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

Page 1

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.

Page 2

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

```
1  \l
```

LANGUAGE: Unknown

```
1                          List of databases
2     Name       |     Owner      | Encoding | Collate | Ctype   |  Access privileges
3  ---------------+---------------+----------+---------+---------+----------------------
4  dsd_22         | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
5  mongo-2021-fix | mongo-2021-fix | UTF8    | C.UTF-8 | C.UTF-8 |
6  postgres       | postgres      | UTF8     | C.UTF-8 | C.UTF-8 |
7  template0      | postgres      | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres        +
8                 |               |          |         |         | postgres=CTc/postgres
9  template1      | postgres      | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres        +
10                |               |          |         |         | postgres=CTc/postgres
11 up2108121      | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
12 (6 rows)
```

2. Connect to the database

LANGUAGE: SQL

```
1  \c dsd_22
```

LANGUAGE: Unknown

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

3. List everything in this database

LANGUAGE: SQL

```
1  \d
```

LANGUAGE: Unknown

```
1                  List of relations
2  Schema |            Name            |   Type   |   Owner
3  --------+----------------------------+----------+-----------
4  public | category                   | table    | up2108121
5  public | category_cat_id_seq        | sequence | up2108121
6  public | cust_order                 | table    | up2108121
7  public | cust_order_cust_ord_id_seq | sequence | up2108121
8  public | customer                   | table    | up2108121
9  public | customer_cust_id_seq       | sequence | up2108121
10 public | manifest                   | table    | up2108121
11 public | manifest_manifest_id_seq   | sequence | up2108121
12 public | product                    | table    | up2108121
13 public | product_prod_id_seq        | sequence | up2108121
14 public | role                       | table    | up2108121
15 public | role_role_id_seq           | sequence | up2108121
16 public | staff                      | table    | up2108121
17 public | staff_staff_id_seq         | sequence | up2108121
18 (14 rows)
```

4. List just the tables

LANGUAGE: SQL

```
1  \dt
```

LANGUAGE: Unknown

```
1           List of relations
2  Schema |    Name    | Type  |   Owner
3  --------+------------+-------+-----------
4  public | category   | table | up2108121
5  public | cust_order | table | up2108121
6  public | customer   | table | up2108121
7  public | manifest   | table | up2108121
8  public | product    | table | up2108121
9  public | role       | table | up2108121
10 public | staff      | table | up2108121
11 (7 rows)
```

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

LANGUAGE: SQL

```
1  SELECT * FROM customer;
```

LANGUAGE: Unknown

```
1  cust_id |   cust_fname   |    cust_lname    |        addr1        |    addr2     |    town
       ↪   | postcode  |            email
2  --------+----------------+------------------+---------------------+--------------+
       ↪ ----------------+-----------+--------------------------------
3        1 | Jobey          | Boeter           | 6 Claremont Park    | Truax        | La
   ↪ Mohammedia  | CV42 3EF  | jboeter0@mail.ru
4        2 | York           | O'Deegan         | 882 Hooker Trail    |              | Chemnitz
   ↪         | YA92 2OJ  | yodeegan1@nydailynews.com
5        3 | Penelope       | Hexter           | 25 Jackson Lane     |              | Pingshan
   ↪         | LY32 8LN  | phexter2@cbslocal.com
6        4 | Chadd          | Franz-Schoninger | 7 Division Point    | Texas        | Baojia
   ↪         | XA22 0UR  | cfranzschoninger3@google.com.hk
7        5 | Vikky          | Eke              | 293 Colorado Drive  | Browning     | Kamenny
   ↪ Privoz | WQ12 3SF  | veke4@elegantthemes.com
8        6 | Marie-francoise | Currier         | 032 Eagan Junction  | Duke         |
   ↪ Waekolong      | NB52 4MV  | acurrier0@economist.com
9        7 | Benedicte      | Dozdill          | 579 Dryden Terrace  |              | Dawuhan
   ↪         | GY32 6GQ  | cdozdill1@amazon.de
10       8 | Gorel          | Douthwaite       | 2946 Bluejay Parkway | Heath       | Sunbu
   ↪         | PH02 3ZX  | edouthwaite2@feedburner.com
11       9 | Berengere      | Menendez         | 06154 Jackson Way   | Doe Crossing |
   ↪ Tsagaanders    | HO82 5XL  | amenendez3@dell.com
12      10 | Pelagie        | Hachard          | 1777 Hauk Center    |              | Jiantou
   ↪         | NA52 4LM  | fhachard4@blinklist.com
13      11 | Adaobi         | Musa             | 6 Clariss Ave       |              | La
   ↪ Mohammedia  | CV4 3F    | amusa9@mail.ca
14 (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

```
1  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)
```

Page 3

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.

```sql
LANGUAGE: SQL
1 DELETE
2 INSERT
3 REPLACE
4 SELECT
5 UPDATE
```

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

```sql
LANGUAGE: SQL
1 select property_id,
2 street,
3 city,
4 postcode,
5 owner_id from property_for_rent
6 where city = 'Glasgow';
```

Page 4

PRACTICAL: Further Introduction

📅 06-10-22   🕑 14:00   🎓 Mark & team   📍 FTC Floor 3

Introductory Tasks

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

```sql
\l
```

```
    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.

```sql
\c dsd_22
```

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

3. List the contents of the database

```sql
\d
```

```
                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
```

```
8  public | customer                | table    | up2108121
9  public | customer_cust_id_seq    | sequence | up2108121
10 public | manifest                | table    | up2108121
11 public | manifest_manifest_id_seq | sequence | up2108121
12 public | product                 | table    | up2108121
13 public | product_prod_id_seq     | sequence | up2108121
14 public | role                    | table    | up2108121
15 public | role_role_id_seq        | sequence | up2108121
16 public | staff                   | table    | up2108121
17 public | staff_staff_id_seq      | sequence | up2108121
18 (14 rows)
```

5. List just the tables

LANGUAGE: SQL

```
1  \dt
```

LANGUAGE: Unknown

```
1           List of relations
2  Schema |    Name    | Type  |   Owner
3  --------+------------+-------+-----------
4  public | category   | table | up2108121
5  public | cust_order | table | up2108121
6  public | customer   | table | up2108121
7  public | manifest   | table | up2108121
8  public | product    | table | up2108121
9  public | role       | table | up2108121
10 public | staff      | table | up2108121
11 (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

```
1  \d role
```

LANGUAGE: Unknown

```
1  Table "public.role"
2  Column  |         Type          | Collation | Nullable |                Default
3  ----------+----------------------+-----------+----------+-------------------------------------------
      ↪
4  role_id  | integer               |           | not null | nextval('role_role_id_seq'::regclass
      ↪ )
5  role_name | character varying(20) |           |          |
6  Indexes:
7      "role_pkey" PRIMARY KEY, btree (role_id)
8  Referenced by:
9      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
```sql
1 SELECT * FROM CATEGORY;
```

