University Of Portsmouth
BSc (Hons) Computer Science
First Year

Database Systems Development

M30232
September 2022 - May 2023
20 Credits

Thomas Boxall
up2108121@myport.ac.uk

Contents

LECTURE: Introduction to module (29-09-22) 2

PRACTICAL: Introduction to Practicals (29-09-22) 5

LECTURE: The Database Environment (06-10-22) 9

PRACTICAL: Further Introduction (06-10-22) 12

LECTURE: Database Concepts (13-10-22) 18

PRACTICAL: Count() (13-10-22) 20

LECTURE: Coursework & Entity Relationship Diagrams (20-10-22) 27

PRACTICAL: SQL and Entities (20-10-22) 29

LECTURE: ERD, Attributes & Datatypes (27-10-22) 33

LECTURE: Normalisation (10-11-22) 36

PRACTICAL: Keys & Joins (10-11-22) 37

LECTURE: Joins and Narrowing Focus (17-11-22) 44

PRACTICAL: Normalisations and Joins (24-11-22) 46

LECTURE: Types of Joins (01-12-22) 51

PRACTICAL: further joins (01-12-22) 52

LECTURE Security Basics I (08-12-22) 58

PRACTICAL: More Joins (08-12-22) 60

LECTUER: Christmas Lecture (15-12-22) 66

LECTURE: Introduction to module

29-09-22 13:00 Mark RB LT1

The Module Coordinator for this module is Mark (based in BK 3.09), he's assisted by Valentin and Roy in some sessions alongside others too.
Mark is using a new piece of software to make his presentations with, this is currently in the test phases and he may change back to PowerPoint if people don't like it. Slides are available on Moodle as HTML format, they can be printed to PDF files for offline viewing.

Module Aims

This module aims to help you understand where the database sits in modern systems. It does not train us to be database administrators. It gives us the skills to design a database and the knowledge of how to access it and do so safely.
This module will start from the ground up.

Learning Outcomes

- Demonstrate the fundamental principles of database design & development
- Use appropriate analysis techniques to identify the requirements of a database.
- Design and build a relational database, given a set of requirements.
- Understand how to apply data manipulation using SQL.

Historically, this module used to focus on the elements of Computer Science which relate to databases (for example, software development lifecycles). Now, it focuses on just databases.

Content Overview

This module provides an understanding of the theory of relational database design using tools standard to the industry. We will be taught how to design databases using Crows Foot Entity Relationship Diagrams and SQL to create the database. This module will also cover normalisation.

Teaching Overview

The module is a year long, worth 20 credits and has two different styles of teaching.
There will be one, one hour lecture per week. In this session, we will be taught the knowledge which we can put into practice in the following weeks practical session.
There will be one, one and a half hour practical session per week. In this session, we will practice the skills required for databases. *(N.B. This session is timetabled for two hours on the timetable, generally the lecturers will leave after an hour and a half however students can remain in the room until the end of the two hours.)*
If you are unable to make it to a lecture, you need to read the content provided on Moodle. If you are unable to make it to a practical, you need to read and do (most importantly, do) the content on Moodle; this is so you are able to complete the following practical as they all build on each other.

Resources

There are a number of resources talked through:

- Moodle - the universities Virtual Learning Environment. Notes from lectures and from practicals will be uploaded here along with quizzes and other resources.
- Google Virtual Machine - the virtual machine in which our database lives. You do not need the university VPN to access it, as it requires a SSH connection. The data is hosted by Google, the module staff have some control over the machines. More detail on this will be provided in the first practical session.
- Google Workspace
- Microsoft Office. This is available free from the university. At some point, this will include Microsoft Visio, which is useful for coursework.

Expectations

Lecturers Expectations of Students

- Turn up for lectures (from next week, the content taught in the lectures will be used in the following weeks practical sessions)
- Arrive on time (there is usually useful information given out at the beginning of sessions)
- Participate and take notes in sessions
- Catch up on sessions if you miss them
- Finish the practical work before the following weeks practical sessions
- Study for about 4 hours a week total

These things are proven to increase the likelihood that a student gets a better mark at the end of the year.

Students Expectations of Lecturers

They are nice to students; start and end sessions on time; provide students with support and feedback on work throughout the module; and to return feedback and marks on work as quickly as they can (this usually should be within two weeks).

Assessments

There are two forms of assessment in this module.

Coursework

This will be worth 50% of the overall module mark. It will be released in the next few weeks and will be due at the end of the first week after the January assessment period (probably the Friday of that week at 11pm). The content assessed will all be from the first teaching block. We will get extra marks if we include content which hasn't been taught yet.

Exam

This will be worth 50% of the overall module mark. It will take place in the May/June assessment period and be computer based. It can include anything from the entire year however we won't have to write code (probably will have to look at code and say whats wrong). It will be multiple choice questions. There will be quizzes available on Moodle which will be similar to this where we can practice.

Brief Introduction to Databases

> **Database**
>
> "A single, possiblely large, repository of data that can be used simultaneuously by many departments and users" *(Database Solutions: A Step by Step Guide to Databases - T Connolly & C Begg)*

Spreadsheets

Spreadsheets are not databases. This is because a spreadsheet cannot hold the amount of data which a database can and eventhough though using some software, a database could be shared with multiple people, it cannot be edited by multiple people simultaneously.
This also applies to Microsoft Access.

Database Management System (DBMS)

> **DBMS**
>
> "The software which interacts with the users' application programs and the database" *(Database Solutions: A Step by Step Guide to Databases - T Connolly & C Begg)*

Examples of a DBMS include PostgreSQL, MySQL, SQL Server, Oracle and Mongo DB.

Why Use a Database

An alternative to databases are file based systems.
File based systems: are old fashioned; are not necessarily digital; they often contain duplicate data; are difficult to search; are very difficult to update; have the possibility to contain different file types which may not be compatible together; are inaccessible; and security may be an issue.
A database is: a modern approach; digital; duplicates can be removed; easy to search; easy to update; comprised of only one file type; capable of having multiple levels of access control; able to limit user access.
There are times at which a Database is not suitable for the setting. In this case, it may be more suitable to use a spreadsheet.

Integrated Database Environment

In an integrated database environment, the DBMS sites as a communication hub between all nodes. The DBMS is the server on which the database is hosted.
When the database is setup correctly, you can get more information out of it than you put in.

PRACTICAL: Introduction to Practicals

29-09-22 14:00 Mark & team FTC 3rd floor

Introduction to Practical sessions

Practical documents are available on Moodle, make a copy of these and store within your university Google Drive so you can edit them during the sessions and make notes.

Access Levels

In PostgreSQL, the first level of security is that a user cannot login unless they have been given access or there is a database with the same name as their username.
We don't have `sudo` access to linux, however we have full administrative access to PostgreSQL. Don't drop the database called `upxxxxxxx` (where `xxxxxxx` is replaced with student number) or anything that is owned by `postgres` as this breaks things.

PostgreSQL

PostgreSQL is ready to accept code when the prompt ends in `=#`. If you enter part of a command and press enter, the prompt will change to `-#`, this indicates that Postgres is waiting for you to finish the command.
PostgreSQL gives some useful error messages, SQL does not.

Code Editors

A code editor should be used to write SQL into, then the SQL should be copied and pasted into the Linux machine. The only thing that should be directly entered into the shell is to connect to a different database.
This is so that a. we have a copy of what we have done and b. so that if the VM is deleted, we are able to re-build our VM with less pain than if we didn't save all the code.
A recommended setup is to use VS code, with a SQL syntax extension. VS Code comes with integrated Powershell, allowing you to ssh to the VM from the same window.

SQL

SQL works like a procedural programming language, in that it reads the code inputted line by line. This also means that long and complex lines of code can be split across many lines, making it easier to read them.

Installing The First Database

Due to an issue with the image used to build the Virtual Machines, we have to create the database which we will use for the first few sessions. The code to do this was available on Moodle, copy and paste into the code editor then copy and paste again, this time into the Postgres prompt of the linux machine. This executes and creates the database, pre-populated with some sample data.

Tasks

1. List the databases in your server

LANGUAGE: SQL

```
\l
```

LANGUAGE: Unknown

```
                                 List of databases
    Name      |     Owner      | Encoding | Collate |  Ctype  |   Access privileges
--------------+----------------+----------+---------+---------+-----------------------
dsd_22        | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
mongo-2021-fix | mongo-2021-fix | UTF8     | C.UTF-8 | C.UTF-8 |
postgres      | postgres       | UTF8     | C.UTF-8 | C.UTF-8 |
template0     | postgres       | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
              |                |          |         |         | postgres=CTc/postgres
template1     | postgres       | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
              |                |          |         |         | postgres=CTc/postgres
up2108121     | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
(6 rows)
```

2. Connect to the database

LANGUAGE: SQL

```
\c dsd_22
```

LANGUAGE: Unknown

```
You are now connected to database "dsd_22" as user "up2108121".
```

3. List everything in this database

LANGUAGE: SQL

```
\d
```

LANGUAGE: Unknown

```
                    List of relations
Schema |            Name             |   Type   |   Owner
--------+-----------------------------+----------+-----------
public | category                    | table    | up2108121
public | category_cat_id_seq         | sequence | up2108121
public | cust_order                  | table    | up2108121
public | cust_order_cust_ord_id_seq  | sequence | up2108121
public | customer                    | table    | up2108121
public | customer_cust_id_seq        | sequence | up2108121
public | manifest                    | table    | up2108121
public | manifest_manifest_id_seq    | sequence | up2108121
public | product                     | table    | up2108121
public | product_prod_id_seq         | sequence | up2108121
public | role                        | table    | up2108121
public | role_role_id_seq            | sequence | up2108121
public | staff                       | table    | up2108121
public | staff_staff_id_seq          | sequence | up2108121
(14 rows)
```

4. List just the tables

LANGUAGE: SQL

```
\dt
```

LANGUAGE: Unknown

```
           List of relations
Schema |    Name     | Type  |   Owner
--------+-------------+-------+-----------
public | category    | table | up2108121
public | cust_order  | table | up2108121
public | customer    | table | up2108121
public | manifest    | table | up2108121
public | product     | table | up2108121
public | role        | table | up2108121
public | staff       | table | up2108121
(7 rows)
```

5. Get a list of all the customers in the `customer` table

LANGUAGE: SQL

```
SELECT * FROM customer;
```

LANGUAGE: Unknown

```
cust_id |   cust_fname    |    cust_lname    |        addr1          |    addr2     |      town
    ↪      | postcode  |              email
---------+-----------------+------------------+-----------------------+--------------+
    ↪ ---------------+-----------+---------------------------------
        1 | Jobey           | Boeter           | 6 Claremont Park      | Truax        | La
    ↪ Mohammedia  | CV42 3EF  | jboeter0@mail.ru
        2 | York            | O'Deegan         | 882 Hooker Trail      |              | Chemnitz
    ↪        | YA92 2OJ  | yodeegan1@nydailynews.com
        3 | Penelope        | Hexter           | 25 Jackson Lane       |              | Pingshan
    ↪        | LY32 8LN  | phexter2@cbslocal.com
        4 | Chadd           | Franz-Schoninger | 7 Division Point      | Texas        | Baojia
    ↪        | XA22 0UR  | cfranzschoninger3@google.com.hk
        5 | Vikky           | Eke              | 293 Colorado Drive    | Browning     | Kamenny
    ↪ Privoz | WQ12 3SF  | veke4@elegantthemes.com
        6 | Marie-francoise | Currier          | 032 Eagan Junction    | Duke         |
    ↪ Waekolong      | NB52 4MV  | acurrier0@economist.com
        7 | Benedicte       | Dozdill          | 579 Dryden Terrace    |              | Dawuhan
    ↪        | GY32 6GQ  | cdozdill1@amazon.de
        8 | Gorel           | Douthwaite       | 2946 Bluejay Parkway  | Heath        | Sunbu
    ↪        | PH02 3ZX  | edouthwaite2@feedburner.com
        9 | Berengere       | Menendez         | 06154 Jackson Way     | Doe Crossing |
    ↪ Tsagaanders    | HO82 5XL  | amenendez3@dell.com
        10 | Pelagie         | Hachard          | 1777 Hauk Center      |              | Jiantou
    ↪         | NA52 4LM  | fhachard4@blinklist.com
        11 | Adaobi          | Musa             | 6 Clariss Ave         |              | La
    ↪ Mohammedia  | CV4 3F    | amusa9@mail.ca
(11 rows)
```

6. Choose a different table from the output of `\dt` and get a list of all the records in that table.

LANGUAGE: SQL

```
SELECT * FROM role;
```

LANGUAGE: Unknown

```
role_id |    role_name
---------+-----------------
```

```
   1 | Order Picker
    2 | Final Packer
    3 | Post Sales
    4 | Customer Retain
    5 | Misc
(5 rows)
```

LECTURE: The Database Environment

06-10-22 13:00 Mark RB LT1

Data or Information

When we think about real world things, we will generally think of these in terms of information, not data. Everyone and everything has information. We have to break information down into data to be able to store it.

