These are the commands for the follow-along exercises in Lesson 2.1.

2.104 Installing Postgres

commands

If you find when running the apt-get commands that postgres is installed then you can move on from there.

```
# apt-get install postgresql postgresql-client postgresql-contrib
# postgresql-server-dev-all
```

There are 2 possible outcomes. Firstly the packages install as in the video. Secondly the packages are already installed. If the output says:

```
postgresql is already the newest version (10+190ubuntu0.1).
postgresql-client is already the newest version (10+190ubuntu0.1).
postgresql-contrib is already the newest version
(10+190ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

This indicates that postgres is already installed. In either case you can move on to the next commands

```
# chmod uog+rw /var/run/postgresql/
# su coder
```

```
# /usr/lib/postgresql/10/bin/initdb -D /home/coder/project/django_databases/
```
YOU NO LONGER NEEDS THIS COMMAND AS THE LAB DOES THIS FOR YOU

```
# /usr/lib/postgresql/10/bin/postgres -D ~/project/django_databases >
~/project/django_databases/logfile 2>&1 &
# /usr/lib/postgresql/10/bin/psql -h localhost -d postgres
```

2_105 Building a simple database with SQL

commands

```
# su coder
# psql -d postgres
```

SQL

```
# CREATE DATABASE example;
# \list
# \c example
# CREATE TABLE people (pk SERIAL PRIMARY KEY,
 name VARCHAR(256) NOT NULL,
 height_cm INT,
 gender VARCHAR(256),
 date_of_birth DATE);
# CREATE TABLE address (pk SERIAL PRIMARY KEY,
 house_number INT,
 street_name VARCHAR(256),
 city VARCHAR(256),
 country VARCHAR(256));
```

```
# \d
# \d people
# ALTER TABLE address ADD people_pk INT;
# ALTER TABLE address ADD FOREIGN KEY (people_pk) REFERENCES people(pk);
# CREATE TABLE cars (pk SERIAL PRIMARY KEY,
engine_size_cc INT,
colour VARCHAR(256),
manufacturer VARCHAR(256),
model VARCHAR(256),
year INT,
people_pk INT,
FOREIGN KEY (people_pk)
REFERENCES people(pk));
# CREATE TABLE pets (pk SERIAL PRIMARY KEY,
species VARCHAR(256),
coat VARCHAR(256),
age INT,
people_pk INT,
FOREIGN KEY (people_pk) REFERENCES people(pk));
\d
# INSERT INTO people (name, height_cm, gender, date_of_birth) VALUES ('Ben',
167, 'Male', '1978-03-14');
# select * from people;
# INSERT INTO pets (species, coat, age, people_pk) VALUES ('cat', 'brown', 4,
1);
```

2.106 Video lecture: Relational modelling limitations

SQL

```
# CREATE TABLE csv_data (pk SERIAL PRIMARY KEY, id VARCHAR(256), source
VARCHAR(256), type VARCHAR(256), start INT, stop INT, score FLOTA, strand CHAR,
phase CHAR, attributes TEXT)
# CREATE TABLE three_node (pk SERIAL PRIMARY KEY, id VARCHAR(256) NOT NULL,
first_connection_pk INT, second_connection_pk INT)
# CREATE TABLE four_node (pk SERIAL PRIMARY KEY, id VARCHAR(256) NOT NULL, name
VARCHAR(256)m first_connection_pk INT, second_connection_pk INT,
third_connection_pk INT)
# CREATE TABLE people (pk SERIAL PRIMARY KEY, id VARCHAR(256) NOT NULL)
# CREATE TABLE connections (pair_a INT, pair_b INT)
# CREATE TABLE tree (pk SERIAL PRIMARY KEY, name VARCHAR(256) NOT NULL,
parent_one_pk INT, parent_two_pk INT);
```

2.107 Video lecture: Database good practice

SQL

```
# CREATE DATABASE gene_test;

# CREATE TABLE ec(pk SERIAL PRIMARY KEY, EC_name VARCHAR(256));

# CREATE TABLE sequencing(pk SERIAL PRIMARY KEY,
sequencing_factory VARCHAR(256), factory_location VARCHAR(256));

# CREATE TABLE genes (pk SERIAL PRIMARY KEY, gene_id VARCHAR(256)
NOT NULL, entity VARCHAR(256), source VARCHAR(256), start INT,
stop INT, sequencing_pk INT, ec_pk INT, FOREIGN KEY
(sequencing_pk) REFERENCES sequencing(pk), FOREIGN KEY (ec_pk)
REFERENCES ec(pk));
```

```
# CREATE TABLE products (genes_pk INT, type VARCHAR(256), product
VARCHAR(256), FOREIGN KEY (genes_pk) REFERENCES genes(pk));

# CREATE TABLE attributes (pk SERIAL PRIMARY KEY, key
VARCHAR(256), value VARCHAR(256));

# CREATE TABLE gene_attribute_link(genes_pk INT, attributes_pk
INT, FOREIGN KEY (genes_pk) REFERENCES genes(pk), FOREIGN KEY
(attributes_pk) REFERENCES attributes(pk));
```