> LANGUAGE: Unknown
```
1 cat_id |  cat_name
2 --------+-------------
3     1 | Men's Wear
4     2 | Ladies Wear
5     3 | Kid's Wear
6     4 | Outdoor
7     5 | Sport
8     6 | Health
9 (6 rows)
```

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

> LANGUAGE: SQL
```sql
1 select * from category;
```

> LANGUAGE: Unknown
```
1 cat_id |  cat_name
2 --------+-------------
3     1 | Men's Wear
4     2 | Ladies Wear
5     3 | Kid's Wear
6     4 | Outdoor
7     5 | Sport
8     6 | Health
9 (6 rows)
```

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

> LANGUAGE: SQL
```sql
1 select * from 'Category';
```

> LANGUAGE: Unknown
```
1 ERROR:  syntax error at or near "'Category'"
2 LINE 1: select * from 'Category';
3                       ^
```

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

> LANGUAGE: SQL
```sql
1 select * from "Category";
```

> LANGUAGE: Unknown
```
1 ERROR:  relation "Category" does not exist
2 LINE 1: select * from "Category";
3                       ^
```

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

```
LANGUAGE: SQL
1  select * from 'category';
```

```
LANGUAGE: Unknown
1  ERROR:  syntax error at or near "'category'"
2  LINE 1: select * from 'category';
3                        ^
```

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

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

```
LANGUAGE: Unknown
1  cat_id |   cat_name
2  -------+-------------
3      1 | Men's Wear
4      2 | Ladies Wear
5      3 | Kid's Wear
6      4 | Outdoor
7      5 | Sport
8      6 | Health
9  (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
1  SELECT * FROM "category";
```

```
LANGUAGE: Unknown
1  ERROR:  relation ""category"" does not exist
2  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
1  \d customer
```

```
LANGUAGE: Unknown
1      Table "public.customer"
2      Column    |         Type          | Collation | Nullable |            Default
3
    ↪ -----------+-----------------------+-----------+----------+--------------------------------
    ↪
4  cust_id    | integer               |           | not null | nextval('customer_cust_id_seq'::
    ↪ regclass)
5  cust_fname | character varying(25) |           | not null |
6  cust_lname | character varying(35) |           | not null |
7  addr1      | character varying(50) |           | not null |
8  addr2      | character varying(50) |           |          |
9  town       | character varying(60) |           | not null |
10 postcode   | character(9)          |           | not null |
11 email      | character varying(255)|           | not null |
12 Indexes:
13     "customer_pkey" PRIMARY KEY, btree (cust_id)
14 Referenced by:
15     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.

```sql
LANGUAGE: SQL
1 CREATE TABLE tableName(
2 attributeName dataType (options),
3 attributeName dataType (options),
4 ...
5 );
```

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).

```sql
LANGUAGE: SQL
1 CREATE DATABASE week02;
2
3 CREATE TABLE NEWTABLE(
4 IAMNUMBER INT PRIMARY KEY,
5 IAMSTRING VARCHAR(10)
6 );
```

Now, insert a record into the table.

```sql
LANGUAGE: SQL
1 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.
```

Page 5

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.

Page 6

PRACTICAL: Count()

📅 13-10-22   🕐 14:00   🎓   Mark   and   📍 FTC Floor 3
                                                        team

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
1 dsd_22=# select count(*) from category;
2 count
3 -------
4      6
5 (1 row)
6 dsd_22=# select count(*) from cust_order;
7 count
8 -------
9 150
10 (1 row)
11 dsd_22=# select count(*) from customer;
12 count
13 -------
14 11
15 (1 row)
16 dsd_22=# select count(*) from manifest;
17 count
18 -------
19 150
20 (1 row)
21
22 dsd_22=# select count(*) from product;
23 count
24 -------
25 100
26 (1 row)
27
28 dsd_22=# select count(*) from role;
29 count
30 -------
31      5
32 (1 row)
33
34 dsd_22=# select count(*) from staff;
35 count
36 -------
37 10
38 (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

```sql
LANGUAGE: SQL
1 select max(role_id) from role;
```

```
LANGUAGE: Unknown
1 max
2 -----
3     5
4 (1 row)
```

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

```
LANGUAGE: SQL
1 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
1 select count(*) from role;
```

```
LANGUAGE: Unknown
1 count
2 -------
3     6
4 (1 row)
```

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

```
LANGUAGE: SQL
1 select max(role_id) from role;
```

```
LANGUAGE: Unknown
1 max
2 -----
3     6
4 (1 row)
```

Q6. Delete this new row with

```
LANGUAGE: SQL
1 DELETE FROM ROLE WHERE ROLE_NAME = 'Pre Sales';
```

```
LANGUAGE: Unknown
1 DELETE 1
```

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

```
LANGUAGE: Unknown
1 count
2 -------
3     5
4 (1 row)
```

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

```
LANGUAGE: Unknown
1 max
2 -----
3     5
4 (1 row)
```

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

```
LANGUAGE: SQL
1 INSERT INTO ROLE (ROLE_NAME) VALUES ('Cleaning Team');
```

```
LANGUAGE: Unknown
1 INSERT 0 1
```

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

```
LANGUAGE: Unknown
1 count
2 -------
3       6
4 (1 row)
```

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

```
LANGUAGE: Unknown
1 max
2 -----
3     6
4 (1 row)
```

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

```
LANGUAGE: SQL
1 SELECT RANDOM();
```

```
LANGUAGE: Unknown
1       random
2 -------------------
3  0.175315219908953
4 (1 row)
```

Q12a. Create another random number. Copy the value below

```
LANGUAGE: SQL
1 SELECT RANDOM();
```

```
LANGUAGE: Unknown
1       random
2 -------------------
3  0.272884896956384
4 (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.

```
LANGUAGE: Unknown

1  dsd_22=# select random()*11;
2        ?column?
3  ------------------
4   9.60335403773934
5  (1 row)
6  dsd_22=# select random()*11;
7        ?column?
8  -------------------
9   0.160588529892266
10 (1 row)
11
12 dsd_22=# select random()*11;
13        ?column?
14 ------------------
15  5.25661591161042
16 (1 row)
17
18 dsd_22=# select random()*11;
19        ?column?
20 ------------------
21  7.78145408304408
22 (1 row)
23 dsd_22=# select random()*11;
24        ?column?
25 ------------------
26  10.1819118564017
27 (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.