> **Data**
>
> Facts and statistics collected together for reference or analysis *(https://en.oxforddictionaries.com/definition/data)*

> **Information**
>
> The result of applying data processing to data, giving it context and meaning. Information can then be further processed to yield knowledge *(http://foldoc.org/information)*

When we need to store information in a database, we first have to break it down into data items. These can be entered into the database then pulled out again in different states. When done right, these different states should be able to tell us more information than we put in.
We also have knowledge, this is the ability to find things.

Processing Data

If we are given random data items, we can assume what they mean. For example, if we are given `1.99`; `cheeseburger`; and `Bob's Midnight Burgers`, you could assume that you could purchase a cheeseburger from an establishment called Bob's Midnight Burger for £1.99. However, this might be completely wrong! It could in fact be three un-related pieces of information or we may have mis-interpreted the information completely. This shows that it is imperative we look at the context which surrounds data, before drawing information from it.

Database Management System

The Database Management System (DBMS) is the core of the database system. Every communication to the database is done through the DBMS, this includes queries, data in and data out. The DBMS also controls access to the data and schema (which is stored within the database itself).

> **Schema**
>
> The 'blueprint' of the database.

An advantage of using a DBMS is that different users can be restricted as to what they can access; the data can easily be managed and the DBMS provides an integrated view of an enterprise's operations. The DBMS also removes the risk of inconsistent data and improves the ease with security can be controlled.

Database Languages

There are two different types of database languages (DDL and DML), each have a different purpose. SQL is both.
Before we look at DDL and DML in more detail, we first need to understand what the term 'Query' means.

Queries

A query is the code which interacts with the database.
This can be to read the contents of the database, you can 'query the database'. However it is also the code that puts the data into the database and the code which is used to build the database in the first place.

DDL

Data Definition Language (DDL) allows the DBA or users to describe and name entities, attributes and relationships required for the applications that access it and associated integrity and security constraints. It is the set of commands which are used to define the structure of the database. These are the commands used to create, modify or remove database objects (e.g., tables, users and indexes). Listed below are a number of the most commonly used DDL commands.

LANGUAGE: SQL

```sql
CREATE DATABASE
CREATE TABLE
ALTER TABLE
DROP DATABASE
DROP TABLE
RENAME TABLE
```

The following is an example of SQL code which creates a new table and as part of that defines the attributes within it.

LANGUAGE: SQL

```sql
create table property_for_rent (
    Property_id varchar(4) PRIMARY KEY,
    Street varchar(14) not null,
    City varchar(10) not null,
    Postcode varchar(10) not null,
    Type varchar(6) not null,
    Rooms integer not null,
    Rent decimal(6,2) not null,
    Owner_id varchar(4) not null REFERENCES private_owner(owner_id),
    Staff_id varchar(4) REFERENCES staff(Staff_id),
    branch_id varchar(4) REFERENCES branch(Branch_id)
);
```

DML

Data Manipulation Language (DML) provides the ability to manipulate data within the database. Its commands are used to select, insert, update and delete data items within a database. Listed below are a number of the most commonly used DML commands.
When selecting attributes to display, do not use `SELECT * FROM ...` as this selects everything. Instead, use `SELECT attribute, anotherAttribute, yetAnotherAttribute FROM ...`.
Take care when entering commands, for the configuration of our Virtual Machines, we are super users within PostgreSQL. Whatever we enter will be executed without question by the machine, this includes dropping data.

LANGUAGE: SQL

```sql
DELETE
INSERT
REPLACE
SELECT
UPDATE
```

The following is an example of SQL code which queries a table based on an attribute.

LANGUAGE: SQL

```sql
select property_id,
street,
city,
postcode,
owner_id from property_for_rent
where city = 'Glasgow';
```

PRACTICAL: Further Introduction

06-10-22 14:00 Mark & team FTC Floor 3

Introductory Tasks

1. After getting into PostgreSQL client, list the databases.

LANGUAGE: SQL

```
\l
```

LANGUAGE: Unknown

```
    List of databases
    Name      |     Owner      | Encoding | Collate |  Ctype  |   Access privileges
---------------+----------------+----------+---------+---------+-----------------------
dsd_22         | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
mongo-2021-fix | mongo-2021-fix | UTF8     | C.UTF-8 | C.UTF-8 |
postgres       | postgres       | UTF8     | C.UTF-8 | C.UTF-8 |
template0      | postgres       | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
               |                |          |         |         | postgres=CTc/postgres
template1      | postgres       | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
               |                |          |         |         | postgres=CTc/postgres
up2108121      | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
(6 rows)
```

2. Connect to the `dsd_22` database.

LANGUAGE: SQL

```
\c dsd_22
```

LANGUAGE: Unknown

```
You are now connected to database "dsd_22" as user "up2108121".
```

3. List the contents of the database

LANGUAGE: SQL

```
\d
```

LANGUAGE: Unknown

```
                    List of relations
Schema |            Name             |   Type   |   Owner
--------+-----------------------------+----------+-----------
public | category                    | table    | up2108121
public | category_cat_id_seq         | sequence | up2108121
public | cust_order                  | table    | up2108121
public | cust_order_cust_ord_id_seq  | sequence | up2108121
public | customer                    | table    | up2108121
public | customer_cust_id_seq        | sequence | up2108121
public | manifest                    | table    | up2108121
public | manifest_manifest_id_seq    | sequence | up2108121
public | product                     | table    | up2108121
public | product_prod_id_seq         | sequence | up2108121
public | role                        | table    | up2108121
```

```
public | role_role_id_seq           | sequence | up2108121
public | staff                     | table    | up2108121
public | staff_staff_id_seq        | sequence | up2108121
(14 rows)
```

5. List just the tables

LANGUAGE: SQL

```
\dt
```

LANGUAGE: Unknown

```
            List of relations
Schema |    Name    | Type  |   Owner
--------+------------+-------+-----------
public | category   | table | up2108121
public | cust_order | table | up2108121
public | customer   | table | up2108121
public | manifest   | table | up2108121
public | product    | table | up2108121
public | role       | table | up2108121
public | staff      | table | up2108121
(7 rows)
```

The `\dt` command removes the sequences (which will be discussed further in a couple of weeks time).
6. Look at the structure of the role table.

LANGUAGE: SQL

```
\d role
```

LANGUAGE: Unknown

```
Table "public.role"
Column   |         Type          | Collation | Nullable |                Default
----------+-----------------------+-----------+----------+---------------------------------------
    ↪
role_id   | integer               |           | not null | nextval('role_role_id_seq'::regclass
    ↪ )
role_name | character varying(20) |           |          |
Indexes:
    "role_pkey" PRIMARY KEY, btree (role_id)
Referenced by:
    TABLE "staff" CONSTRAINT "staff_role_fkey" FOREIGN KEY (role) REFERENCES role(role_id)
```

Using SQL To Access Data

Most of the commands used so far are PostgreSQL specific commands (these are the ones which begin with \).
If the output from a command is too long, PostgreSQL will show a colon (`:`) at the bottom of the screen. To show the next screen, press the space bar. Once all the records have been seen, the screen will show (`END`). At this point, hit `q` to exit back to the prompt. `q` can also be pressed at the colon to exit back to the prompt from there too.
1. Read all the records in the `dsd_22` table `category`.

LANGUAGE: SQL

```
SELECT * FROM CATEGORY;
```

LANGUAGE: Unknown

```
cat_id |  cat_name
--------+-------------
     1 | Men's Wear
     2 | Ladies Wear
     3 | Kid's Wear
     4 | Outdoor
     5 | Sport
     6 | Health
(6 rows)
```

2. Run the following command and see if the output is different.

LANGUAGE: SQL

```
select * from category;
```

LANGUAGE: Unknown

```
cat_id |  cat_name
--------+-------------
     1 | Men's Wear
     2 | Ladies Wear
     3 | Kid's Wear
     4 | Outdoor
     5 | Sport
     6 | Health
(6 rows)
```

3. Run the following command, and see if the output is different.

LANGUAGE: SQL

```
select * from 'Category';
```

LANGUAGE: Unknown

```
ERROR:  syntax error at or near "'Category'"
LINE 1: select * from 'Category';
                      ^
```

4. Run the following command and see if the output is different.

LANGUAGE: SQL

```
select * from "Category";
```

LANGUAGE: Unknown

```
ERROR:  relation "Category" does not exist
LINE 1: select * from "Category";
                      ^
```

5. Run the following command and see if the output is different.

LANGUAGE: SQL

```
select * from 'category';
```

```
LANGUAGE: Unknown
ERROR:  syntax error at or near "'category'"
LINE 1: select * from 'category';
                      ^
```

6. Run the following command and see if the output is different.

```
LANGUAGE: SQL
select * from "category";
```

```
LANGUAGE: Unknown
cat_id |  cat_name
--------+-------------
     1 | Men's Wear
     2 | Ladies Wear
     3 | Kid's Wear
     4 | Outdoor
     5 | Sport
     6 | Health
(6 rows)
```

7. Run the `\dt` command again, look at the case of the table names.
8. Run the following command (nb, this is supposed to contain non-standard quote marks as copied from the Google Doc).

```
LANGUAGE: SQL
SELECT * FROM "category";
```

```
LANGUAGE: Unknown
ERROR:  relation ""category"" does not exist
LINE 1: SELECT * FROM "category";
```

From these exercises, it is clear that case doesn't matter when the table name is not in quotes; and that the type of quotes used matter (there are extensions available for Google Docs which allow code to be stored in them and for it to keep its formatting).

Table Structure

To see how tables are linked together, it is possible to view the table structures. This information tells you how the attributes are linked together and what the data types and sizes of said data types are (where this is applicable).
1. Run the following command.

```
LANGUAGE: SQL
\d customer
```

```
LANGUAGE: Unknown
    Table "public.customer"
    Column   |          Type          | Collation | Nullable |                  Default
```

```
    ↪ ------------+-----------------------+-----------+----------+-------------------------------------
    ↪
 cust_id    | integer                |           | not null | nextval('customer_cust_id_seq'::
   ↪ regclass)
 cust_fname | character varying(25)  |           | not null |
 cust_lname | character varying(35)  |           | not null |
 addr1      | character varying(50)  |           | not null |
 addr2      | character varying(50)  |           |          |
 town       | character varying(60)  |           | not null |
 postcode   | character(9)           |           | not null |
 email      | character varying(255) |           | not null |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (cust_id)
Referenced by:
    TABLE "cust_order" CONSTRAINT "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES
   ↪ customer(cust_id)
```

From the output, we can see that the data type of `cust_id` is integer and the data type of `postcode` is a fixed 9 length character.

Creating new Tables in SQL

The syntax for creating a table (or relation, if we're being proper) is shown below.

LANGUAGE: SQL

```sql
CREATE TABLE tableName(
attributeName dataType (options),
attributeName dataType (options),
...
);
```

Task

1. Create a new database with a name of your choice.
2. Connect to the database.
3. Create a new table with two attributes (one of data type INT, that is also the primary key and one that has a data type of your own choosing).

LANGUAGE: SQL

```sql
CREATE DATABASE week02;

CREATE TABLE NEWTABLE(
IAMNUMBER INT PRIMARY KEY,
IAMSTRING VARCHAR(10)
);
```

Now, insert a record into the table.

LANGUAGE: SQL

```sql
INSERT INTO NEWTABLE (IAMNUMBER, IAMSTRING) VALUES(12, 'cheese');
```

Now, insert another new record into the table, using the same INT value as the first record. Take note of the message which is displayed.

LANGUAGE: SQL

```sql
INSERT INTO NEWTABLE (IAMNUMBER, IAMSTRING) VALUES(12, 'ham');
```

LANGUAGE: Unknown

```
ERROR:  duplicate key value violates unique constraint "newtable_pkey"
DETAIL:  Key (iamnumber)=(12) already exists.
```

LECTURE: Database Concepts

13-10-22 13:00 Mark RB LT1

Despite the fact that the relational database model was designed by Codd in the 1970s, it is a valid system and used widely.

Key Terms

Database Term	Description
Entity	An object or a 'thing' about which data is stored.
Attributes	Some quality associated with the entity (eg ID number, username, size). These have data types (eg number, string etc) and maximum sizes. Other terms are elements and properties.
Relation	A two dimensional representation (table) of entities and/ or relationships. Other terms used are relation table or table.
Entity Set	A set of entities of the same type.
Relationship	How two relations (tables) are related to each other. Relationships are represented in relations.
Tuple	Corresponds to rows of the table or records of a relation. Other terms used are record and row.
Domain	A pool of all legal values from which actual attribute values are drawn.
Primary Key	An attribute or combination of attributes for which values uniquely identify tuples in the relation. The primary key is chosen from a set of candidate keys. If you have a numeric value which the system can generate, let it do it for you.
Candidate Key	There may be more than one potential primary keys for a relation. Each is called a candidate key or super-key.
Alternate Key	An alternate access path to data that is not via the primary key.
Composite Key	A combination of attributes that act as a candidate key in a relation. Each participating attribute in the composite key (also known as candidate key) is called a simple key.
Foreign Key	An attribute (or combination of attributes) that is a primary key in another relation. They can appear many times.
Degree	Number of attributes in a relation; also called the arity.

