as installation proceeds, some dialog boxes will come up. Most of the time, the default options are fine. There is one thing that you should pay attention to: when the database is installed, a first user, called the *superuser*, is created (username in Postgres: `postgres`; in MySQL: `root`). The system will ask for a password for this superuser. It is important that you do not forget this password, since in order to start using the database you will need to connect with the superuser's credentials until more users are created (see Sect. A.3).

A.2 Getting the Server Started

The software you just downloaded and installed contains two vital pieces, called the *server* and the *client*. The server is the part that will take care of storing the data, modifying it, and accessing it. It executes all SQL commands. The client is the part that interacts with the user: it issues a prompt on the screen (when used in command line mode, see below), takes a command from the user, sends it to the server, gets an answer back from the server and displays it for the user, and issues the prompt again. Clearly, server and client must talk to each other: this is called a *connection*. Using a connection, the client sends the server the user commands; the server executes them and gives the client an answer (and also error messages, if anything went wrong). A server can talk to several clients at once; this makes it possible for whole teams of people to access and manipulate the same data, in the same database, at the same time.

Many times, server and client both run in the same computer. However, they can run on different computers; sometimes, a server is installed in a large, powerful computer so it can handle large amounts of data and many clients, while the client is installed in a personal computer. In this case, communication takes place using a network. That is why sometimes when trying to establish a connection, it is necessary to provide the client with a *host* name: this is the name of the computer where the server is (when the server is in the same computer as the client, the name 'localhost' is sometimes used).

Because the server must be ready to accept requests from a client at any time, the server is a program that runs continuously; when no client is contacting it, the server is simply waiting for some client to start a connection. Thus, after installing database software, the first thing to do is to start running the server. In some systems, this is done automatically: the system realizes it is a database server that is being installed, so as soon as it is ready, the system gets the server running and ready to receive connections. By contrast, a client need not be active until a user actually

needs to work with the database. The user activates the client and then uses it to establish a connection with the server.

Before accessing a database, a user must be registered with the database. As indicated above, when a server is installed, a first user is created by default. After starting the server (if not started by default), it is necessary to establish a connection with the database as the default user and create other users so that they can access the database. This is done with a specific SQL command—user management is described in the next section.

There are two types of clients: CLI (command line based) and GUI (Graphical User Interface) based. A command line client is activated by typing the command name in a terminal (the name is `mysql` for MySQL and `psql` for Postgres). With the command, it is typically necessary to give several arguments, including a username and a password as well as a hostname (see above). A GUI client is a separate program that offers a window-based environment to manage connections to the server. There are several free GUI based clients for both MySQL and Postgres; the most popular for MySQL seems to be MySQL Workbench (pictured in Fig. A.1), while the most popular for Postgres is probably pgAdmin (pictured in Fig. A.2). Both can be downloaded and installed in a personal computer from the same sites from which the database server was obtained. When starting a GUI based client, a window pops us and offers the user a set of menus. A connection is started by choosing "Add a server" (an option under the File menu) and then clicking on the Add a Connection icon (in pgAdmin) or by choosing "Connect to a Database" from the Database menu (in Workbench). In either case, a hostname, a database name (the name given in the CREATE DATABASE or CREATE SCHEMA command), a username, and a password must be provided (it is possible to set defaults and/or