```sql
LANGUAGE: SQL

1  create table numb1(numb_id int primary key, ran_val decimal(17,15));
2
3  insert into numb1(numb_id, ran_val) values
4  (1,random()),(2,random()),(3,random()),(4,random()),(5,random()),(6,random()),(7,random()),(8,
      ↪ random()),(9,random()),(10,random());
```

```
LANGUAGE: Unknown

1      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

1   count
2  -------
3      10
4  (1 row)
```

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

```
LANGUAGE: Unknown

1   numb_id |       ran_val
2  ---------+-------------------
3         1 | 0.481754711363465
```

```
4        2 | 0.020102311857045
5        3 | 0.541421711910516
6        4 | 0.046512784436345
7        5 | 0.842869907151908
8        6 | 0.137599688488990
9        7 | 0.925696460530162
10       8 | 0.765472991392016
11       9 | 0.712954005226493
12      10 | 0.161490791942924
13 (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

1 test_num=# SELECT * FROM NUMB1;
2  numb_id |      ran_val
3 ---------+------------------
4      1 | 0.477631121408194
5      2 | 0.978080025874078
6      3 | 0.516494689509273
7      4 | 0.849129045847803
8      5 | 0.484937957022339
9      6 | 0.895700289402157
10     7 | 0.852438564877957
11     8 | 0.727535046637058
12     9 | 0.062769805546850
13    10 | 0.594313766807318
14 (10 rows)
```

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

```
LANGUAGE: SQL

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

```
LANGUAGE: Unknown

1        max
2 ------------------
3  0.925696460530162
4 (1 row)
```

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

```
LANGUAGE: SQL

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

```
LANGUAGE: Unknown

1        min
2 ------------------
3  0.020102311857045
4 (1 row)
```

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

```
LANGUAGE: SQL

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

```
LANGUAGE: Unknown

1            avg
2  -----------------------
3   0.46358753642998640000
4  (1 row)
```

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

```
LANGUAGE: SQL

1  select now();
```

```
LANGUAGE: Unknown

1              now
2  ------------------------------
3   2022-10-13 13:43:49.196518+00
4  (1 row)
```

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

```
LANGUAGE: SQL

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

```
LANGUAGE: Unknown

1   cust_fname
2  ------------
3   Penelope
4  (1 row)
```

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

```
LANGUAGE: SQL

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

```
LANGUAGE: Unknown

1   cat_id
2  --------
3        4
4  (1 row)
```

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

```
LANGUAGE: SQL

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

```
LANGUAGE: Unknown
1   count
2  -------
3       0
4  (1 row)
```

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

```
LANGUAGE: SQL
1  select staff_fname, staff_lname from staff where town='Portsmouth';
```

```
LANGUAGE: Unknown
1   staff_fname | staff_lname
2  -------------+-------------
3   Niel        | Welsby
4   Janeva      | Gillicuddy
5  (2 rows)
```

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

```
LANGUAGE: SQL
1      select addr1 , addr2 from staff where staff_id=4;
```

```
LANGUAGE: Unknown
1        addr1        | addr2
2  ------------------+-------
3   959 Algoma Plaza |
4  (1 row)
```

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

```
LANGUAGE: SQL
1  select count(*) from staff where role=3;
```

```
LANGUAGE: Unknown
1   count
2  -------
3       3
4  (1 row)
```

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

```
LANGUAGE: SQL
1  select count(*) from product where prod_id=2;
```

```
LANGUAGE: Unknown
1   count
2  -------
```

```
3        1
4 (1 row)
```

Page 7

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.
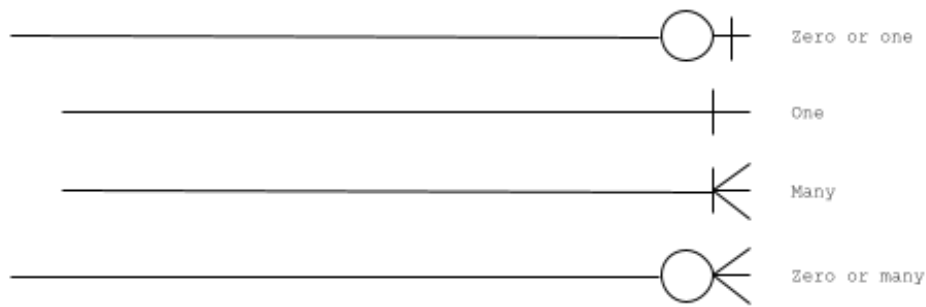
Crow's feet notation

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.

Page 8

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.

```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.

```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

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

Insert more records, some of these return errors.

```
LANGUAGE: SQL
1 UPDATE customer SET cust_email = 'i_love_elephants_1@gmail.com' where cust_id = 66;
2 SELECT * FROM customer;
3
4 INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (67,'Alia', 'Anand','
  ↪ aanand98@gmail.com');
5
6 INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (70,'Connor', 'Murphy
  ↪ ','connormurphy99@gmail.com');
7
8 INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (71,'Connor', 'Murphy
  ↪ ','connormurphy199@gmail.com');
9
10 INSERT INTO customer (cust_id, cust_fname, cust_lname, cust_email) VALUES (1,'Tim', 'Nice-but-
   ↪ Dimm','alongemailaddresswillfit@quitealongdomainnametoo.com');
11
12 INSERT INTO customer ( cust_fname, cust_lname, cust_email) VALUES ('Tim', 'Nice-but-Dimm','
   ↪ averylongemailaddresswillnotfit111111111111111111111111111111111111111a@quitealongdomainnametoo
   ↪ .com');
```

Task 2: Write SQL code for the following questions.

1. Create a new database called `code_test`

   ```
   LANGUAGE: SQL
   1 CREATE DATABASE code_test;
   ```

2. Connect to this new database

   ```
   LANGUAGE: SQL
   1 \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
   1 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
   1 \d table_one
   ```

```
LANGUAGE: Unknown
1                     Table "public.table_one"
2   Column   |          Type           | Collation | Nullable | Default
3 -----------+-------------------------+-----------+----------+---------
4  record_id | integer                 |           | not null |
5  att_1     | character varying(30)   |           |          |
6  att_2     | character(10)           |           |          |
7  att_3     | numeric(3,2)            |           |          |
8 Indexes:
9     "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
1 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
1 \d table_one
```

```
LANGUAGE: Unknown
1                     Table "public.table_one"
2   Column   |          Type           | Collation | Nullable | Default
3 -----------+-------------------------+-----------+----------+---------
4  record_id | integer                 |           | not null |
5  att_1     | character varying(30)   |           |          |
6  att_2     | character(10)           |           |          |
7  att_3     | numeric(3,2)            |           |          |
8  att_4     | integer                 |           |          |
9 Indexes:
10    "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
1 INSERT INTO table_one (Record_id, Att_1, Att_2, Att_3, Att_4) VALUES (1, 'continent', '0
    ↪ olP[dollarSign]fguj', 9.99, 42);
2 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
1 SELECT * FROM table_one;
```

9. Get a screenshot of the data

```
LANGUAGE: Unknown
1  record_id |          att_1         |    att_2    | att_3 | att_4
2 -----------+------------------------+-------------+-------+-------
3          1 | continent              | OolP[dollarSign]fguj |  9.99 |    42
4          2 | Portsmouth University  | Violet      |  9.99 | 9999
5 (2 rows)
```

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

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

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

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

12. Get a screenshot of the results.