When designing a database, the first thing you need to think about is what entities do you need to store information about. Then think about the attributes which you need to store about each entity. Then create relations. At this point, think about the domain for any of the attributes (for example, month 1-12 or day 0-6 (Sunday to Saturday) or hours 0-23). Now think about keys.

Entity

An entity is a thing, it could be a person or a specific type of person.
To identify entities, look at the information given to you and identify the nouns. The nouns give an idea of what the entities look like but they require fine tuning.
There can be as many entities as needed.
We can describe entities using their attributes.
We now think about keys.

Primary Key

To identify what will be a primary key, we look for something that is unique. This should be something which cannot be changed. If there is nothing suitable, create your own primary key.

Foreign key

Does not have to be primary key in other table, however it has to be unique within the other table.

PRACTICAL: COUNT()

13-10-22 14:00 Mark and team FTC Floor 3

Q1. using the `count()` function demonstrated by your tutor, how many records are there in each of the tables in the dsd_22 database. (Remember to use `\dt` to give you a list of tables in the database.) Copy the outputs below.

LANGUAGE: Unknown

```
dsd_22=# select count(*) from category;
count
-------
    6
(1 row)
dsd_22=# select count(*) from cust_order;
count
-------
150
(1 row)
dsd_22=# select count(*) from customer;
count
-------
11
(1 row)
dsd_22=# select count(*) from manifest;
count
-------
150
(1 row)

dsd_22=# select count(*) from product;
count
-------
100
(1 row)

dsd_22=# select count(*) from role;
count
-------
    5
(1 row)

dsd_22=# select count(*) from staff;
count
-------
10
(1 row)
```

Q2. Use the max() function to find the highest value of the role_id attribute in the role table. Copy the output below

LANGUAGE: SQL

```
select max(role_id) from role;
```

LANGUAGE: Unknown

```
max
-----
    5
(1 row)
```

Q3. Insert a new row of data into the role table with

LANGUAGE: SQL

```sql
INSERT INTO ROLE (ROLE_NAME) VALUES ('Pre Sales');
```

Q4. How many rows of data are now in the role table? Copy it below.

LANGUAGE: SQL

```sql
select count(*) from role;
```

LANGUAGE: Unknown

```
count
-------
    6
(1 row)
```

Q5. What is the maximum value of the role_id now? Copy it below.

LANGUAGE: SQL

```sql
select max(role_id) from role;
```

LANGUAGE: Unknown

```
max
-----
    6
(1 row)
```

Q6. Delete this new row with

LANGUAGE: SQL

```sql
DELETE FROM ROLE WHERE ROLE_NAME = 'Pre Sales';
```

LANGUAGE: Unknown

```
DELETE 1
```

Q7. How many rows of data are now in the role table? Copy it below.

LANGUAGE: Unknown

```
count
-------
      5
(1 row)
```

Q8. What is the maximum value of the role_id now? Copy it below.

LANGUAGE: Unknown

```
max
-----
    5
(1 row)
```

Q9. Reinsert the row of data into the role table again with

```sql
INSERT INTO ROLE (ROLE_NAME) VALUES ('Cleaning Team');
```

```
INSERT 0 1
```

Q10. How many rows of data are now in the role table? Copy it below.

```
count
-------
        6
(1 row)
```

Q11. What is the maximum value of the role_id now? Copy it below.

```
max
-----
    6
(1 row)
```

Q12. Create a random value using the random function. Copy the value below

```sql
SELECT RANDOM();
```

```
      random
-------------------
 0.175315219908953
(1 row)
```

Q12a. Create another random number. Copy the value below

```sql
SELECT RANDOM();
```

```
      random
-------------------
 0.272884896956384
(1 row)
```

Q13. Create one more random value but now multiply it by 11. Remember that to multiply you do not use x but use the * symbol. Run this code 5 times and copy the values below.

```
dsd_22=# select random()*11;
     ?column?
```

```
------------------
 9.60335403773934
(1 row)
dsd_22=# select random()*11;
     ?column?
-------------------
 0.160588529892266
(1 row)

dsd_22=# select random()*11;
     ?column?
------------------
 5.25661591161042
(1 row)

dsd_22=# select random()*11;
     ?column?
------------------
 7.78145408304408
(1 row)
dsd_22=# select random()*11;
     ?column?
------------------
 10.1819118564017
(1 row)
```

Q14. Connect to your home database, upxxxxxxx and run the following code to create a new table and insert some random numbers into it.

LANGUAGE: SQL

```
create table numb1(numb_id int primary key, ran_val decimal(17,15));

insert into numb1(numb_id, ran_val) values
(1,random()),(2,random()),(3,random()),(4,random()),(5,random()),(6,random()),(7,random()),(8,
    ↪ random()),(9,random()),(10,random());
```

LANGUAGE: Unknown

```
    INSERT 0 10
```

Q14a. Check that there are 10 rows of data with `SELECT COUNT(*) FROM NUMB1;` If not, check your output for any error messages. You should get responses below except the prompt will be your student id number.

LANGUAGE: Unknown

```
 count
-------
    10
(1 row)
```

Q15. Run a `SELECT * FROM NUMB1;` Copy the output below.

LANGUAGE: Unknown

```
 numb_id |      ran_val
---------+-------------------
       1 | 0.481754711363465
       2 | 0.020102311857045
       3 | 0.541421711910516
       4 | 0.046512784436345
       5 | 0.842869907151908
       6 | 0.137599688488990
       7 | 0.925696460530162
       8 | 0.765472991392016
       9 | 0.712954005226493
```

```
      10 | 0.161490791942924
(10 rows)
```

Q15a. Compare the values that you get with the values below. They should be different. This is because the code used inserts a fixed value, the numb_id and a completely random value into the ran_val attribute for each row.

LANGUAGE: Unknown

```
test_num=# SELECT * FROM NUMB1;
 numb_id |      ran_val
---------+-------------------
     1 | 0.477631121408194
     2 | 0.978080025874078
     3 | 0.516494689509273
     4 | 0.849129045847803
     5 | 0.484937957022339
     6 | 0.895700289402157
     7 | 0.852438564877957
     8 | 0.727535046637058
     9 | 0.062769805546850
    10 | 0.594313766807318
(10 rows)
```

Q16. Find the highest value of ran_val using the max() function. Copy it below.

LANGUAGE: SQL

```
select max(ran_val) from numb1;
```

LANGUAGE: Unknown

```
        max
-------------------
 0.925696460530162
(1 row)
```

Q17. Find the lowest value of ran_val using the min() function. Copy it below.

LANGUAGE: SQL

```
select min(ran_val) from numb1;
```

LANGUAGE: Unknown

```
        min
-------------------
 0.020102311857045
(1 row)
```

Q18. What is the average value of ran_val. Reminder: look at the basic functions document for ideas.

LANGUAGE: SQL

```
select avg(ran_val) from numb1;
```

LANGUAGE: Unknown

```
          avg
```

```
------------------------
 0.46358753642998640000
(1 row)
```

Q19. What is the current timestamp on your server? Copy it below

LANGUAGE: SQL

```
select now();
```

LANGUAGE: Unknown

```
              now
-------------------------------
 2022-10-13 13:43:49.196518+00
(1 row)
```

Q20. What is the first name of the customer with the ID number of 3?

LANGUAGE: SQL

```
select cust_fname from customer where cust_id=3;
```

LANGUAGE: Unknown

```
 cust_fname
------------
 Penelope
(1 row)
```

Q21. What is the category id number of the outdoor category? Copy below.

LANGUAGE: SQL

```
select cat_id from category where cat_name='Outdoor';
```

LANGUAGE: Unknown

```
 cat_id
--------
      4
(1 row)
```

Q22. How many orders in the cust_order table are for cust_id 15? Copy below.

LANGUAGE: SQL

```
select count(*) from cust_order where cust_id=15;
```

LANGUAGE: Unknown

```
 count
-------
     0
(1 row)
```

Q23. List the first and last names of the staff members who live in Portsmouth. Copy below.

```sql
select staff_fname, staff_lname from staff where town='Portsmouth';
```

```
 staff_fname | staff_lname
-------------+-------------
 Niel        | Welsby
 Janeva      | Gillicuddy
(2 rows)
```

Q24. What values does addr1 and addr2 have for the staff member whose id = 4? Copy below.

```sql
    select addr1 , addr2 from staff where staff_id=4;
```

```
      addr1       | addr2
------------------+-------
 959 Algoma Plaza |
(1 row)
```

Q25. How many members of staff have the role value of 3? Copy below.

```sql
select count(*) from staff where role=3;
```

```
 count
-------
     3
(1 row)
```

Q26. How many products are in the product category = 2?

```sql
select count(*) from product where prod_id=2;
```

```
 count
-------
     1
(1 row)
```

LECTURE: Coursework & Entity Relationship Diagrams

20-10-22 13:00 Mark RB LT1

Coursework

Coursework

The coursework is now available on Moodle, within Assessment and Support Materials.

The deadline for the coursework isn't until February.
It is recommended to submit the files to Moodle well in advance of the deadline because there is a chance there will be a technical issue with Moodle when the deadline is, no extenuating circumstances will be given if this is the case.
The Entity Relationship Diagram submitted must be produced digitally, hand drawn diagrams will gain 0 credits.
Mark uses Mocakroo and Lucid Charts for generating dummy data and drawing ERDs respectively. This is what works well for him, there are other platforms available for both, with more information in the Coursework document.

Entity Relationship Diagrams

Entity Relationship Diagrams (ERDs) are diagrams which show how entities are related, down to the detail of what the attributes are and how they relate to each other as well.

Business Rules

When designing databases, business rules will be taken into consideration.

Business Ruls

A statement that defines how a company does stuff or how stuff works within a company.

We can use business rules to help guide us on how to design databases.

Relationship Links

We will be using Crows Foot Notation, there are a number of other types of notation however we won't look at any of these.

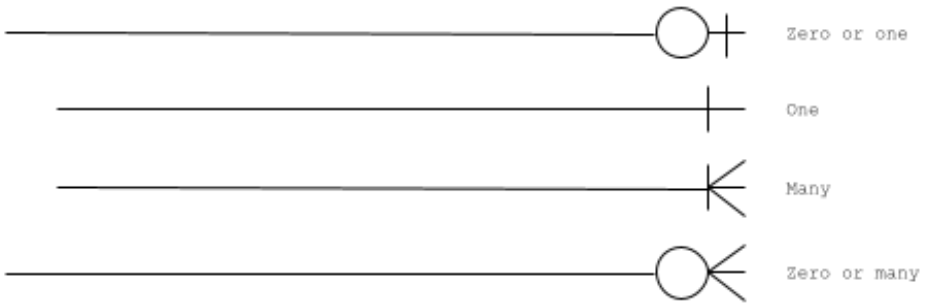

Figure 1: Crow's feet notation

When designing entities, it is important to name them in the singular, for example `pig` not `pigs`, and to use underscore notation where multiple words comprise the entity name, not camel notation.
Many-to-many relationships are not permitted. We will return to this in a future lecture.

Constraints

> **Constraint**
>
> A rule that protects your data or enforces certain behaviour.

For example, a constraint may be set to be `NOT NULL`, this would ensure that whenever a row of data is inserted into a table, that attribute would have to contain data.
Keys are constraints. The primary key is automatically set to be `NOT NULL`, we do not have to specify that when creating a table. We could use a default constraint, to specify the the time that a record was entered into a table.
Check constraints can be used to validate data as it is entered, for example a price must contain two decimal places. Check may be needed as part of the coursework.

PRACTICAL: SQL AND ENTITIES

20-10-22 14:00 Mark & Team FTC 3rd Floor

Task 1: Run the provided code and observe the outputs.

Run the following DDL code.

LANGUAGE: SQL

```
CREATE DATABASE customer_db;

\c customer_db

CREATE TABLE customer1 (cust_id SERIAL PRIMARY KEY, cust_fname VARCHAR(20) NOT NULL, cust_lname
    ↪  VARCHAR(20) NOT NULL);

\d customer1

ALTER TABLE customer1 ADD COLUMN cust_email varchar(100) NOT NULL UNIQUE;

\d customer1

DROP TABLE customer1;

-- check that the table is gone now

-- anything in the line after the two dashes is a comment by the way

\l
\d customer1
```

Run the following DML code.
Creating a new table and populating it with some dummy data.

LANGUAGE: SQL

```
CREATE TABLE customer (cust_id SERIAL PRIMARY KEY, cust_fname VARCHAR(20) NOT NULL, cust_lname
    ↪ VARCHAR(20) NOT NULL, cust_email varchar(60) NOT NULL);

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (22,'Kamil', 'Novak',
    ↪ 'kamnovak@gmail.com');

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (66,'Aarav', 'Anand',
    ↪ 'aanand98@gmail.com');

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (67,'Alia', 'Anand','
    ↪ aanand98@gmail.com');
```

Viewing what is in the table

LANGUAGE: SQL

```
SELECT * FROM customer;

SELECT cust_fname, cust_email from customer;
```

Selecting only the attributes which we need, so we don't have to retrieve all of the data from a table.