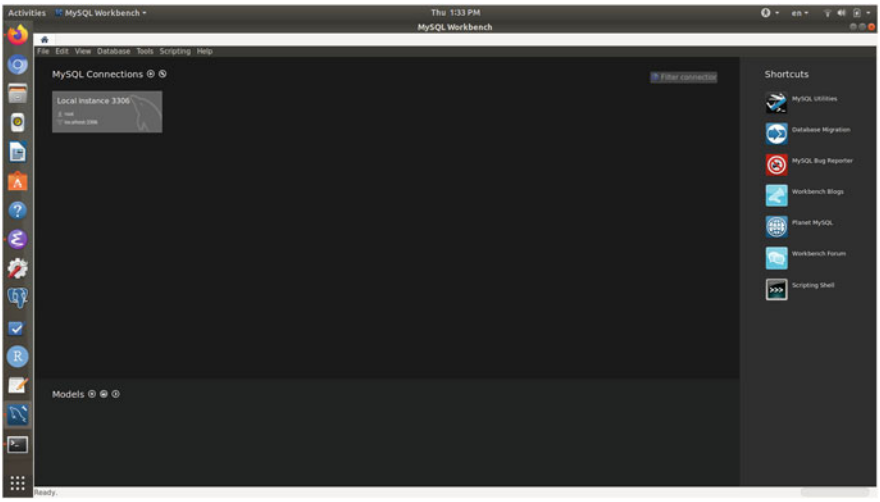

Fig. A.1 A screenshot of MySQL Workbench

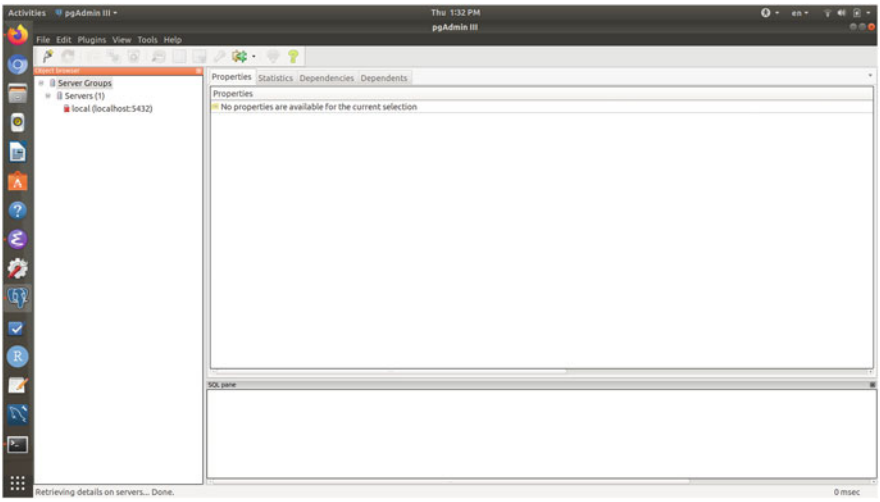

Fig. A.2 A screenshot of Postgres pgAdmin

have the system remember some of those). In most systems, after a connection to a server is established for the first time, it can be saved and an icon created for it. Clicking on it will start the connection without the need to retype anything but the password.

Commands are always sent to the database in SQL. In the command line, the user types SQL; in the GUI, there is always an SQL pane where the user can type SQL, but there are also shortcuts for some common operations (like CREATE DATABASE, CREATE TABLE, and LOAD). In such cases, a combination of icon clicking and filling in some forms will get the job done. However, it is important to emphasize that even when these shortcuts are used, the communication with the server is carried out in SQL. What the GUI client does is to take the user clicks and choices and compose SQL commands automatically (in Workbench, the user is always given the option to check the generated SQL before sending it to the server). Therefore, what we have shown in this book can be used with either type of client.

A.3 User Management

When connecting to the database server, the connection is always as a database user. A user is identified by a *username* and a *password*. As stated earlier, when the database is first installed, a 'superuser' is created. This makes it possible to access the database for the first time; the user simply uses the superuser's username and password. Once logged in as the superuser, it is possible to create (or drop) new users for the database. After a user is created, anyone who knows the username and password of that user can connect to the database as that user.

Unfortunately, user specification is not part of the SQL standard, so each system does it a bit differently. However, most systems have coalesced around a set of very similar operations, so once one is familiar with a given system, it is usually not that different to learn a new one.

Postgres uses the concept of a *role*, which is a user or group of users with similar permissions. In Postgres, the SQL command to create a user is

```
CREATE ROLE rolename WITH PASSWORD password;
```

or, equivalently,

```
CREATE USER username WITH LOGIN PASSWORD password;
```

There are also `DROP ROLE` and `ALTER ROLE` commands.

Associated with each user is a series of *permissions* (also called *privileges*) that tell the system what the user is authorized to do. To give a user permissions on a database, in Postgres one must start by allowing the user to connect to the database:

```
GRANT CONNECT ON DATABASE database TO username;
```

Then, access to schemas in the database is granted with

```
GRANT USAGE ON SCHEMA schema-name TO username;
```

After that, permissions can be granted:

```
GRANT permission-list ON TABLE table-list TO username;
```

The `permission-list` is a list of the actions that a user is authorized to carry out in the database. These include `SELECT, INSERT, DELETE, UPDATE, CREATE, DROP, ALTER` (the last three refer to tables). To give someone blank permission to do anything, the keyword `ALL` is used.