```
LANGUAGE: Unknown
1  record_id |   att_1   |    att_2    | att_3 | att_4
2 -----------+-----------+-------------+-------+-------
3          1 | continent | OolP[dollarSign]fguj |  9.99 |    66
4 (1 row)
```

Page 9

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.

Page 10

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.

Page 11

PRACTICAL: Keys & Joins

📅 10-11-22   ⏰ 14:00   🎓 Mark and   📍 FTC Floor 3
                                                              team

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.

```
LANGUAGE: Unknown
1  SELECT COUNT(*) FROM category;
2   count
3  -------
4       6
5  (1 row)
6
7  SELECT COUNT(*) FROM cust_order;
8   count
9  -------
10    150
11 (1 row)
12
13 SELECT COUNT(*) FROM customer;
14  count
15 -------
16     11
17 (1 row)
18
19 SELECT COUNT(*) FROM manifest;
20  count
21 -------
22    150
23 (1 row)
24
25 SELECT COUNT(*) FROM product;
26  count
27 -------
28    100
29 (1 row)
30
31 SELECT COUNT(*) FROM role;
32  count
33 -------
34      5
35 (1 row)
36
37 SELECT COUNT(*) FROM staff;
38  count
39 -------
40     10
41 (1 row)
```

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

```
LANGUAGE: Unknown
1  \d category
2      Table "public.category"
3  Column  |          Type           | Collation | Nullable |                Default
4
5  ----------+----------------------+-----------+----------+-------------------------------------------
   ↪
6  cat_id   | integer              |           | not null | nextval('category_cat_id_seq'::
   ↪ regclass)
7  cat_name | character varying(40) |           |          |
8  Indexes:
9  "category_pkey" PRIMARY KEY, btree (cat_id)
10 Referenced by:
11 TABLE "product" CONSTRAINT "product_prod_cat_fkey" FOREIGN KEY (prod_cat) REFERENCES
   ↪ category(cat_id)
12
13 \d cust_order
14  Table "public.cust_order"
15 Column    | Type  | Collation | Nullable |                Default
16 ------------+--------+-----------+----------+---------------------------------------------
   ↪
17 cust_ord_id | integer |          | not null | nextval('cust_order_cust_ord_id_seq'::
   ↪ regclass)
18 staff_id    | integer |          |          |
19 cust_id     | integer |          |          |
20 Indexes:
```

```
21 "cust_order_pkey" PRIMARY KEY, btree (cust_ord_id)
22 Foreign-key constraints:
23 "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
24 "cust_order_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
25 Referenced by:
26 TABLE "manifest" CONSTRAINT "manifest_cust_ord_id_fkey" FOREIGN KEY (cust_ord_id)
       ↪ REFERENCES cust_order(cust_ord_id)
27
28 \d customer
29        Table "public.customer"
30 Column  |           Type          | Collation | Nullable |          Default
31
32 -----------+----------------------+-----------+----------+---------------------------------------
       ↪
33 cust_id    | integer               |           | not null | nextval('customer_cust_id_seq
       ↪ '::regclass)
34 cust_fname | character varying(25)  |           | not null |
35 cust_lname | character varying(35)  |           | not null |
36 addr1      | character varying(50)  |           | not null |
37 addr2      | character varying(50)  |           |          |
38 town       | character varying(60)  |           | not null |
39 postcode   | character(9)           |           | not null |
40 email      | character varying(255) |           | not null |
41 Indexes:
42 "customer_pkey" PRIMARY KEY, btree (cust_id)
43 Referenced by:
44 TABLE "cust_order" CONSTRAINT "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES
       ↪ customer(cust_id)
45
46 \d manifest
47  Table "public.manifest"
48 Column   | Type  | Collation | Nullable |               Default
49 -------------+---------+-----------+----------+-----------------------------------------------
       ↪
50 manifest_id | integer |           | not null | nextval('manifest_manifest_id_seq'::
       ↪ regclass)
51 cust_ord_id | integer |           | not null |
52 prod_id     | integer |           | not null |
53 Indexes:
54 "manifest_pkey" PRIMARY KEY, btree (manifest_id)
55 Foreign-key constraints:
56 "manifest_cust_ord_id_fkey" FOREIGN KEY (cust_ord_id) REFERENCES cust_order(cust_ord_id)
57 "manifest_prod_id_fkey" FOREIGN KEY (prod_id) REFERENCES product(prod_id)
58
59
60 \d product
61      Table "public.product"
62 Column  |           Type          | Collation | Nullable |          Default
63
64 -----------+----------------------+-----------+----------+-----------------------------------
       ↪
65 prod_id    | integer               |           | not null | nextval('product_prod_id_seq'::
       ↪ regclass)
66 prod_name  | character varying(50) |           | not null |
67 prod_cat   | integer               |           | not null |
68 Indexes:
69 "product_pkey" PRIMARY KEY, btree (prod_id)
70 Foreign-key constraints:
71 "product_prod_cat_fkey" FOREIGN KEY (prod_cat) REFERENCES category(cat_id)
72 Referenced by:
73 TABLE "manifest" CONSTRAINT "manifest_prod_id_fkey" FOREIGN KEY (prod_id) REFERENCES
       ↪ product(prod_id)
74
75 \d role
76      Table "public.role"
77 Column  |           Type          | Collation | Nullable |          Default
78
79 -----------+----------------------+-----------+----------+----------------------------------
       ↪
80 role_id    | integer               |           | not null | nextval('role_role_id_seq'::
       ↪ regclass)
81 role_name | character varying(20)  |           |          |
82 Indexes:
83 "role_pkey" PRIMARY KEY, btree (role_id)
84 Referenced by:
```

```
85  TABLE "staff" CONSTRAINT "staff_role_fkey" FOREIGN KEY (role) REFERENCES role(role_id)
86
87
88
89  \d staff
90
91      Table "public.staff"
92  Column    |         Type          | Collation | Nullable |              Default
93
94  ------------+-----------------------+-----------+----------+-----------------------------------
       ↪
95  staff_id   | integer               |           | not null | nextval('staff_staff_id_seq
       ↪ '::regclass)
96  staff_fname | character varying(25) |           | not null |
97  staff_lname | character varying(35) |           | not null |
98  addr1      | character varying(50) |           | not null |
99  addr2      | character varying(50) |           |          |
100 town       | character varying(60) |           | not null |
101 postcode   | character(9)          |           | not null |
102 home_email | character varying(255)|           | not null |
103 work_email | character varying(100)|           | not null |
104 role       | integer               |           | not null |
105 Indexes:
106 "staff_pkey" PRIMARY KEY, btree (staff_id)
107 Foreign-key constraints:
108 "staff_role_fkey" FOREIGN KEY (role) REFERENCES role(role_id)
109 Referenced by:
110 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

