The Australian National University, School of Computing
COMP2400/6240 (Relational Databases)
Semester 2, 2023

Lab 2

SQL Basics

In this lab, we will begin to use the database management system (DBMS) PostgreSQL. The documentation for PostgreSQL, including a great tutorial, can be found at `https://www.postgresql.org/docs/14/index.html`.

1 Command-Line Interface `psql`

The command-line interface for PostgreSQL works similarly to the UNIX shell, and is entered from the shell by typing the command `psql`. **Students may mix up commands for the shell and DBMS, try to always remember which one you are in. The prompts are different:**

- **The shell on your lab computer shows you**
  `u1234567@n11XltYZ:$`
- **The shell on the `partch` server shows you**
  `u1234567@partch:$`
- **The psql on `partch` or on the local computer shows you**
  `u1234567=>`
  Note: In case you see `u1234567->` instead of `u1234567=>`, this indicates an unfinished SQL command. Type a semicolon ';' to finish the command and press Enter to execute.

(1) *Start the PostgreSQL command-line interface by entering* `psql` *on the* `partch` *server.*

Open a command shell and log in to `partch` by typing ssh u123456@partch.anu.edu.au at the command line in the terminal window

```
u1234567@n11XltYZ:$ ssh u1234567@partch.anu.edu.au
```

Start the PostgreSQL interface by entering `psql` in your terminal.

```
u1234567@partch:$ psql
```

Please note that `psql` has also been installed locally on each lab computer, but you are recommended to use the `psql` installed on the `partch` server. The information (e.g., tables) stored through the local `psql` on each lab computer is not linked to your personal account and you can not retrieve it next time you login. The information stored through the `psql` on the `partch` server is linked to your personal account and you can retrieve that next time you login to the `partch` server (from the same or different lab computer, or access the `partch` server remotely from your personal computer).

There are two kinds of commands that the PostgreSQL command-line interface can handle: SQL commands and meta-commands. The meta-commands always begin with \\. Below are some of the most useful meta-commands in PostgreSQL.

`\?`	Help on psql commands
`\q`	Quit psql and return to the Unix shell
`\dt`	List the currently defined tables
`\d <table-name>`	Describe a table, listing its columns and datatypes
`\i <file-name>`	Input commands from a file

(2) *Type* `\dt` *to list the tables in the current database.*

At the moment, your database has no table.

(3) *Quit out of* `psql` *using* `\q` *and start it up again.*

You can also quit by pressing CTRL-d (hold the CTRL key down and press 'd' at the same time).

2 Data Definition Language

(4) *Enter the following SQL command, which will create a new table in your database.*

```
CREATE TABLE student (
    name varchar(20),
    email varchar(50)
);
```

Note: Don't forget the semicolon ';' at the end of each SQL command. In case that happens, you will see that the command prompt is changed from

“=>” to “->” to indicate an unfinished SQL command. Type a semicolon to finish the command and press Enter to execute.

(5) *Now, add one record to the new table using the following command.*

```
INSERT INTO student (name, email) VALUES
    ('Peter', 'peter@gmail.com');
```

Rather than typing everything through the command-line interface, it is often more convenient to prepare code in a file and run it from that.

(6) *Open a text editor on your lab computer, and save the following code into a file called* `insertStudents.sql` *and copy this file from your own computer to your directory for this week's lab on* `partch`*.*

```
INSERT INTO student (name, email) VALUES
    ('Aiden', 'aiden@hotmail.com');
INSERT INTO student (name, email) VALUES
    ('Emilia', 'emilia@yahoo.com');
INSERT INTO student (name, email) VALUES
    ('Ian', 'ian@github.com');
```

To run this .sql code, type `\i insertStudents.sql` within `psql`. If you get an error like

```
insertStudents.sql:  No such file or directory
```

then you are probably not in the directory where you saved the file. Exit postgreSQL using `\q`, then do a `pwd` to see where you are, and use `ls` to see whether the file `insertStudents.sql` is there. Find your way to the correct directory, or save the file where you should have saved it and try again.

You may also use the following query command to show all records in the table `student`.

(7) *Run this query.*

```
SELECT * FROM student;
```

(8) *Download the following two files from the Wattle course site (in the folder Lab 2: SQL Basics), and copy the files to the directory for this week's lab on* `partch`*.*

- `employeeCreate.sql`,
- `employeeQueries.sql`

(9) *Open* `employeeCreate.sql` *in an editor, and look through the code.*

Notice some language features such as `NOT NULL`.