LANGUAGE: SQL

```
SELECT cust_email, cust_id, cust_fname, cust_lname from customer;
```

Insert more records, some of these return errors.

LANGUAGE: SQL

```sql
UPDATE customer SET cust_email = 'i_love_elephants_1@gmail.com' where cust_id = 66;
SELECT * FROM customer;

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (67,'Alia', 'Anand','aanand98@gmail.com');

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (70,'Connor', 'Murphy','connormurphy99@gmail.com');

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (71,'Connor', 'Murphy','connormurphy199@gmail.com');

INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (1,'Tim', 'Nice-but-Dimm','alongemailaddresswillfit@quitealongdomainnametoo.com');

INSERT INTO customer ( cust_fname, cust_lname, cust_email) VALUES ('Tim', 'Nice-but-Dimm','averylongemailaddresswillnotfit1111111111111111111111111111111111111111111a@quitealongdomainnametoo.com');
```

Task 2: Write SQL code for the following questions.

1. Create a new database called `code_test`

LANGUAGE: SQL

```sql
CREATE DATABASE code_test;
```

2. Connect to this new database

LANGUAGE: SQL

```sql
\c code_test
```

3. Create a new table called `table_one`, which has the following attributes
   (a) `Record_id` an integer
   (b) `Att_1` a varchar that will hold upto 30 characters
   (c) `Att_2` a char that will hold 10 characters
   (d) `Att_3` a decimal that can hold the value of 9.99.

LANGUAGE: SQL

```sql
CREATE TABLE table_one(Record_id INT PRIMARY KEY, Att_1 VARCHAR(30), Att_2 CHAR(10), Att_3 DECIMAL(3,2));
```

4. Look at the structure of this table once you have created it. Show the output below.

LANGUAGE: SQL

```sql
\d table_one
```

LANGUAGE: Unknown

```
                    Table "public.table_one"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
```

```
 record_id | integer                |           | not null |
 att_1     | character varying(30) |           |          |
 att_2     | character(10)          |           |          |
 att_3     | numeric(3,2)           |           |          |
Indexes:
    "table_one_pkey" PRIMARY KEY, btree (record_id)
```

5. Alter the table by adding a new column called `Att_4` that will hold another integer.

LANGUAGE: SQL

```sql
ALTER TABLE table_one ADD COLUMN Att_4 INT;
```

6. Look at the structure of this table again once you have added this new column. Show the output below.

LANGUAGE: SQL

```sql
\d table_one
```

LANGUAGE: Unknown

```
                      Table "public.table_one"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
 record_id | integer                |           | not null |
 att_1     | character varying(30) |           |          |
 att_2     | character(10)          |           |          |
 att_3     | numeric(3,2)           |           |          |
 att_4     | integer                |           |          |
Indexes:
    "table_one_pkey" PRIMARY KEY, btree (record_id)
```

7. Insert two records into the table called `table_one`

   (a) Record_id = 1, Att_1 = continent , Att2 = 0olP$fguj , Att_3 = 9.99 , Att_4 = 42

   (b) Record_id = 2, Att_1 = Portsmouth University , Att2 = Violet , Att_3 = 9.99 , Att_4 = 99999

LANGUAGE: SQL

```sql
INSERT INTO table_one (Record_id, Att_1, Att_2, Att_3, Att_4) VALUES (1, 'continent', '0
    ↪ olP[dollarSign]fguj', 9.99, 42);
INSERT INTO table_one (Record_id, Att_1, Att_2, Att_3, Att_4) VALUES (2, 'Portsmouth
    ↪ University', 'Violet', 9.99, 9999);
```

8. Get all fo the data from the table

LANGUAGE: SQL

```sql
SELECT * FROM table_one;
```

9. Get a screenshot of the data

LANGUAGE: Unknown

```
 record_id |         att_1          |   att_2    | att_3 | att_4
-----------+------------------------+------------+-------+-------
         1 | continent              | 0olP[dollarSign]fguj  |  9.99 |    42
         2 | Portsmouth University | Violet     |  9.99 |  9999
(2 rows)
```

10. Change the value of `Att_4` in record 1 from 44 to 66

```
LANGUAGE: SQL
UPDATE table_one SET Att_4 = 66 WHERE record_id = 1;
```

11. Get the data from the table for only record 1

```
LANGUAGE: SQL
SELECT * FROM table_one WHERE record_id = 1;
```

12. Get a screenshot of the results.

```
LANGUAGE: Unknown
 record_id |   att_1   |   att_2    | att_3 | att_4
-----------+-----------+------------+-------+-------
         1 | continent | 0olP[dollarSign]fguj  |  9.99 |    66
(1 row)
```

LECTURE: ERD, Attributes & Datatypes

27-10-22 13:00 RB LT1 Mark

Attributes

An entity is a thing. The attributes, of an entity, are the things which describe the thing. We need to be able to identify individual entities.

Example: People

If we are having a person as an entity, the attributes we will probably need are: date of birth; given name; family name. There are attributes which we don't need to store (for example: weight, height).

Addresses

When we store people, we will usually store their address in their record. This will be explored when do normalisation after consolidation week.

GDPR

When we store data, we have to be sure we are being GDPR compliant and storing what what you need to store.
GDPR states that you must ensure the personal data you are processing is:

- adequate - sufficient to properly fulfil your stated purpose;
- relevant - has a rational link to that purpose; and
- limited to what is necessary - you do not hold more than you need for that purpose.

Data Types

Now we know what attributes we need to store about the attribute, we need to think about types of data that is.

Names

Names are made up from characters, these could include apostrophes and hyphens. There is a question here as to how long names can be. A rule of thumb would be to use 20 characters for first name and 25 for surnames.

Numeric

There are a number of different numeric data types.

- `smallint` - holds an integer range -32768 to +32767
- `integer` - holds an integer range -2147483648 to +2147483647
- `bigint` - holds an integer range -9223372036854775808 to +9223372036854775807

- `decimal` - holds a decimal number with up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
- `real` - similar to decimal but provides 6 decimal digits precision
- `double` - similar to real but provides 15 decimal digits precision
- `serial` - holds an integer range 1 to 2147483647
- `bigserial` - holds an integer range 1 to 9223372036854775807

Characters

There are a number of different character data types.
Phone numbers should be stored as a character not as a numeric data type as they will often have leading zeros.

- `text` - variable 'unlimited' length
- `character/ char` - fixed length (blank padding is added if less than given size)
- `varying character / varchar` - variable length with limit

Dates and Times

There are a number of different date/ time data types.

- `timestamp without timezone` - both date and time (no time zone) range 4713 BC to 294276 AD with 1 microsecond resolution
- `timestamp with timezone` - both date and time (with time zone) range 4713 BC to 294276 AD with 1 microsecond resolution
- `date` - date without time range 4713 BC to 5874897 AD with 1 day resolution
- `time without timezone` - time of day (no date) range 00:00:00 to 24:00:00 with 1 microsecond resolution
- `time with timezone` - time of day (no date), with time zone range 00:00:00 to 24:00:00 with 1 microsecond resolution and adjustment for time zone

Example of drawing up an entity

If we have a draft entity with the following attributes `cust_id`, `cust_name`, `addeess`, `email`. This presents a number of problems.
If we want to search for a specific name, this is more complicated because the customer name is stored as a single attribute where it should be multiple attributes.
Addresses should not be stored as a single attribute.

Break down data

We should break down information into usable data. For example, addresses should be broken down into: `address1`, `address2`, `town`, `county`, `postcode`, `country`.
Names should be broken down into `firstName`, `lastName`. It could also be argued that a single middle name could also be included.

Adding data types

- `cust_id` - `int`
- `cust_fname` - `varchar`
- `cust_mname` - `varchar`
- `cust_lname` - `varchar`
- `addr1` - `varchar`
- `addr2` - `varchar`
- `town` - `varchar`
- `postcode` - `char` (could be a `varchar`)
- `email` - `varchar`

Sizes of data types

Now we have worked out what data types we want to use, we need to think about the sizes of those data types.

LECTURE: Normalisation

10-11-22 13:00 Mark RB LT1

Introduction to Normalisation

Normalisation is the process of designing a database in a way that reduces data redundancy and makes the database more efficient. As part of doing this, we have set rules to follow which enables us to decide what is stored in an entity and then within a table. There are five levels of normalisation, information which has not been normalised is in zero form and a database that has been normalised will be in 3rd normal form.

First Normal Form

Rules for a table to be in 1NF:

- It should only have single (atomic) valued attributes/ columns (each column should not hold more than one value)
- Values stored in a column should be of the same domain (this means don't hold char data in one row and int in another, both in the same columns)
- All the columns in a table should have unique names (there cannot be two or more columns or attributes with the same name)
- The order in which data is stored doesn't matter

Whilst converting data to the first normal form, you may find that a new entity is created. This can be done to reduce data redundancy.

Second Normal Form

Rules for a table to be in 2NF:

- Be in 1NF
- Have no partial dependencies

A partial dependency is where part of an attribute can be identified by something other than the primary key.

Third Normal Form

Rules for a table to be in 3NF:

- Be in 2NF
- Not have transitive dependencies

A transitive dependency is a n attribute which is dependent on an attribute which is not the primary key.

PRACTICAL: KEYS & JOINS

10-11-22 14:00 Mark and team FTC Floor 3

1. Connect to the `dsd_22` Database
2. Drop the `dsd_22` database using the code shown below and show the output below.

```sql
DROP DATABASE dsd_22;
```

```
ERROR:  cannot drop the currently open database
```

3. If you were unable to drop the database, how did you do it? Show your code below.

```sql
\c up2108121
DROP DATABASE dsd_22;
```

```
DROPPED DATABASE
```

4. Create the table but do not create any tableofcontents

```sql
CREATE DATABASE dsd_22;
```

5. Exit Postgres client but don't close connection to the VM
6. Download the code from Moodle
7. Use SCP through the terminal to copy the file to the virtual machine
8. Run the code to populate the database
9. Connect to the `dsd_22` database.
10. Check that the tables have been created with the `\dt` command and to check that there is data in each of them, select the number of rows in each table.

```
SELECT COUNT(*) FROM category;
 count
-------
     6
(1 row)

SELECT COUNT(*) FROM cust_order;
 count
-------
   150
```

```
(1 row)

SELECT COUNT(*) FROM customer;
 count
-------
    11
(1 row)

SELECT COUNT(*) FROM manifest;
 count
-------
   150
(1 row)

SELECT COUNT(*) FROM product;
 count
-------
   100
(1 row)

SELECT COUNT(*) FROM role;
 count
-------
     5
(1 row)

SELECT COUNT(*) FROM staff;
 count
-------
    10
(1 row)
```

11. Get a printout of the structure of each table by using the `\d` command.

LANGUAGE: Unknown

```
\d category
    Table "public.category"
Column  |          Type          | Collation | Nullable |                 Default

----------+------------------------+-----------+----------+----------------------------------------
    ↪
cat_id   | integer                |           | not null | nextval('category_cat_id_seq'::
    ↪ regclass)
cat_name | character varying(40) |           |          |
Indexes:
"category_pkey" PRIMARY KEY, btree (cat_id)
Referenced by:
TABLE "product" CONSTRAINT "product_prod_cat_fkey" FOREIGN KEY (prod_cat) REFERENCES
    ↪ category(cat_id)

\d cust_order
 Table "public.cust_order"
Column    |  Type   | Collation | Nullable |                    Default
------------+---------+-----------+----------+-----------------------------------------------
    ↪
cust_ord_id | integer |           | not null | nextval('cust_order_cust_ord_id_seq'::
    ↪ regclass)
staff_id    | integer |           |          |
cust_id     | integer |           |          |
Indexes:
"cust_order_pkey" PRIMARY KEY, btree (cust_ord_id)
Foreign-key constraints:
"cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
"cust_order_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
Referenced by:
TABLE "manifest" CONSTRAINT "manifest_cust_ord_id_fkey" FOREIGN KEY (cust_ord_id)
    ↪ REFERENCES cust_order(cust_ord_id)

\d customer
      Table "public.customer"
Column   |          Type          | Collation | Nullable |                  Default