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

LANGUAGE: Unknown

```
1
2   prod_id |                 prod_name                | prod_cat | cat_id |
     ↪ cat_name
3  ---------+------------------------------------------+----------+--------+-------------
     ↪
4       1 | Multi-layered multi-tasking initiative   |        2 |      1 | Men's
     ↪ Wear
5       2 | Operative analyzing task-force           |        1 |      1 | Men's
     ↪ Wear
6       3 | Exclusive client-server array            |        5 |      1 | Men's
     ↪ Wear
7       4 | Balanced client-server product           |        6 |      1 | Men's
     ↪ Wear
8       5 | Exclusive background website             |        5 |      1 | Men's
     ↪ Wear
9       6 | Pre-emptive holistic intranet            |        6 |      1 | Men's
     ↪ Wear
10      7 | Re-engineered cohesive methodology       |        1 |      1 | Men's
     ↪ Wear
11      8 | Robust directional projection            |        2 |      1 | Men's
     ↪ Wear
12      9 | Inverse transitional infrastructure      |        4 |      1 | Men's
     ↪ Wear
13     10 | Multi-tiered explicit paradigm           |        6 |      1 | Men's
     ↪ Wear
14 ...
15 (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

```sql
LANGUAGE: SQL
1  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

```sql
LANGUAGE: SQL
1  select * from category
2  join product on category.cat_id = product.prod_cat;
```

```
LANGUAGE: Unknown
1   cat_id |   cat_name   | prod_id |                      prod_name                      |
    ↪ prod_cat
2  --------+--------------+---------+-----------------------------------------------------+----------
    ↪
3       2 | Ladies Wear  |       1 | Multi-layered multi-tasking initiative              |
    ↪    2
4       1 | Men's Wear   |       2 | Operative analyzing task-force                      |
    ↪    1
5       5 | Sport        |       3 | Exclusive client-server array                       |
    ↪    5
6       6 | Health       |       4 | Balanced client-server product                      |
    ↪    6
7       5 | Sport        |       5 | Exclusive background website                        |
    ↪    5
8       6 | Health       |       6 | Pre-emptive holistic intranet                       |
    ↪    6
9       1 | Men's Wear   |       7 | Re-engineered cohesive methodology                  |
    ↪    1
10      2 | Ladies Wear  |       8 | Robust directional projection                       |
    ↪    2
11      4 | Outdoor      |       9 | Inverse transitional infrastructure                 |
    ↪    4
12      6 | Health       |      10 | Multi-tiered explicit paradigm                      |
    ↪    6
13 ...
14 (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"

```sql
LANGUAGE: SQL
1  select * from category
2  join product on category.cat_id = product.prod_cat
3  where prod_name = 'Multi-layered multi-tasking initiative';
```

```
LANGUAGE: Unknown
1  cat_id |  cat_name   | prod_id |                prod_name                | prod_cat
2 --------+-------------+---------+-----------------------------------------+----------
3       2 | Ladies Wear |       1 | Multi-layered multi-tasking initiative  |        2
4 (1 row)
```

20. Run the following code

```
LANGUAGE: SQL
1 select count(*) from customer, cust_order;
```

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

```
LANGUAGE: Unknown
1  count
2 -------
3   1650
4 (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
1 select customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id from customer
2 join cust_order on customer.cust_id = cust_order.cust_id
3 where cust_order.cust_id = 1;
```

```
LANGUAGE: Unknown
1  cust_fname | cust_lname | cust_ord_id
2 ------------+------------+-------------
3  Jobey      | Boeter     |          26
4  Jobey      | Boeter     |          34
5  Jobey      | Boeter     |          39
6  Jobey      | Boeter     |          57
7  Jobey      | Boeter     |          68
8  Jobey      | Boeter     |          71
9  Jobey      | Boeter     |          77
10 Jobey      | Boeter     |          91
11 Jobey      | Boeter     |          98
12 Jobey      | Boeter     |          99
13 Jobey      | Boeter     |         131
14 Jobey      | Boeter     |         143
15 Jobey      | Boeter     |         146
16 (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
1 select customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id, staff.staff_fname
   ↪ , staff.staff_lname from customer
2 join cust_order on customer.cust_id = cust_order.cust_id
3 join staff on cust_order.staff_id = staff.staff_id
4 where cust_order.cust_id = 1;
```

```
LANGUAGE: Unknown
1  cust_fname | cust_lname | cust_ord_id | staff_fname | staff_lname
2  ------------+------------+-------------+-------------+-------------
3  Jobey      | Boeter     |          26 | Montgomery  | Housegoe
4  Jobey      | Boeter     |          34 | Hanan       | Gloster
5  Jobey      | Boeter     |          39 | Hanan       | Gloster
6  Jobey      | Boeter     |          57 | Nikoletta   | Shrimpton
7  Jobey      | Boeter     |          68 | Montgomery  | Housegoe
8  Jobey      | Boeter     |          71 | Nikoletta   | Shrimpton
9  Jobey      | Boeter     |          77 | Hanan       | Gloster
10 Jobey      | Boeter     |          91 | Niel        | Welsby
11 Jobey      | Boeter     |          98 | Montgomery  | Housegoe
12 Jobey      | Boeter     |          99 | Janeva      | Gillicuddy
13 Jobey      | Boeter     |         131 | Aura        | Clewlowe
14 Jobey      | Boeter     |         143 | Janeva      | Gillicuddy
15 Jobey      | Boeter     |         146 | Montgomery  | Housegoe
16 (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.

```sql
1 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
2 join cust_order on customer.cust_id = cust_order.cust_id
3 join staff on cust_order.staff_id = staff.staff_id
4 join role on staff.role = role.role_id
5 where customer.cust_id = 4;
```

```
LANGUAGE: Unknown
1  cust_fname |     cust_lname    | cust_ord_id | staff_fname | staff_lname | role_id |
      ↪ role_name
2  ------------+------------------+-------------+-------------+------------+--------+----------------
      ↪
3  Chadd      | Franz-Schoninger |           1 | Aura        | Clewlowe    |       3 | Post
      ↪ Sales
4  Chadd      | Franz-Schoninger |           7 | Aura        | Clewlowe    |       3 | Post
      ↪ Sales
5  Chadd      | Franz-Schoninger |          66 | Montgomery  | Housegoe    |       1 | Order
      ↪  Picker
6  Chadd      | Franz-Schoninger |          81 | Janeva      | Gillicuddy  |       5 | Misc
7  Chadd      | Franz-Schoninger |          93 | Niel        | Welsby      |       2 | Final
      ↪  Packer
8  Chadd      | Franz-Schoninger |          97 | Aura        | Clewlowe    |       3 | Post
      ↪ Sales
9  Chadd      | Franz-Schoninger |         107 | Hanan       | Gloster     |       4 |
      ↪ Customer Retain
10 Chadd      | Franz-Schoninger |         109 | Nikoletta   | Shrimpton   |       4 |
      ↪ Customer Retain
11 Chadd      | Franz-Schoninger |         124 | Aura        | Clewlowe    |       3 | Post
      ↪ Sales
12 Chadd      | Franz-Schoninger |         129 | Nikoletta   | Shrimpton   |       4 |
      ↪ Customer Retain
13 (10 rows)
```