(10) *Go and look at the PostgreSQL manual in your browser (https://www.postgresql.org/docs/14/index.html)*

Section II, "The SQL Language" is a great reference that you can use to find neat ways of doing things, and learn the meaning of pieces of code you do not understand. The `CREATE TABLE` command is covered in Section II.5, "Data Definition". Click on the link to Section II.5.4.2, "Not-Null Constraints", and read a little of the entry.

(11) *Use the PostgreSQL manual to find out the meaning of some other not-so-obvious constraints in* `employeeCreate.sql`*, such as: check constraints, primary keys and foreign keys.*

The file `employeeCreate.sql`, as you have seen, is written to create the database tables and add records into them. There are several `DROP TABLE` commands included at the beginning of `employeeCreate.sql`, which are used to remove the existing tables (if any) in our example database before creating the new tables.

(12) *Run* `employeeCreate.sql` *now by typing*

```
\i employeeCreate.sql
```

You should see a lot of messages saying `CREATE TABLE` and `INSERT 0 1`. Note that `INSERT 0 1` indicates that exactly one row is added to the table.

Now we have a database to play with.

PostgreSQL provides the `ALTER TABLE` command to change the definition of a table.

(13) *Consider the table* `project` *in our example database, how can you add an attribute* `StartDate` *to* `project`*?*

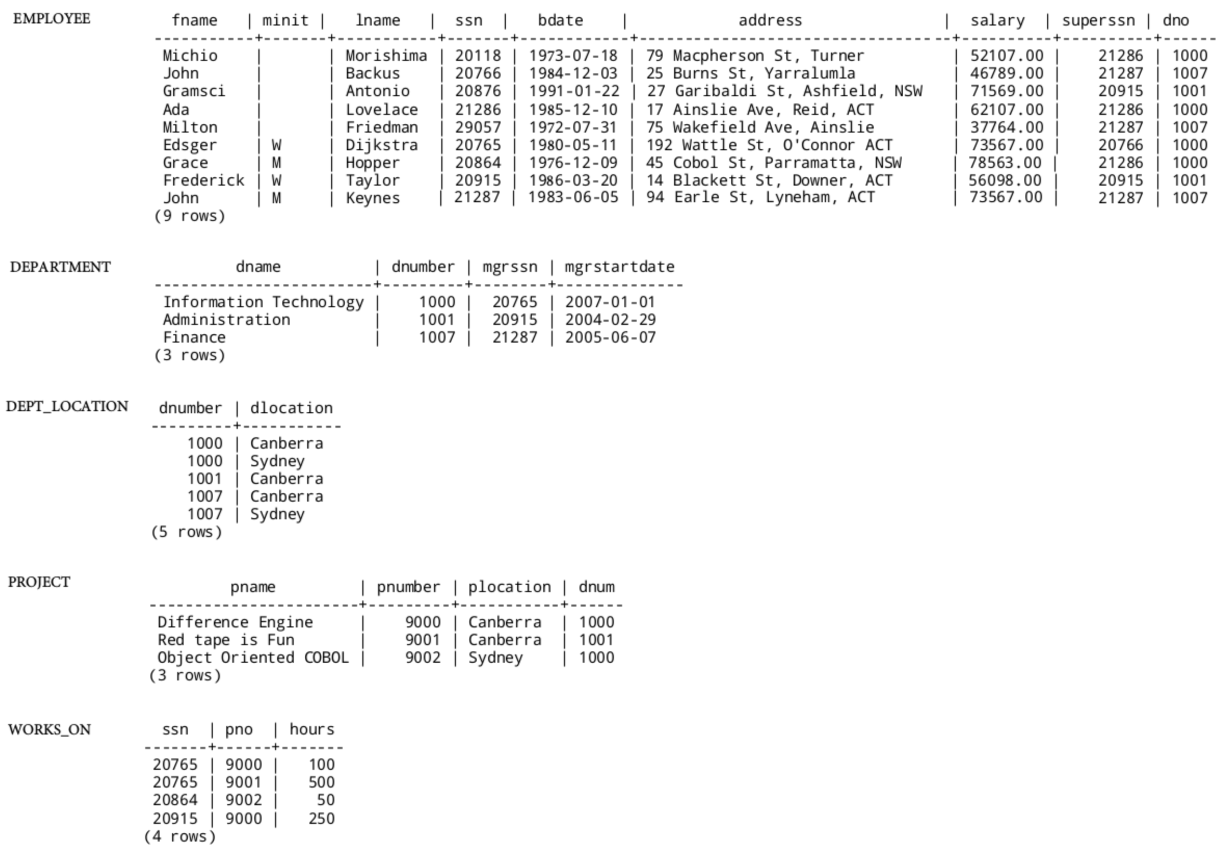
```
ALTER TABLE project ADD StartDate Date;
```