```

```
------------+------------------------+-----------+----------+-------------------------------------
    ↪
cust_id     | integer                |           | not null | nextval('customer_cust_id_seq
    ↪ '::regclass)
cust_fname | character varying(25)  |           | not null |
cust_lname | character varying(35)  |           | not null |
addr1      | character varying(50)  |           | not null |
addr2      | character varying(50)  |           |          |
town       | character varying(60)  |           | not null |
postcode   | character(9)           |           | not null |
email      | character varying(255) |           | not null |
Indexes:
"customer_pkey" PRIMARY KEY, btree (cust_id)
Referenced by:
TABLE "cust_order" CONSTRAINT "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES
    ↪ customer(cust_id)

\d manifest
 Table "public.manifest"
Column     |  Type   | Collation | Nullable |                    Default
-------------+---------+-----------+----------+-----------------------------------------------
    ↪
manifest_id | integer |           | not null | nextval('manifest_manifest_id_seq'::
    ↪ regclass)
cust_ord_id | integer |           | not null |
prod_id     | integer |           | not null |
Indexes:
"manifest_pkey" PRIMARY KEY, btree (manifest_id)
Foreign-key constraints:
"manifest_cust_ord_id_fkey" FOREIGN KEY (cust_ord_id) REFERENCES cust_order(cust_ord_id)
"manifest_prod_id_fkey" FOREIGN KEY (prod_id) REFERENCES product(prod_id)


\d product
     Table "public.product"
Column   |         Type          | Collation | Nullable |                 Default

-----------+-----------------------+-----------+----------+---------------------------------------
    ↪
prod_id   | integer               |           | not null | nextval('product_prod_id_seq'::
    ↪ regclass)
prod_name | character varying(50) |           | not null |
prod_cat  | integer               |           | not null |
Indexes:
"product_pkey" PRIMARY KEY, btree (prod_id)
Foreign-key constraints:
"product_prod_cat_fkey" FOREIGN KEY (prod_cat) REFERENCES category(cat_id)
Referenced by:
TABLE "manifest" CONSTRAINT "manifest_prod_id_fkey" FOREIGN KEY (prod_id) REFERENCES
    ↪ product(prod_id)

\d role
     Table "public.role"
Column   |         Type          | Collation | Nullable |                Default

-----------+-----------------------+-----------+----------+-------------------------------------
    ↪
role_id   | integer               |           | not null | nextval('role_role_id_seq'::
    ↪ regclass)
role_name | character varying(20) |           |          |
Indexes:
"role_pkey" PRIMARY KEY, btree (role_id)
Referenced by:
TABLE "staff" CONSTRAINT "staff_role_fkey" FOREIGN KEY (role) REFERENCES role(role_id)



\d staff

       Table "public.staff"
Column    |          Type          | Collation | Nullable |                 Default

-------------+------------------------+-----------+----------+---------------------------------------
    ↪
staff_id    | integer                |           | not null | nextval('staff_staff_id_seq
```

```
    ↪ '::regclass)
staff_fname | character varying(25)  |           | not null |
staff_lname | character varying(35)  |           | not null |
addr1       | character varying(50)  |           | not null |
addr2       | character varying(50)  |           |          |
town        | character varying(60)  |           | not null |
postcode    | character(9)           |           | not null |
home_email  | character varying(255) |           | not null |
work_email  | character varying(100) |           | not null |
role        | integer                |           | not null |
Indexes:
"staff_pkey" PRIMARY KEY, btree (staff_id)
Foreign-key constraints:
"staff_role_fkey" FOREIGN KEY (role) REFERENCES role(role_id)
Referenced by:
TABLE "cust_order" CONSTRAINT "cust_order_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES
    ↪  staff(staff_id)#
```

12. Compare the printouts to the ERD found on Moodle.
13. Use the ERD to see which tables are related to which table.
14. How many rows of data do you get from the following:

LANGUAGE: SQL

```
Select * from product, category;
```

LANGUAGE: Unknown

```

 prod_id |                    prod_name                     | prod_cat | cat_id |
    ↪ cat_name
---------+--------------------------------------------------+----------+--------+-------------
    ↪
       1 | Multi-layered multi-tasking initiative           |        2 |      1 | Men's
    ↪ Wear
       2 | Operative analyzing task-force                   |        1 |      1 | Men's
    ↪ Wear
       3 | Exclusive client-server array                    |        5 |      1 | Men's
    ↪ Wear
       4 | Balanced client-server product                   |        6 |      1 | Men's
    ↪ Wear
       5 | Exclusive background website                     |        5 |      1 | Men's
    ↪ Wear
       6 | Pre-emptive holistic intranet                    |        6 |      1 | Men's
    ↪ Wear
       7 | Re-engineered cohesive methodology               |        1 |      1 | Men's
    ↪ Wear
       8 | Robust directional projection                    |        2 |      1 | Men's
    ↪ Wear
       9 | Inverse transitional infrastructure              |        4 |      1 | Men's
    ↪ Wear
      10 | Multi-tiered explicit paradigm                   |        6 |      1 | Men's
    ↪ Wear
...
(600 rows)
```

15. Look at the printout for the question above and find the category of the product "Multi-layered multi-tasking initiative"
16. Use the following command to narrow down the search

LANGUAGE: SQL

```
select * from category, product where prod_name = 'Multi-layered multi-tasking initiative'
    ↪ ;
```

When we don't join tables properly, the output we are given is called a 'Cartesian Product'. This is bad.

17. Run the following code

LANGUAGE: SQL

```sql
select * from category
join product on category.cat_id = product.prod_cat;
```

LANGUAGE: Unknown

```
 cat_id |  cat_name   | prod_id |                    prod_name                     |
    ↪ prod_cat
--------+-------------+---------+--------------------------------------------------+----------
    ↪
      2 | Ladies Wear |       1 | Multi-layered multi-tasking initiative           |
    ↪   2
      1 | Men's Wear  |       2 | Operative analyzing task-force                   |
    ↪   1
      5 | Sport       |       3 | Exclusive client-server array                    |
    ↪   5
      6 | Health      |       4 | Balanced client-server product                   |
    ↪   6
      5 | Sport       |       5 | Exclusive background website                     |
    ↪   5
      6 | Health      |       6 | Pre-emptive holistic intranet                    |
    ↪   6
      1 | Men's Wear  |       7 | Re-engineered cohesive methodology               |
    ↪   1
      2 | Ladies Wear |       8 | Robust directional projection                    |
    ↪   2
      4 | Outdoor     |       9 | Inverse transitional infrastructure              |
    ↪   4
      6 | Health      |      10 | Multi-tiered explicit paradigm                   |
    ↪   6
...
(100 rows)
```

18. How many rows are returned now.
100

19. Write the code to find the category information for the product "Multi-layered multi-tasking initiative"

LANGUAGE: SQL

```sql
select * from category
join product on category.cat_id = product.prod_cat
where prod_name = 'Multi-layered multi-tasking initiative';
```

LANGUAGE: Unknown

```
 cat_id |  cat_name   | prod_id |               prod_name                | prod_cat
--------+-------------+---------+----------------------------------------+----------
      2 | Ladies Wear |       1 | Multi-layered multi-tasking initiative |        2
(1 row)
```

20. Run the following code

LANGUAGE: SQL

```sql
select count(*) from customer, cust_order;
```

This will connect every customer to every order stored in the `cust_order` table.

LANGUAGE: Unknown

```
 count
-------
  1650
(1 row)
```

21. Write a query that will display the customer's first name, their last name and the order numbers, stored in the `cust_order` table as the `cust_ord_id`, but only for the customer with the `cust_id` of 1. Copy the code and the printout below.

LANGUAGE: SQL

```
select customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id from customer
join cust_order on customer.cust_id = cust_order.cust_id
where cust_order.cust_id = 1;
```

LANGUAGE: Unknown

```
 cust_fname | cust_lname | cust_ord_id
------------+------------+-------------
 Jobey      | Boeter     |          26
 Jobey      | Boeter     |          34
 Jobey      | Boeter     |          39
 Jobey      | Boeter     |          57
 Jobey      | Boeter     |          68
 Jobey      | Boeter     |          71
 Jobey      | Boeter     |          77
 Jobey      | Boeter     |          91
 Jobey      | Boeter     |          98
 Jobey      | Boeter     |          99
 Jobey      | Boeter     |         131
 Jobey      | Boeter     |         143
 Jobey      | Boeter     |         146
(13 rows)
```

22. Now try to see if you can add the `staff_fname`, the `staff_lname` to the above printout. You will need to join the `staff` table. Look at the ERD and the printout from to find the matching primary key and foreign key

LANGUAGE: SQL

```
select customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id, staff.staff_fname
    ↪ , staff.staff_lname from customer
join cust_order on customer.cust_id = cust_order.cust_id
join staff on cust_order.staff_id = staff.staff_id
where cust_order.cust_id = 1;
```

LANGUAGE: Unknown

```
 cust_fname | cust_lname | cust_ord_id | staff_fname | staff_lname
------------+------------+-------------+-------------+-------------
 Jobey      | Boeter     |          26 | Montgomery  | Housegoe
 Jobey      | Boeter     |          34 | Hanan       | Gloster
 Jobey      | Boeter     |          39 | Hanan       | Gloster
 Jobey      | Boeter     |          57 | Nikoletta   | Shrimpton
 Jobey      | Boeter     |          68 | Montgomery  | Housegoe
 Jobey      | Boeter     |          71 | Nikoletta   | Shrimpton
 Jobey      | Boeter     |          77 | Hanan       | Gloster
 Jobey      | Boeter     |          91 | Niel        | Welsby
 Jobey      | Boeter     |          98 | Montgomery  | Housegoe
 Jobey      | Boeter     |          99 | Janeva      | Gillicuddy
```

```
 Jobey       | Boeter      |          131 | Aura         | Clewlowe
 Jobey       | Boeter      |          143 | Janeva       | Gillicuddy
 Jobey       | Boeter      |          146 | Montgomery   | Housegoe
(13 rows)
```

23. If you have got this far, try to get a printout that joins the `role` table, the `staff` table, the `cust_order` table and the `customer` table. Retrieve the roles of anyone who has worked on an order for `cust_id` of 4.

LANGUAGE: SQL

```sql
select customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id, staff.staff_fname
    ↪ , staff.staff_lname, role.role_id, role.role_name from customer
join cust_order on customer.cust_id = cust_order.cust_id
join staff on cust_order.staff_id = staff.staff_id
join role on staff.role = role.role_id
where customer.cust_id = 4;
```

LANGUAGE: Unknown

```
 cust_fname |    cust_lname    | cust_ord_id | staff_fname | staff_lname | role_id |
    ↪ role_name
------------+------------------+-------------+-------------+-------------+---------+-----------------
    ↪
 Chadd      | Franz-Schoninger |           1 | Aura        | Clewlowe    |       3 | Post
    ↪ Sales
 Chadd      | Franz-Schoninger |           7 | Aura        | Clewlowe    |       3 | Post
    ↪ Sales
 Chadd      | Franz-Schoninger |          66 | Montgomery  | Housegoe    |       1 | Order
    ↪  Picker
 Chadd      | Franz-Schoninger |          81 | Janeva      | Gillicuddy  |       5 | Misc
 Chadd      | Franz-Schoninger |          93 | Niel        | Welsby      |       2 | Final
    ↪  Packer
 Chadd      | Franz-Schoninger |          97 | Aura        | Clewlowe    |       3 | Post
    ↪ Sales
 Chadd      | Franz-Schoninger |         107 | Hanan       | Gloster     |       4 |
    ↪ Customer Retain
 Chadd      | Franz-Schoninger |         109 | Nikoletta   | Shrimpton   |       4 |
    ↪ Customer Retain
 Chadd      | Franz-Schoninger |         124 | Aura        | Clewlowe    |       3 | Post
    ↪ Sales
 Chadd      | Franz-Schoninger |         129 | Nikoletta   | Shrimpton   |       4 |
    ↪ Customer Retain
(10 rows)
```

LECTURE: Joins and Narrowing Focus

17-11-22 13:00 Mark RB LT1

Introduction to Joins

Joins are key to understanding how to get useful information out of a database. Data in an individual table is of limited use, to get good data, we need to join multiple tables together. We might only want some information.
To get these individual items from one table, we can do this with

```sql
SELECT firstName, lastName, emailAddress from TABLE;
```

However, this will still return every record.
We can narrow this down, using the `WHERE=condition` clause. For example,

```sql
SELECT firstName, lastName, emailAddress WHERE town = 'Portsmouth';
```

This will give us all the records where the town attribute is equal to Portsmouth
What if we want to get data from multiple tables? Here we have to use Joins.

Joins

To create a join between two tables, one table needs to have a foreign key where that is the primary key in the other table you wish to join.
When creating joins between tables, it's important to ensure that the correct attributes in each tables are joined. Just because an result is produced form the query, it doesn't necessarily mean its the right one.
The data types between the two attributes which are being joined have to match whilst the names used in each table do not.

Cartesian Product

This is the result of a wrong join.
It is where every single record in one table is joined to every single table in another table. For example, two tables: customer and order. Customer has 11 records and order has 150. 150×11 gives 1650 rows as output. This provides a big problem when attempting to join two big tables together.

The Correct Way

When joining two tables correctly, we have to tell the DMBS what values match.

```sql
SELECT CUSTOMER.CUST_ID, CUST_ORD_ID FROM CUSTOMER JOIN cust_order ON CUSTOMER.CUST_ID =
    ↪ CUST_ORDER.CUST_ID;
```

Query above returns 150 rows of data. We know this is correct as it is the same as the number of rows in orders table.

Another Correct Way

We do not have to use the `join` keyword, instead we can use the `WHERE` condition.

```sql
SELECT CUST_LNAME, CUST_ORD_ID FROM CUSTOMER, CUST_ORDER WHERE CUSTOMER.CUST_ID = CUST_ORDER.
    ↪ CUST_ID;
```

This will happily produce 150 rows.
To join more than two tables, we have to use an `AND` statement in the `WHERE` condition.