Page 12

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
LANGUAGE: 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
LANGUAGE: 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.

LANGUAGE: SQL

```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.

LANGUAGE: SQL

```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.

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

Page 13

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.

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

```
   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

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

LANGUAGE: Unknown

```
1   cust_fname    |    cust_lname     |      town
2 ----------------+-------------------+----------------
3  Jobey           | Boeter            | La Mohammedia
4  York            | O'Deegan          | Chemnitz
5  Penelope        | Hexter            | Pingshan
6  Chadd           | Franz-Schoninger  | Baojia
7  Vikky           | Eke               | Kamenný řPívoz
8  Marie-françoise | Currier           | Waekolong
9  Bénédicte       | Dozdill           | Dawuhan
10 Görel           | Douthwaite        | Sunbu
11 Bérengère       | Menendez          | Tsagaanders
12 Pélagie         | Hachard           | Jiantou
13 Adaobi          | Musa              | La Mohammedia
14 (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

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

LANGUAGE: Unknown

```
1  cust_fname | cust_lname
2 ------------+------------
3  Jobey      | Boeter
4  Adaobi     | Musa
5 (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

```
1 dsd_22=# \d customer
2                                 Table "public.customer"
3    Column   |          Type          | Collation | Nullable |                Default
4 ------------+------------------------+-----------+----------+------------------------------------
     ↪
5  cust_id    | integer                |           | not null | nextval('customer_cust_id_seq'::
     ↪ regclass)
6  cust_fname | character varying(25)  |           | not null |
7  cust_lname | character varying(35)  |           | not null |
8  addr1      | character varying(50)  |           | not null |
9  addr2      | character varying(50)  |           |          |
10 town       | character varying(60)  |           | not null |
11 postcode   | character(9)           |           | not null |
12 email      | character varying(255) |           | not null |
13 Indexes:
14     "customer_pkey" PRIMARY KEY, btree (cust_id)
15 Referenced by:
16     TABLE "cust_order" CONSTRAINT "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES
     ↪ customer(cust_id)
```

```
17
18  dsd_22=# \d cust_order
19                                  Table "public.cust_order"
20     Column   |  Type   | Collation | Nullable |                    Default
21  -------------+---------+-----------+----------+-----------------------------------------------
        ↪
22   cust_ord_id | integer |           | not null | nextval('cust_order_cust_ord_id_seq'::regclass)
23   staff_id    | integer |           |          |
24   cust_id     | integer |           |          |
25  Indexes:
26      "cust_order_pkey" PRIMARY KEY, btree (cust_ord_id)
27  Foreign-key constraints:
28      "cust_order_cust_id_fkey" FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
29      "cust_order_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
30  Referenced by:
31      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

```sql
1  SELECT * FROM category;
```

LANGUAGE: Unknown

```
1   cat_id |   cat_name
2  --------+-------------
3        1 | Men's Wear
4        2 | Ladies Wear
5        3 | Kid's Wear
6        4 | Outdoor
7        5 | Sport
8        6 | Health
9  (6 rows)
```

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

LANGUAGE: SQL

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

LANGUAGE: Unknown

```
1   cat_id
2  --------
3        5
4  (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

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

LANGUAGE: Unknown

```
1            prod_name                    |  cat_name
2 ----------------------------------------+-------------
3  Multi-layered multi-tasking initiative |  Ladies Wear
4  Operative analyzing task-force         |  Men's Wear
5  Exclusive client-server array          |  Sport
6  Balanced client-server product         |  Health
7  Exclusive background website           |  Sport
8  Pre-emptive holistic intranet          |  Health
9  Re-engineered cohesive methodology     |  Men's Wear
10 Robust directional projection          |  Ladies Wear
11 Inverse transitional infrastructure    |  Outdoor
12 Multi-tiered explicit paradigm         |  Health
13 ...
14 (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

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

LANGUAGE: Unknown

```
1  staff_fname | staff_lname |          work_email          |    role_name
2 -------------+-------------+------------------------------+-----------------
3  Montgomery  | Housegoe    | Montgomery.Housegoe@dsd.com  | Order Picker
4  Niel        | Welsby      | Niel.Welsby@dsd.com          | Final Packer
5  Jillene     | Revitt      | Jillene.Revitt@dsd.com       | Post Sales
6  Harriette   | Fewster     | Harriette.Fewster@dsd.com    | Post Sales
7  Aura        | Clewlowe    | Aura.Clewlowe@dsd.com        | Post Sales
8  Hanan       | Gloster     | Hanan.Gloster@dsd.com        | Customer Retain
9  Nikoletta   | Shrimpton   | Nikoletta.Shrimpton@dsd.com  | Customer Retain
10 Tim         | Illem       | Tim.Illem@dsd.com            | Misc
11 Nell        | Olsson      | Nell.Olsson@dsd.com          | Misc
12 Janeva      | Gillicuddy  | Janeva.Gillicuddy@dsd.com    | Misc
13 (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

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

LANGUAGE: Unknown

```
1  staff_lname |    role_name     | cust_ord_id | cust_fname | cust_lname
2 -------------+------------------+-------------+------------+------------
```

```
3   Welsby       | Final Packer    |         82 | Vikky       | Eke
4   Clewlowe     | Post Sales      |         90 | Vikky       | Eke
5   Welsby       | Final Packer    |        105 | Vikky       | Eke
6   Housegoe     | Order Picker    |        115 | Vikky       | Eke
7   Gillicuddy   | Misc            |        118 | Vikky       | Eke
8   Welsby       | Final Packer    |        130 | Vikky       | Eke
9   Shrimpton    | Customer Retain |        132 | Vikky       | Eke
10  Welsby       | Final Packer    |        135 | Vikky       | Eke
11  Housegoe     | Order Picker    |        145 | Vikky       | Eke
12  (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
1 SELECT customer.cust_lname, category.cat_name FROM customer
2 JOIN cust_order ON customer.cust_id = cust_order.cust_id
3 JOIN manifest ON cust_order.cust_ord_id = manifest.cust_ord_id
4 JOIN product ON product.prod_id = manifest.prod_id
5 JOIN category ON category.cat_id = product.prod_cat
6 WHERE customer.town = 'Sunbu';
```

LANGUAGE: Unknown