(14) *Consider the table* `department` *in the example database, how can you ensure that the values of the attribute* `mgrssn` *are valid, i.e., each of them must be a ssn in the table* `employee`*? (Hint: add a foreign key constraint on* `department`*)*

```
ALTER TABLE department ADD FOREIGN KEY (mgrssn) REFERENCES employee(ssn);
```

3 Simple Queries

The above example database state (created by `employeeCreate.sql`) is shown in the following figure. Here you may try to run some simple queries and please don't worry if you don't understand them right now.



```
EMPLOYEE
   fname   | minit |   lname   |  ssn  |   bdate    |              address              |  salary  | superssn | dno
-----------+-------+-----------+-------+------------+-----------------------------------+----------+----------+------
 Michio    |       | Morishima | 20118 | 1973-07-18 | 79 Macpherson St, Turner          | 52107.00 |    21286 | 1000
 John      |       | Backus    | 20766 | 1984-12-03 | 25 Burns St, Yarralumla           | 46789.00 |    21287 | 1007
 Gramsci   |       | Antonio   | 20876 | 1991-01-22 | 27 Garibaldi St, Ashfield, NSW    | 71569.00 |    20915 | 1001
 Ada       |       | Lovelace  | 21286 | 1985-12-10 | 17 Ainslie Ave, Reid, ACT         | 62107.00 |    21286 | 1000
 Milton    |       | Friedman  | 29057 | 1972-07-31 | 75 Wakefield Ave, Ainslie         | 37764.00 |    21287 | 1007
 Edsger    | W     | Dijkstra  | 20765 | 1980-05-11 | 192 Wattle St, O'Connor ACT       | 73567.00 |    20766 | 1000
 Grace     | M     | Hopper    | 20864 | 1976-12-09 | 45 Cobol St, Parramatta, NSW      | 78563.00 |    21286 | 1000
 Frederick | W     | Taylor    | 20915 | 1986-03-20 | 14 Blackett St, Downer, ACT       | 56098.00 |    20915 | 1001
 John      | M     | Keynes    | 21287 | 1983-06-05 | 94 Earle St, Lyneham, ACT         | 73567.00 |    21287 | 1007
(9 rows)

DEPARTMENT
         dname          | dnumber | mgrssn | mgrstartdate
------------------------+---------+--------+--------------
 Information Technology |    1000 |  20765 | 2007-01-01
 Administration         |    1001 |  20915 | 2004-02-29
 Finance                |    1007 |  21287 | 2005-06-07
(3 rows)

DEPT_LOCATION
 dnumber | dlocation
---------+-----------
    1000 | Canberra
    1000 | Sydney
    1001 | Canberra
    1007 | Canberra
    1007 | Sydney
(5 rows)

PROJECT
         pname         | pnumber | plocation | dnum
-----------------------+---------+-----------+------
 Difference Engine     |    9000 | Canberra  | 1000
 Red tape is Fun       |    9001 | Canberra  | 1001
 Object Oriented COBOL |    9002 | Sydney    | 1000
(3 rows)

WORKS_ON
  ssn  | pno  | hours
-------+------+-------
 20765 | 9000 |   100
 20765 | 9001 |   500
 20864 | 9002 |    50
 20915 | 9000 |   250
(4 rows)
```

(15) *Open* `employeeQueries.sql`*, and take a look at the queries there.*

(16) *For each query: work out what result you would expect it to yield against our example database. Write and save this result in a comment-block below the query. Then run it in* `psql` *by copying and pasting into the terminal. Compare actual and expected results, and get an idea of simple SQL queries.*

(17) *Refer to the online resources to learn PostgreSQL* ***https://www.postgresql.org/docs/online-resources/***