The `table-list` is a list of tables in the schema; the expression

`ALL TABLES IN SCHEMA schema-name`

can be used to give permission in all tables of the given schema.

A common procedure in Postgres is to create a role, assign this role permissions using the `GRANT` commands shown, and then create individual users and assigning roles to users. Each user inherits all permissions from the role. The procedure looks like this:

```
CREATE ROLE rolename;

GRANT CONNECT ON DATABASE database_name TO rolename;

GRANT USAGE ON SCHEMA schema_name TO rolename;

#Or, to allow permission to create tables:
GRANT USAGE, CREATE ON SCHEMA schema_name TO rolename;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES
```

```
IN SCHEMA schema_name TO rolename;

CREATE USER username WITH PASSWORD 'user_password';
GRANT rolename TO username;
```

To view all users in a server, in the command line simply type `du`. This will produce a table with schema `(Role-name, attributes)`, that will display the permissions per role (command `du+` provides additional detail, while command `du username` provides information about a given user). This is actually a shortcut for an SQL command that looks into a table called `pg_catalog.pg_user`, which stores information about roles/users.

To change existing permissions on a user, use

`ALTER USER permission-list IN database`

where the `permission-list` can be `ALL` or a list of allowed actions (as above). To change the name of a user, the shortcut

`ALTER USER old-username RENAME TO new-username`

is provided. To take away some privileges, the `REVOKE` command is used:

`REVOKE permission-list ON database FROM username;`

To delete a user, simply use

`DROP USER username;`

If roles were used instead, one can drop the role instead:

`DROP ROLE rolename;`

In MySQL, the SQL command to create a user is

```
CREATE USER username IDENTIFIED BY password;
```

The `username` is sometimes written as `username@host`, where `host` indicates the computer from which the user is expected to connect to the database. For a database running in the same computer as the client, `localhost` is used.

Of course, there is also a `DROP USER username` to delete a user and `ALTER USER username` to make changes to a user. The most common changes are to change the username and/or the password or to adjust the permissions of a user.

The command to assign permissions to a user is

`GRANT permission-list ON database TO username`

The `permission-list` works as in Postgres. The `database` indicates on which databases the permissions are granted (since a server may run several databases); to give permissions in all the databases in a server, the '*' symbol can be used. If one wants to give permissions on only some of the tables of the database, dot notation (`database.tablename`) is used to indicate so.

Finally, `username` identifies the user to which the permissions are given.

The most important aspects to know about permissions are:

- permissions can be roughly divided into *read* and *write* permissions. Read permissions allow a user to 'see' things and are needed to run queries (i.e. the `SELECT` permission). It is common to create users with read permission over all or a part of the tables in a database, which will allow those users to run queries over such tables. Write permissions are needed to 'do' things, includ-

ing changing existing things (i.e. the `INSERT, DELETE, UPDATE, CREATE, DROP, ALTER`). Thus, write permissions are needed to insert, delete, or update data in tables and also to create, drop, or alter tables. Obviously, users who need to upload data into the database (or move it out) need to write permissions.

- a user may be given permission to create other users (a specific `CREATE USER` permission). In this case, a separate `GRANT OPTION` permission allows the user to pass some or all of its privileges to the newly created user. If user A is given the permission to create another user B, it is typically the case that A cannot give B permissions that are more powerful than its own. This prevents A from creating a user B with all-encompassing permissions and then (knowing B's username and password) log into the system as B and do things that it (A) was not allowed to do in the first place.

The original user is called a 'superuser' because it has permissions to do anything—otherwise, there would be something that no user could ever do, since the superuser is the origin of all other accounts. Some system administrators reserve the superuser account for themselves and make all other database users to be 'ordinary' users, with limited permissions.

To view all users in a server, type

```
SELECT User, Host FROM mysql.user;
```

To see what kind of permissions a user has, type

```
SHOW GRANTS FOR username;
```

To remove privileges, use the same `REVOKE` command as Postgres.

A shortcut to change a user's username is to use `RENAME`:

```
RENAME USER old-username TO new-username;
```

and to change the password

```
SET PASSWORD FOR username = PASSWORD(password);
```