```
1    cust_lname |   cat_name
2   ------------+-------------
3    Douthwaite | Outdoor
4    Douthwaite | Sport
5    Douthwaite | Kid's Wear
6    Douthwaite | Outdoor
7    Douthwaite | Sport
8    Douthwaite | Sport
9    Douthwaite | Ladies Wear
10   Douthwaite | Outdoor
11   Douthwaite | Sport
12   Douthwaite | Ladies Wear
13  (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
1 SELECT customer.cust_lname, count(category.cat_name), category.cat_name FROM customer
2 JOIN cust_order ON customer.cust_id = cust_order.cust_id
3 JOIN manifest ON cust_order.cust_ord_id = manifest.cust_ord_id
4 JOIN product ON product.prod_id = manifest.prod_id
5 JOIN category ON category.cat_id = product.prod_cat
6 GROUP BY customer.cust_lname, category.cat_name, customer.town
7 HAVING customer.town='Sunbu';
```

LANGUAGE: Unknown

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

Page 14

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
LANGUAGE: SQL
1  SELECT CUSTOMER.CUST_ID, CUST_ORDER.CUST_ORD_ID FROM CUSTOMER
2  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
LANGUAGE: SQL
1  SELECT CUSTOMER.CUST_ID, CUST_ORDER.CUST_ORD_ID FROM CUSTOMER
2  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
LANGUAGE: SQL
1  SELECT CUSTOMER.CUST_ID, CUST_ORDER.CUST_ORD_ID FROM CUSTOMER
2  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
LANGUAGE: SQL
1  SELECT role_name, staff_lname, staff_fname FROM staff FULL OUTER JOIN
2  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

Page 15

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.

```sql
LANGUAGE: SQL
1  INSERT INTO ROLE (role_name)
2  VALUES ('Cleaner');
3
4  INSERT INTO ROLE (role_name)
5  VALUES ('Pre Sales');
```

Then run the following.

```sql
LANGUAGE: SQL
1  SELECT count(*)
2  FROM ROLE;
```

This generates the following output

```
LANGUAGE: Unknown
1   count
2  -------
3       7
4  (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.

```sql
LANGUAGE: SQL
1  SELECT staff.staff_fname, staff.staff_lname, staff.home_email, role.role_name FROM staff
2  JOIN role on staff.role = role.role_id;
```

```
LANGUAGE: Unknown
1   staff_fname | staff_lname |          home_email          |   role_name
2  -------------+-------------+------------------------------+----------------
3   Montgomery  | Housegoe    | mhousegoe2@ucoz.ru           | Order Picker
4   Niel        | Welsby      | nwelsby0@rambler.ru          | Final Packer
5   Jillene     | Revitt      | jrevitt8@cornell.edu         | Post Sales
6   Harriette   | Fewster     | hfewster7@independent.co.uk  | Post Sales
7   Aura        | Clewlowe    | aclewlowe5@google.com.au     | Post Sales
8   Hanan       | Gloster     | hgloster3@blogger.com        | Customer Retain
```

```
9   Nikoletta   | Shrimpton   | nshrimpton1@unblog.fr      | Customer Retain
10  Tim         | Illem       | tillem9@dedecms.com        | Misc
11  Nell        | Olsson      | nolsson6@jiathis.com       | Misc
12  Janeva      | Gillicuddy  | jgillicuddy4@altervista.org | Misc
13  (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
1  SELECT staff.staff_fname, staff.staff_lname, staff.home_email, role.role_name FROM staff, role
2  WHERE staff.role = role.role_id;
```

LANGUAGE: Unknown

```
1   staff_fname | staff_lname |       home_email           |   role_name
2  -------------+-------------+----------------------------+-----------------
3   Montgomery  | Housegoe    | mhousegoe2@ucoz.ru         | Order Picker
4   Niel        | Welsby      | nwelsby0@rambler.ru        | Final Packer
5   Jillene     | Revitt      | jrevitt8@cornell.edu       | Post Sales
6   Harriette   | Fewster     | hfewster7@independent.co.uk | Post Sales
7   Aura        | Clewlowe    | aclewlowe5@google.com.au   | Post Sales
8   Hanan       | Gloster     | hgloster3@blogger.com      | Customer Retain
9   Nikoletta   | Shrimpton   | nshrimpton1@unblog.fr      | Customer Retain
10  Tim         | Illem       | tillem9@dedecms.com        | Misc
11  Nell        | Olsson      | nolsson6@jiathis.com       | Misc
12  Janeva      | Gillicuddy  | jgillicuddy4@altervista.org | Misc
13  (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 Waeko-long. Copy the code and the answer below.

LANGUAGE: SQL

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

LANGUAGE: Unknown

```
1    cust_fname   | cust_lname |       email            |                  prod_name
2  ---------------+------------+------------------------+-------------------------------------------------
       ↪
3   Marie-françoise | Currier   | acurrier0@economist.com | Vision-oriented attitude-oriented
       ↪ core
4   Marie-françoise | Currier   | acurrier0@economist.com | Balanced client-server product
5   Marie-françoise | Currier   | acurrier0@economist.com | Exclusive client-server array
6   Marie-françoise | Currier   | acurrier0@economist.com | Universal encompassing conglomeration
7   Marie-françoise | Currier   | acurrier0@economist.com | Synergistic homogeneous ability
8   Marie-françoise | Currier   | acurrier0@economist.com | Universal exuding protocol
9   Marie-françoise | Currier   | acurrier0@economist.com | Universal global hub
10  Marie-françoise | Currier   | acurrier0@economist.com | Balanced real-time info-mediaries
11  Marie-françoise | Currier   | acurrier0@economist.com | Integrated 24/7 interface
12  Marie-françoise | Currier   | acurrier0@economist.com | Re-engineered explicit software
13  Marie-françoise | Currier   | acurrier0@economist.com | Customizable cohesive capacity
14  Marie-françoise | Currier   | acurrier0@economist.com | Robust mission-critical complexity
15  Marie-françoise | Currier   | acurrier0@economist.com | Organic clear-thinking system engine
16  Marie-françoise | Currier   | acurrier0@economist.com | Stand-alone composite Graphical User
       ↪ Interface
17  (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