PRACTICAL: Normalisations and Joins

24-11-22 14:00 Val FTC Floor 3

Order of Execution

1. FROM & JOIN (chose and join tables to get base data)
2. WHERE & SUBQUERY/ INTERSECTION/ UNION/ EXCEPT (filters the base data)
3. GROUP BY (aggregates the base data)
4. HAVING (filters the aggregated ata)
5. SELECT (returns the final data, as functionality not displayed)
6. ORDER BY (sort the final data)
7. LIMIT (limits the returned data to a row count)
8. display data

Task 1

See Google Doc and Lucid Chart.

Task 2

1. Write a query to retrieve the first and last names of the customers in the customer table. Copy the query and the answer below.

LANGUAGE: SQL

```sql
SELECT cust_fname, cust_lname from customer;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname
------------------+------------------
 Jobey            | Boeter
 York             | O'Deegan
 Penelope         | Hexter
 Chadd            | Franz-Schoninger
 Vikky            | Eke
 Marie-françoise  | Currier
 Bénédicte        | Dozdill
 Görel            | Douthwaite
 Bérengère        | Menendez
 Pélagie          | Hachard
 Adaobi           | Musa
(11 rows)
```

2. Write a query to retrieve the first and last names and the towns they live in of the customers in the customer table. Copy the query and the answer below.

LANGUAGE: SQL

```sql
SELECT cust_fname, cust_lname, town FROM customer;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     |      town
------------------+-------------------+----------------
 Jobey            | Boeter            | La Mohammedia
 York             | O'Deegan          | Chemnitz
 Penelope         | Hexter            | Pingshan
 Chadd            | Franz-Schoninger  | Baojia
 Vikky            | Eke               | Kamenný řPívoz
 Marie-françoise  | Currier           | Waekolong
 Bénédicte        | Dozdill           | Dawuhan
 Görel            | Douthwaite        | Sunbu
 Bérengère        | Menendez          | Tsagaanders
 Pélagie          | Hachard           | Jiantou
 Adaobi           | Musa              | La Mohammedia
(11 rows)
```

3. Print out the first and last name of the customer / customers who live in La Mohammedia. Copy the query and the answer below.

LANGUAGE: SQL

```sql
SELECT cust_fname, cust_lname FROM customer WHERE town= 'La Mohammedia';
```

LANGUAGE: Unknown

```
 cust_fname | cust_lname
------------+------------
 Jobey      | Boeter
 Adaobi     | Musa
(2 rows)
```

4. Get the structure of the tables customer and cust_order using the `\d` command. Copy the code and the answer below.

LANGUAGE: Unknown

```
dsd_22=# \d customer
                                      Table "public.customer"
   Column    |          Type          | Collation | Nullable |                    Default
-------------+------------------------+-----------+----------+-----------------------------------------
    ↪
 cust_id     | integer                |           | not null | nextval('customer_cust_id_seq'::
    ↪ regclass)
 cust_fname  | character varying(25)  |           | not null |
 cust_lname  | character varying(35)  |           | not null |
 addr1       | character varying(50)  |           | not null |
 addr2       | character varying(50)  |           |          |
 town        | character varying(60)  |           | not null |
 postcode    | character(9)           |           | not null |
 email       | character varying(255) |           | not null |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (cust_id)
Referenced by:
    TABLE "cust_order" CONSTRAINT "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES
    ↪ customer(cust_id)

dsd_22=# \d cust_order
                                  Table "public.cust_order"
   Column     |  Type   | Collation | Nullable |                      Default
--------------+---------+-----------+----------+-------------------------------------------------
    ↪
```

```
 cust_ord_id | integer |           | not null | nextval('cust_order_cust_ord_id_seq'::regclass)
 staff_id    | integer |           |          |
 cust_id     | integer |           |          |
Indexes:
    "cust_order_pkey" PRIMARY KEY, btree (cust_ord_id)
Foreign-key constraints:
    "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
    "cust_order_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
Referenced by:
    TABLE "manifest" CONSTRAINT "manifest_cust_ord_id_fkey" FOREIGN KEY (cust_ord_id)
    ↪ REFERENCES cust_order(cust_ord_id)
```

5. According to the answer from question 4, what are the names of the attributes in both tables that are the primary key and foreign keys? (hint - look at the section “Foreign-key constraints:” that appears in one of your outputs. Remember we are looking at customer and cust_order)
customer pk - cust_id
cust_order pk - cust_ord_id
cust_order fk - cust_id
cust_order fk - staff_id
6. List all of the categories. Copy the query and the answer below.

LANGUAGE: SQL

```
SELECT * FROM category;
```

LANGUAGE: Unknown

```
 cat_id |  cat_name
--------+-------------
      1 | Men's Wear
      2 | Ladies Wear
      3 | Kid's Wear
      4 | Outdoor
      5 | Sport
      6 | Health
(6 rows)
```

7. What is the id number for the category Sport? Copy the query and the answer below.

LANGUAGE: SQL

```
SELECT cat_id from category where cat_name='Sport';
```

LANGUAGE: Unknown

```
 cat_id
--------
      5
(1 row)
```

8. Write a query that joins the product table and the category table and prints out the prod_name and the appropriate category. Copy the query and the answer below. (You can copy the just first screen of data if you want)

LANGUAGE: SQL

```
SELECT product.prod_name, category.cat_name FROM product
JOIN category ON category.cat_id = product.prod_cat;
```

LANGUAGE: Unknown

```
                    prod_name                      |  cat_name
---------------------------------------------------+-------------
 Multi-layered multi-tasking initiative            | Ladies Wear
 Operative analyzing task-force                    | Men's Wear
 Exclusive client-server array                     | Sport
 Balanced client-server product                    | Health
 Exclusive background website                      | Sport
 Pre-emptive holistic intranet                     | Health
 Re-engineered cohesive methodology                | Men's Wear
 Robust directional projection                     | Ladies Wear
 Inverse transitional infrastructure               | Outdoor
 Multi-tiered explicit paradigm                    | Health
...
(100 rows)
```

9. Write a query that will list each staff member's first and last name along with their work email and the role name that they hold. Copy the query and the answer below.

LANGUAGE: SQL

```
SELECT staff.staff_fname, staff.staff_lname, staff.work_email, role.role_name from staff
JOIN role ON staff.role = role.role_id;
```

LANGUAGE: Unknown

```
 staff_fname | staff_lname |         work_email          |    role_name
-------------+-------------+-----------------------------+-----------------
 Montgomery  | Housegoe    | Montgomery.Housegoe@dsd.com | Order Picker
 Niel        | Welsby      | Niel.Welsby@dsd.com         | Final Packer
 Jillene     | Revitt      | Jillene.Revitt@dsd.com      | Post Sales
 Harriette   | Fewster     | Harriette.Fewster@dsd.com   | Post Sales
 Aura        | Clewlowe    | Aura.Clewlowe@dsd.com       | Post Sales
 Hanan       | Gloster     | Hanan.Gloster@dsd.com       | Customer Retain
 Nikoletta   | Shrimpton   | Nikoletta.Shrimpton@dsd.com | Customer Retain
 Tim         | Illem       | Tim.Illem@dsd.com           | Misc
 Nell        | Olsson      | Nell.Olsson@dsd.com         | Misc
 Janeva      | Gillicuddy  | Janeva.Gillicuddy@dsd.com   | Misc
(10 rows)
```

10. Write a query that will show the last name and the role of staff members who put together orders from the customer whose last name is Eke. Include the cust_order_id and the customer's first and last names. Copy the query and the answer below.

LANGUAGE: SQL

```
SELECT staff.staff_lname, role.role_name, cust_order.cust_ord_id, customer.cust_fname, customer
    ↪ .cust_lname FROM staff
JOIN role ON staff.role = role.role_id
JOIN cust_order ON cust_order.staff_id = staff.staff_id
JOIN customer ON customer.cust_id = cust_order.cust_id
WHERE customer.cust_lname = 'Eke';
```

LANGUAGE: Unknown

```
 staff_lname |    role_name    | cust_ord_id | cust_fname | cust_lname
-------------+-----------------+-------------+------------+------------
 Welsby      | Final Packer    |          82 | Vikky      | Eke
 Clewlowe    | Post Sales      |          90 | Vikky      | Eke
 Welsby      | Final Packer    |         105 | Vikky      | Eke
 Housegoe    | Order Picker    |         115 | Vikky      | Eke
 Gillicuddy  | Misc            |         118 | Vikky      | Eke
 Welsby      | Final Packer    |         130 | Vikky      | Eke
 Shrimpton   | Customer Retain |         132 | Vikky      | Eke
 Welsby      | Final Packer    |         135 | Vikky      | Eke
```

```
 Housegoe     | Order Picker     |          145 | Vikky       | Eke
(9 rows)
```

11. Write a query that lists only the category names and the custome's last names for orders that have been placed by people who live in Sunbu. Copy the query and answer below.

LANGUAGE: SQL

```sql
SELECT customer.cust_lname, category.cat_name FROM customer
JOIN cust_order ON customer.cust_id = cust_order.cust_id
JOIN manifest ON cust_order.cust_ord_id = manifest.cust_ord_id
JOIN product ON product.prod_id = manifest.prod_id
JOIN category ON category.cat_id = product.prod_cat
WHERE customer.town = 'Sunbu';
```

LANGUAGE: Unknown

```
 cust_lname |  cat_name
------------+-------------
 Douthwaite | Outdoor
 Douthwaite | Sport
 Douthwaite | Kid's Wear
 Douthwaite | Outdoor
 Douthwaite | Sport
 Douthwaite | Sport
 Douthwaite | Ladies Wear
 Douthwaite | Outdoor
 Douthwaite | Sport
 Douthwaite | Ladies Wear
(10 rows)
```

12. This is a bit harder than the previous queries. Try to group the orders and count the number of orders in each category for the results from q11. (hint - this might be a bit difficult. Grouping does not allow a WHERE, use HAVING instead). Copy the query and answer below.

LANGUAGE: SQL

```sql
SELECT customer.cust_lname, count(category.cat_name), category.cat_name FROM customer
JOIN cust_order ON customer.cust_id = cust_order.cust_id
JOIN manifest ON cust_order.cust_ord_id = manifest.cust_ord_id
JOIN product ON product.prod_id = manifest.prod_id
JOIN category ON category.cat_id = product.prod_cat
GROUP BY customer.cust_lname, category.cat_name, customer.town
HAVING customer.town='Sunbu';
```

LANGUAGE: Unknown

```
 cust_lname | count |  cat_name
------------+-------+-------------
 Douthwaite |     1 | Kid's Wear
 Douthwaite |     2 | Ladies Wear
 Douthwaite |     3 | Outdoor
 Douthwaite |     4 | Sport
```

LECTURE: Types of Joins

01-12-22 13:00 Mark RB LT1

The joins we have looked at so far are `inner` joins. This displays the data where the tables overlap. For example

```sql
SELECT CUSTOMER.CUST_ID, CUST_ORDER.CUST_ORD_ID FROM CUSTOMER
JOIN CUST_ORDER ON CUSTOMER.CUST_ID=CUST_ORDER.CUST_ID;
```

Will probably use this the most.

Left Join

This will produce everything form the left table (`customer`) and the overlapping data from the right hand table (`cust_order`) where there is a match on the common attribute to both (`cust_id`)

```sql
SELECT CUSTOMER.CUST_ID, CUST_ORDER.CUST_ORD_ID FROM CUSTOMER
LEFT JOIN CUST_ORDER ON CUSTOMER.CUST_ID= CUST_ORDER.CUST_ID;
```

Right Join

This will return everything from the right table (`cust_order`) and common data where it is there.

```sql
SELECT CUSTOMER.CUST_ID, CUST_ORDER.CUST_ORD_ID FROM CUSTOMER
RIGHT JOIN CUST_ORDER ON CUSTOMER.CUST_ID= CUST_ORDER.CUST_ID;
```

It is important to use the correct join for the situation as when used incorrectly as you won't get the data returned which you are expecting.

Outer Joins

This gives everything from all the tables mentioned in the query.

```sql
SELECT role_name, staff_lname, staff_fname FROM staff FULL OUTER JOIN
ROLE ON ROLE=role_id;
```

Will probably use this the least.

Things To Remember

- Use the correct type of join for the job
- Match like for like

PRACTICAL: FURTHER JOINS

01-12-22 14:00 Mark etc FTC 3

Tutor Led

We need to insert two more roles into the `Role` table.

LANGUAGE: SQL

```sql
INSERT INTO ROLE (role_name)
VALUES ('Cleaner');

INSERT INTO ROLE (role_name)
VALUES ('Pre Sales');
```

Then run the following.

LANGUAGE: SQL

```sql
SELECT count(*)
FROM ROLE;
```

This generates the following output

LANGUAGE: Unknown

```
 count
-------
     7
(1 row)
```

Student Tasks

1. Write a query that correctly displays the staff members first and last names, their email addresses and their roles. Use the method that uses the `JOIN` keyword. Copy the code and answer below.

LANGUAGE: SQL

```sql
SELECT staff.staff_fname, staff.staff_lname, staff.home_email, role.role_name FROM staff
JOIN role on staff.role = role.role_id;
```

LANGUAGE: Unknown

```
 staff_fname | staff_lname |         home_email           |    role_name
-------------+-------------+------------------------------+-----------------
 Montgomery  | Housegoe    | mhousegoe2@ucoz.ru           | Order Picker
 Niel        | Welsby      | nwelsby0@rambler.ru          | Final Packer
 Jillene     | Revitt      | jrevitt8@cornell.edu         | Post Sales
 Harriette   | Fewster     | hfewster7@independent.co.uk  | Post Sales
 Aura        | Clewlowe    | aclewlowe5@google.com.au     | Post Sales
 Hanan       | Gloster     | hgloster3@blogger.com        | Customer Retain
 Nikoletta   | Shrimpton   | nshrimpton1@unblog.fr        | Customer Retain
 Tim         | Illem       | tillem9@dedecms.com          | Misc
 Nell        | Olsson      | nolsson6@jiathis.com         | Misc
 Janeva      | Gillicuddy  | jgillicuddy4@altervista.org  | Misc
(10 rows)
```

2. Rewrite the query created in 1 but this time use the `WHERE` keyword. Copy the code and answer below.

LANGUAGE: SQL

```sql
SELECT staff.staff_fname, staff.staff_lname, staff.home_email, role.role_name FROM staff, role
WHERE staff.role = role.role_id;
```

LANGUAGE: Unknown

```
 staff_fname | staff_lname |         home_email          |    role_name
-------------+-------------+-----------------------------+-----------------
 Montgomery  | Housegoe    | mhousegoe2@ucoz.ru          | Order Picker
 Niel        | Welsby      | nwelsby0@rambler.ru         | Final Packer
 Jillene     | Revitt      | jrevitt8@cornell.edu        | Post Sales
 Harriette   | Fewster     | hfewster7@independent.co.uk | Post Sales
 Aura        | Clewlowe    | aclewlowe5@google.com.au    | Post Sales
 Hanan       | Gloster     | hgloster3@blogger.com       | Customer Retain
 Nikoletta   | Shrimpton   | nshrimpton1@unblog.fr       | Customer Retain
 Tim         | Illem       | tillem9@dedecms.com         | Misc
 Nell        | Olsson      | nolsson6@jiathis.com        | Misc
 Janeva      | Gillicuddy  | jgillicuddy4@altervista.org | Misc
(10 rows)
```

3. List the customer first and last names with their email addresses and the product names of the products they have ordered. But only for the customers who live in Waekolong. Copy the code and the answer below.

LANGUAGE: SQL

```sql
SELECT customer.cust_fname, customer.cust_lname, customer.email, product.prod_name FROM
    ↪ customer
JOIN cust_order ON customer.cust_id=cust_order.cust_id
JOIN manifest ON cust_order.cust_ord_id=manifest.cust_ord_id
JOIN product on manifest.prod_id=product.prod_id
WHERE customer.town='Waekolong';
```

LANGUAGE: Unknown

```
   cust_fname    | cust_lname |          email          |                    prod_name
-----------------+------------+-------------------------+-------------------------------------------------
    ↪
 Marie-françoise | Currier    | acurrier0@economist.com | Vision-oriented attitude-oriented
    ↪ core
 Marie-françoise | Currier    | acurrier0@economist.com | Balanced client-server product
 Marie-françoise | Currier    | acurrier0@economist.com | Exclusive client-server array
 Marie-françoise | Currier    | acurrier0@economist.com | Universal encompassing conglomeration
 Marie-françoise | Currier    | acurrier0@economist.com | Synergistic homogeneous ability
 Marie-françoise | Currier    | acurrier0@economist.com | Universal exuding protocol
 Marie-françoise | Currier    | acurrier0@economist.com | Universal global hub
 Marie-françoise | Currier    | acurrier0@economist.com | Balanced real-time info-mediaries
 Marie-françoise | Currier    | acurrier0@economist.com | Integrated 24/7 interface
 Marie-françoise | Currier    | acurrier0@economist.com | Re-engineered explicit software
 Marie-françoise | Currier    | acurrier0@economist.com | Customizable cohesive capacity
 Marie-françoise | Currier    | acurrier0@economist.com | Robust mission-critical complexity
 Marie-françoise | Currier    | acurrier0@economist.com | Organic clear-thinking system engine
 Marie-françoise | Currier    | acurrier0@economist.com | Stand-alone composite Graphical User
    ↪ Interface
(14 rows)
```

4. Write a query that returns all categories and the product names and order the output into category order. Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT category.cat_name, product.prod_name FROM category
JOIN product ON product.prod_cat = category.cat_id
ORDER BY category.cat_name;
```

LANGUAGE: Unknown

```
  cat_name    |                    prod_name
--------------+---------------------------------------------------
 Health       | Exclusive multimedia middleware
 Health       | Pre-emptive holistic intranet
 Health       | Ameliorated next generation orchestration
 Health       | Monitored asynchronous function
 Health       | Right-sized mission-critical pricing structure
 Health       | Profound human-resource forecast
 Health       | Realigned client-driven database
 Health       | Seamless optimal leverage
 Health       | User-friendly encompassing array
 Health       | Customizable cohesive capacity
...
(100 rows)
```

5. Rewrite the query for Q4 so that the output is ordered by category, then the product id. Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT category.cat_name, product.prod_name FROM category
JOIN product ON product.prod_cat = category.cat_id
ORDER BY category.cat_name, product.prod_id;
```

LANGUAGE: Unknown

```
  cat_name    |                    prod_name
--------------+---------------------------------------------------
 Health       | Balanced client-server product
 Health       | Pre-emptive holistic intranet
 Health       | Multi-tiered explicit paradigm
 Health       | Monitored asynchronous function
 Health       | Right-sized mission-critical pricing structure
 Health       | Open-architected homogeneous concept
 Health       | Fully-configurable full-range interface
 Health       | Customizable cohesive capacity
 Health       | Seamless optimal leverage
 Health       | Realigned client-driven database
...
(100 rows)
```

6. How can you prove that the product id is being used to do the ordering? (You may have already done this in Q5). Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT category.cat_name, product.prod_name, product.prod_id FROM category
JOIN product ON product.prod_cat = category.cat_id
ORDER BY category.cat_name, product.prod_id;
```

LANGUAGE: Unknown

```
  cat_name    |                    prod_name                    | prod_id
--------------+-------------------------------------------------+---------
 Health       | Balanced client-server product                  |       4
 Health       | Pre-emptive holistic intranet                   |       6
```

```
 Health      | Multi-tiered explicit paradigm                    |      10
 Health      | Monitored asynchronous function                   |      20
 Health      | Right-sized mission-critical pricing structure    |      23
 Health      | Open-architected homogeneous concept              |      37
 Health      | Fully-configurable full-range interface           |      46
 Health      | Customizable cohesive capacity                    |      54
 Health      | Seamless optimal leverage                         |      57
 Health      | Realigned client-driven database                  |      59
...
(100 rows)
```

7. Write a query that will list all staff members first and last names along with their email addresses that are cleaners. Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT staff.staff_fname, staff.staff_lname, staff.work_email FROM staff
JOIN role ON staff.role=role.role_id
WHERE role.role_name='Cleaner';
```

LANGUAGE: Unknown

```
 staff_fname | staff_lname | work_email
-------------+-------------+------------
(0 rows)
```

8. How many staff are there who have the role Misc? Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT count(*) FROM staff
JOIN role ON staff.role = role.role_id
WHERE role.role_name='Misc';
```

LANGUAGE: Unknown

```
 count
-------
     3
(1 row)
```

9. What are the addresses of the staff that are returned by the query for Q8? You should output their first and last names too. Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT staff.staff_fname, staff.staff_lname, concat_ws(' ', addr1, addr2, town, postcode) AS "
    ↪ address"
FROM staff
JOIN role ON role.role_id = staff.role
WHERE role.role_name='Misc';
```

LANGUAGE: Unknown

```
 staff_fname | staff_lname |                    address
-------------+-------------+-----------------------------------------------
 Janeva      | Gillicuddy  | 6999 Kings Park Sachtjen Portsmouth PO05 5SF
 Nell        | Olsson      | 18424 Kenwood Court Farmco Havant PO22 6DL
 Tim         | Illem       | 85 Lillian Way Farragut Southsea PO93 0CN
(3 rows)
```

10. List the product id numbers with their names that start with the letters Re . Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT prod_id, prod_name FROM product
WHERE prod_name LIKE 'Re%';
```

LANGUAGE: Unknown

```
 prod_id |                     prod_name
---------+-----------------------------------------------------
       7 | Re-engineered cohesive methodology
      11 | Re-engineered explicit software
      18 | Re-engineered actuating capability
      26 | Realigned 5th generation artificial intelligence
      39 | Realigned homogeneous hub
      56 | Reduced fresh-thinking process improvement
      59 | Realigned client-driven database
      76 | Re-engineered 24/7 knowledge base
(8 rows)
```

11. List the product id numbers with their names that have the word `value` in the name somewhere. Copy the code and the answer below.

LANGUAGE: SQL

```
SELECT prod_id, prod_name FROM product
WHERE prod_name LIKE '%value%';
```

LANGUAGE: Unknown

```
 prod_id |           prod_name
---------+-------------------------------
      80 | Profound value-added intranet
(1 row)
```

12. List the product names along with their id numbers that have `Value` somewhere in their name. Copy the code and the answer below

LANGUAGE: SQL

```
SELECT prod_id, prod_name FROM product
WHERE prod_name LIKE '%Value%';
```

LANGUAGE: Unknown

```
 prod_id | prod_name
---------+-----------
(0 rows)
```

13. List the customer first and last names along with their email addresses, the customer order id, the category names and the product names for orders that have been placed for all products that have the word `able` in the name. (The case matters). Order by the category and the product name. The output should have the category names in alphabetical order then within each category the products should be ordered in alphabetical order. Copy the code and the answer below.

LANGUAGE: SQL

```sql
SELECT customer.cust_fname, customer.cust_lname, customer.email, cust_order.cust_ord_id,
    ↪ category.cat_name, product.prod_name from customer
JOIN cust_order ON customer.cust_id=cust_order.cust_id
JOIN manifest ON cust_order.cust_ord_id=manifest.cust_ord_id
JOIN product on manifest.prod_id=product.prod_id
JOIN category on category.cat_id=product.prod_cat
WHERE product.prod_name LIKE '%able%'
ORDER BY category.cat_name, product.prod_name;
```

LANGUAGE: Unknown

```
   cust_fname    |    cust_lname    |              email               | cust_ord_id |  cat_name
    ↪   |
 prod_name
-----------------+------------------+----------------------------------+-------------+------------+------------
    ↪
 Bérengère       | Menendez         | amenendez3@dell.com              |          64 | Health
    ↪   | Customizable cohesive capacity
 Marie-françoise | Currier          | acurrier0@economist.com          |         133 | Health
    ↪   | Customizable cohesive capacity
 Bérengère       | Menendez         | amenendez3@dell.com              |         102 | Health
    ↪   | Fully-configurable full-range interface
 Chadd           | Franz-Schoninger | cfranzschoninger3@google.com.hk  |           7 | Health
    ↪   | Team-oriented stable project
 Chadd           | Franz-Schoninger | cfranzschoninger3@google.com.hk  |          81 | Health
    ↪   | Team-oriented stable project
 Bénédicte       | Dozdill          | cdozdill1@amazon.de              |          24 | Kid's
    ↪ Wear | Configurable analyzing solution
 Bérengère       | Menendez         | amenendez3@dell.com              |          21 | Kid's
    ↪ Wear | Configurable analyzing solution
 Bérengère       | Menendez         | amenendez3@dell.com              |         113 | Kid's
    ↪ Wear | Configurable analyzing solution
 Jobey           | Boeter           | jboeter0@mail.ru                 |          91 | Kid's
    ↪ Wear | Configurable analyzing solution
 Jobey           | Boeter           | jboeter0@mail.ru                 |          39 | Outdoor
    ↪   | Switchable tangible product
 Jobey           | Boeter           | jboeter0@mail.ru                 |          26 | Outdoor
    ↪   | Switchable tangible product
 Vikky           | Eke              | veke4@elegantthemes.com          |         105 | Sport
    ↪   | Configurable methodical firmware
 Vikky           | Eke              | veke4@elegantthemes.com          |         118 | Sport
    ↪   | Customizable well-modulated encryption
 Pélagie         | Hachard          | fhachard4@blinklist.com          |          89 | Sport
    ↪   | Virtual stable Graphic Interface
(14 rows)
```

LECTURE Security Basics I

08-12-22 13:00 Mark RB LT1

This lecture has been split into two parts, the second part will take place after the Christmas break.
Next week's lecture will be part about MS Learn (& part about Databases) and the practical next week is optional, aimed around coursework questions.

A View on Security

Stealing data is very different to stealing physical objects. To steal data, you just have to make a copy of it; whereas with physical things, you have to pick up the physical thing.
At one time, physical security was talked about much more. Nowerdays, the physical hardware is stored on the cloud where this is dealt with by someone else.
When working on developing applications, you have to 'sanitise' data which is passed to the database.
The biggest risk to data is those who have access to it, generally this will be people who work for the company.

PostgreSQL Basic Security

Our user account in our Postgres install has full administrative rights to Postgres. This is the Superuser account which no one else should have access to. By default, you cannot access the server from a different IP address; it is possible to allow other IP addresses to have access to this however this is un-advised.
Currently, the superuser on our databases doesn't have a password. In the real world, this is very stupid and should never happen. As superusers we can change and set other users passwords.

Roles

In Postgres, a role is the same as a user.
Before you can login to Postgres, there has to be a role in the DBMS to allow you to login. This username is case sensitive.
As well as having a role/ user there has to be other things in the database. For us, this is the table called our up number.
Users should (in the real world, must) be given passwords. Constraints and change-after-time policies can be set. When the user is created, the password is set. This is a potential security risk as if someone else can get into your account, they can view your terminal history, including the passwords you've entered in terminal in plain text.
Users have to be given the ability to log in. Removing the log in ability, can be useful for people who are working temporarily for a company.
The syntax to create a role as follows:

LANGUAGE: SQL

```sql
CREATE role [userName] with login password '[password]';
```

Where `[userName]` and `[password]` are replaced with values you wish to enter.
There is also a `CREATE user` command however this returns the same value as `CREATE role`.

When creating a role, this will create a database called their username, this is essential and should not be deleted.
After creating a role, you have to specify permissions for the different users. However, you can login (if you have login permission) and see all the names of all the databases.

Views

Including views in the coursework will give additional marks.

> **View**
>
> A pre-written query

This enables us to delegate access to certain parts of a table.
When you create views, you can give users access to be able to run that query.
To create a view, the syntax follows

LANGUAGE: SQL

```sql
CREATE [viewName] AS [queryString];

--eg
CREATE VIEW CUST_NAMES AS SELECT CUST_FNAME, CUST_LNAME FROM customer;
```

The view above can be executed as

LANGUAGE: SQL

```sql
SELECT * FROM CUST_NAMES;
```

This will display a list of all the customers first names and customers last names.

PRACTICAL: More Joins

08-12-22 14:00 Mark & Co FTC 3

1. Once you have run the code in this week's tutor section, write a left join that joins the customer and cust_order tables.

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM customer
LEFT JOIN cust_order ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 ...
(252 rows)
```

2. Write a right join that joins the customer and cust_order tables

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM customer
RIGHT JOIN cust_order ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 Bénédicte        | Dozdill           |          11
 ...
(250 rows)
```

3. write an inner join that joins the customer and cust_order tables.

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM customer
JOIN cust_order ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 ...
(250 rows)
```

4. Write a right join that joins the customer and cust_order tables.

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM customer
RIGHT JOIN cust_order ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 ...
(251 rows)
```

5. Write an inner join that joins the customer and cust_order tables.

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM customer
JOIN cust_order ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 ...
(251 rows)
```

6. Write a left join that joins the customer and cust_order tables.

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM customer
LEFT JOIN cust_order ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 ...
(262 rows)
```

7. Rewrite the query for number 6 but reverse the order of the tables. If you started with the customer table in the query and joined cust_order then rewrite starting with cust_order and join customer.

LANGUAGE: SQL

```
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM cust_order
LEFT JOIN customer ON customer.cust_id=cust_order.cust_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
------------------+-------------------+-------------
 Chadd            | Franz-Schoninger  |           1
 York             | O'Deegan          |           2
 Marie-françoise  | Currier           |           3
 Bérengère        | Menendez          |           4
 Bénédicte        | Dozdill           |           5
 Bénédicte        | Dozdill           |           6
 Chadd            | Franz-Schoninger  |           7
 Bénédicte        | Dozdill           |           8
 Penelope         | Hexter            |           9
 York             | O'Deegan          |          10
 ...
(251 rows)
```

8. Depending on the number of rows that are returned from questions 6 and 7, rewrite the one that has the highest number of results so that the result is sorted firstly by the cust_id and then the cust_ord_id. Copy the query AND THE FIRST SCREEN OF DATA RETURNED BELOW. Make sure you have more than 1 cust_id in the results.

LANGUAGE: SQL

```
-- use query from question 6
SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM cust_order
LEFT JOIN customer ON customer.cust_id=cust_order.cust_id
ORDER BY customer.cust_id, cust_order.cust_ord_id;
```

LANGUAGE: Unknown

```
   cust_fname     |    cust_lname     | cust_ord_id
```

```
-----------------+------------------+-------------
 Jobey           | Boeter           |          26
 Jobey           | Boeter           |          34
 Jobey           | Boeter           |          39
 Jobey           | Boeter           |          57
 Jobey           | Boeter           |          68
 Jobey           | Boeter           |          71
 Jobey           | Boeter           |          77
 Jobey           | Boeter           |          91
 Jobey           | Boeter           |          98
 Jobey           | Boeter           |          99
 Jobey           | Boeter           |         131
 Jobey           | Boeter           |         143
 Jobey           | Boeter           |         146
 York            | O'Deegan         |           2
 York            | O'Deegan         |          10
 York            | O'Deegan         |          19
 ...
(251 rows)
```

9. Write a query that uses outer joins on the customer, the cust_order table and the staff table. It must return the cust_id, cust_ord_id and the staff_id as well as the staff members last name and their work email address.

LANGUAGE: SQL

```
SELECT c.cust_id, co.cust_ord_id, s.staff_id, s.staff_lname, s.work_email FROM customer c
FULL OUTER JOIN cust_order co ON c.cust_id=co.cust_id
FULL OUTER JOIN staff s ON s.staff_id=co.staff_id;
```

LANGUAGE: Unknown

```
 cust_id | cust_ord_id | staff_id | staff_lname |          work_email
---------+-------------+----------+-------------+------------------------------
       4 |           1 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
       2 |           2 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
       6 |           3 |        2 | Shrimpton   | Nikoletta.Shrimpton@dsd.com
       9 |           4 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
       7 |           5 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
       7 |           6 |        4 | Gloster     | Hanan.Gloster@dsd.com
       4 |           7 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
       7 |           8 |        3 | Housegoe    | Montgomery.Housegoe@dsd.com
       3 |           9 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
       2 |          10 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
       7 |          11 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
       9 |          12 |        4 | Gloster     | Hanan.Gloster@dsd.com
       7 |          13 |        4 | Gloster     | Hanan.Gloster@dsd.com
       7 |          14 |        4 | Gloster     | Hanan.Gloster@dsd.com
       6 |          15 |        4 | Gloster     | Hanan.Gloster@dsd.com
       9 |          16 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
      10 |          17 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
       7 |          18 |        3 | Housegoe    | Montgomery.Housegoe@dsd.com
       2 |          19 |        3 | Housegoe    | Montgomery.Housegoe@dsd.com
...
(266 rows)
```

10. Rewrite the query from 9 and filter the results to show only those customers who have not placed an order. (Remember that any customer who has placed an order will have a cust_ord_id associated with them).

LANGUAGE: SQL

```
SELECT c.cust_id, co.cust_ord_id, s.staff_id, s.staff_lname, s.work_email FROM customer c
FULL OUTER JOIN cust_order co ON c.cust_id=co.cust_id
FULL OUTER JOIN staff s ON s.staff_id=co.staff_id
WHERE co.cust_ord_id IS NULL AND c.cust_id IS NOT NULL;
```

LANGUAGE: Unknown

```
 cust_id | cust_ord_id | staff_id | staff_lname | work_email
---------+-------------+----------+-------------+------------
      25 |             |          |             |
      27 |             |          |             |
      33 |             |          |             |
      31 |             |          |             |
      34 |             |          |             |
      32 |             |          |             |
      24 |             |          |             |
      28 |             |          |             |
      30 |             |          |             |
      29 |             |          |             |
      35 |             |          |             |
(11 rows)
```

11. Write a query that will display the staff first and last names, their work email addresses, the customer order id, the customer id and the customer's first and last names along with the products that are in the customer's orders. The results must be ordered by customer last name order. Copy the query AND THE FIRST SCREEN OF DATA RETURNED BELOW. (Make sure you have more than 1 customer in the results).

LANGUAGE: SQL

```
SELECT s.staff_fname, s.staff_lname, s.work_email, co.cust_ord_id, c.cust_id, c.cust_fname, c.
    ↪ cust_lname, p.prod_name FROM customer c
JOIN cust_order co ON c.cust_id=co.cust_id
JOIN staff s ON s.staff_id=co.staff_id
JOIN manifest ON manifest.cust_ord_id = co.cust_ord_id
JOIN product p ON p.prod_id = manifest.prod_id
ORDER BY c.cust_lname;
```

LANGUAGE: Unknown

```
 staff_fname | staff_lname |         work_email          | cust_ord_id | cust_id |   cust_fname
    ↪      |    cust_lname    |                    prod_name
-------------+-------------+-----------------------------+-------------+---------+-----------------+----------
    ↪
 Hanan       | Gloster     | Hanan.Gloster@dsd.com       |          39 |       1 | Jobey
    ↪      | Boeter           | Switchable tangible product
 Nikoletta   | Shrimpton   | Nikoletta.Shrimpton@dsd.com |          57 |       1 | Jobey
    ↪      | Boeter           | Persistent demand-driven complexity
 Montgomery  | Housegoe    | Montgomery.Housegoe@dsd.com |          68 |       1 | Jobey
    ↪      | Boeter           | Streamlined asynchronous functionalities
 Aura        | Clewlowe    | Aura.Clewlowe@dsd.com       |         131 |       1 | Jobey
    ↪      | Boeter           | Seamless optimal leverage
 Janeva      | Gillicuddy  | Janeva.Gillicuddy@dsd.com   |          99 |       1 | Jobey
    ↪      | Boeter           | Fundamental global archive
 Hanan       | Gloster     | Hanan.Gloster@dsd.com       |          34 |       1 | Jobey
    ↪      | Boeter           | Right-sized mission-critical pricing structure
 Montgomery  | Housegoe    | Montgomery.Housegoe@dsd.com |          26 |       1 | Jobey
    ↪      | Boeter           | Switchable tangible product
 Hanan       | Gloster     | Hanan.Gloster@dsd.com       |          77 |       1 | Jobey
    ↪      | Boeter           | Realigned homogeneous hub
 Montgomery  | Housegoe    | Montgomery.Housegoe@dsd.com |         146 |       1 | Jobey
    ↪      | Boeter           | Fundamental global archive
 Janeva      | Gillicuddy  | Janeva.Gillicuddy@dsd.com   |         143 |       1 | Jobey
    ↪      | Boeter           | Re-engineered cohesive methodology
 Niel        | Welsby      | Niel.Welsby@dsd.com         |          91 |       1 | Jobey
    ↪      | Boeter           | Configurable analyzing solution
 Nikoletta   | Shrimpton   | Nikoletta.Shrimpton@dsd.com |          71 |       1 | Jobey
    ↪      | Boeter           | Inverse high-level attitude
 Montgomery  | Housegoe    | Montgomery.Housegoe@dsd.com |          98 |       1 | Jobey
    ↪      | Boeter           | Distributed uniform Graphic Interface
 Niel        | Welsby      | Niel.Welsby@dsd.com         |         112 |       6 | Marie-
    ↪ françoise | Currier          | Integrated 24/7 interface
...
(150 rows)
```

12. Write a query that will show only the customer contact details who have NEVER placed an order. It is up to you to decide what we mean by contact details. Copy the output and query below.

LANGUAGE: SQL

```sql
SELECT c.cust_fname, c.email FROM customer c
FULL OUTER JOIN cust_order co ON c.cust_id = co.cust_id
WHERE co.cust_ord_id IS NULL;
```

LANGUAGE: Unknown

```
 cust_fname |               email
------------+-----------------------------------
 Jen        | jsettle222@google.ca
 Fawnia     | fpetchell1@networkadvertising.org
 Nealy      | nstanley7@arstechnica.com
 Tine       | tclopton5@typepad.com
 Cody       | clago8@rambler.ru
 Lonnie     | lmacgilpatrick6@uiuc.edu
 Evie       | 3vi3@google.wh
 Mireielle  | mkillner2@cafepress.com
 Falkner    | fgrouer4@dion.ne.jp
 Kaine      | klawford3@imdb.com
 Theadora   | tajsik9@sfgate.com
(11 rows)
```

LECTUER: Christmas Lecture

15-12-22 13:20 Mark RB LT1

Regardless of the scenario, we have to start with picking out the entities for the Entity Relationship Diagram.
If there is something which happens to an entity, for example a service, then if you store that data in the entity, you won't be able to view information about that event once it is overwritten. You have to store the event in a different table.
There should never be entities which are not connected/ related to any other entities in the ERD.

Coursework Advice

If you have 20-30 entities, you've broken down the coursework too much. Somewhere between 6 and 11 is the right number.