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

LANGUAGE: Unknown

```
1   cat_name   |                    prod_name
2  ------------+-------------------------------------------------
3   Health     | Exclusive multimedia middleware
4   Health     | Pre-emptive holistic intranet
5   Health     | Ameliorated next generation orchestration
6   Health     | Monitored asynchronous function
7   Health     | Right-sized mission-critical pricing structure
8   Health     | Profound human-resource forecast
9   Health     | Realigned client-driven database
10  Health     | Seamless optimal leverage
11  Health     | User-friendly encompassing array
12  Health     | Customizable cohesive capacity
13  ...
14 (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

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

LANGUAGE: Unknown

```
1   cat_name   |                    prod_name
2  ------------+-------------------------------------------------
3   Health     | Balanced client-server product
4   Health     | Pre-emptive holistic intranet
5   Health     | Multi-tiered explicit paradigm
6   Health     | Monitored asynchronous function
7   Health     | Right-sized mission-critical pricing structure
8   Health     | Open-architected homogeneous concept
9   Health     | Fully-configurable full-range interface
10  Health     | Customizable cohesive capacity
11  Health     | Seamless optimal leverage
12  Health     | Realigned client-driven database
13  ...
14 (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

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

```
LANGUAGE: Unknown
1   cat_name   |                      prod_name                     | prod_id
2 -------------+----------------------------------------------------+---------
3  Health      | Balanced client-server product                     |      4
4  Health      | Pre-emptive holistic intranet                      |      6
5  Health      | Multi-tiered explicit paradigm                     |     10
6  Health      | Monitored asynchronous function                    |     20
7  Health      | Right-sized mission-critical pricing structure     |     23
8  Health      | Open-architected homogeneous concept               |     37
9  Health      | Fully-configurable full-range interface            |     46
10 Health      | Customizable cohesive capacity                     |     54
11 Health      | Seamless optimal leverage                          |     57
12 Health      | Realigned client-driven database                   |     59
13 ...
14 (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.

```sql
LANGUAGE: SQL
1 SELECT staff.staff_fname, staff.staff_lname, staff.work_email FROM staff
2 JOIN role ON staff.role=role.role_id
3 WHERE role.role_name='Cleaner';
```

```
LANGUAGE: Unknown
1  staff_fname | staff_lname | work_email
2 -------------+-------------+------------
3 (0 rows)
```

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

```sql
LANGUAGE: SQL
1 SELECT count(*) FROM staff
2 JOIN role ON staff.role = role.role_id
3 WHERE role.role_name='Misc';
```

```
LANGUAGE: Unknown
1  count
2 -------
3      3
4 (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.

```sql
LANGUAGE: SQL
1 SELECT staff.staff_fname, staff.staff_lname, concat_ws(' ', addr1, addr2, town, postcode) AS "
    ↪ address"
2 FROM staff
3 JOIN role ON role.role_id = staff.role
4 WHERE role.role_name='Misc';
```

```
LANGUAGE: Unknown
1  staff_fname | staff_lname |                   address
```

```
2 ------------+------------+------------------------------------------
3  Janeva     | Gillicuddy | 6999 Kings Park Sachtjen Portsmouth PO05 5SF
4  Nell       | Olsson     | 18424 Kenwood Court Farmco Havant PO22 6DL
5  Tim        | Illem      | 85 Lillian Way Farragut Southsea PO93 OCN
6 (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

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

LANGUAGE: Unknown

```
1  prod_id |                 prod_name
2 ---------+------------------------------------------------
3        7 | Re-engineered cohesive methodology
4       11 | Re-engineered explicit software
5       18 | Re-engineered actuating capability
6       26 | Realigned 5th generation artificial intelligence
7       39 | Realigned homogeneous hub
8       56 | Reduced fresh-thinking process improvement
9       59 | Realigned client-driven database
10       76 | Re-engineered 24/7 knowledge base
11 (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

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

LANGUAGE: Unknown

```
1  prod_id |           prod_name
2 ---------+------------------------------
3       80 | Profound value-added intranet
4 (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

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

LANGUAGE: Unknown

```
1  prod_id | prod_name
2 ---------+-----------
3 (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 cate-

gory 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.

```sql
LANGUAGE: SQL
1 SELECT customer.cust_fname, customer.cust_lname, customer.email, cust_order.cust_ord_id,
    ↪ category.cat_name, product.prod_name from customer
2 JOIN cust_order ON customer.cust_id=cust_order.cust_id
3 JOIN manifest ON cust_order.cust_ord_id=manifest.cust_ord_id
4 JOIN product on manifest.prod_id=product.prod_id
5 JOIN category on category.cat_id=product.prod_cat
6 WHERE product.prod_name LIKE '%able%'
7 ORDER BY category.cat_name, product.prod_name;
```

```
LANGUAGE: Unknown
1    cust_fname    |    cust_lname    |                email                 | cust_ord_id |  cat_name
    ↪     |
2  prod_name
3 ----------------+------------------+--------------------------------------+-------------+-----------+------------
    ↪
4 Bérengère       | Menendez         | amenendez3@dell.com                  |          64 | Health
    ↪     | Customizable cohesive capacity
5 Marie-françoise | Currier          | acurrier0@economist.com              |         133 | Health
    ↪     | Customizable cohesive capacity
6 Bérengère       | Menendez         | amenendez3@dell.com                  |         102 | Health
    ↪     | Fully-configurable full-range interface
7 Chadd           | Franz-Schoninger | cfranzschoninger3@google.com.hk |      7 | Health
    ↪     | Team-oriented stable project
8 Chadd           | Franz-Schoninger | cfranzschoninger3@google.com.hk |     81 | Health
    ↪     | Team-oriented stable project
9 Bénédicte       | Dozdill          | cdozdill1@amazon.de                  |          24 | Kid's
    ↪ Wear | Configurable analyzing solution
10 Bérengère      | Menendez         | amenendez3@dell.com                  |          21 | Kid's
    ↪ Wear | Configurable analyzing solution
11 Bérengère      | Menendez         | amenendez3@dell.com                  |         113 | Kid's
    ↪ Wear | Configurable analyzing solution
12 Jobey          | Boeter           | jboeter0@mail.ru                     |          91 | Kid's
    ↪ Wear | Configurable analyzing solution
13 Jobey          | Boeter           | jboeter0@mail.ru                     |          39 | Outdoor
    ↪     | Switchable tangible product
14 Jobey          | Boeter           | jboeter0@mail.ru                     |          26 | Outdoor
    ↪     | Switchable tangible product
15 Vikky          | Eke              | veke4@elegantthemes.com              |         105 | Sport
    ↪     | Configurable methodical firmware
16 Vikky          | Eke              | veke4@elegantthemes.com              |         118 | Sport
    ↪     | Customizable well-modulated encryption
17 Pélagie        | Hachard          | fhachard4@blinklist.com              |          89 | Sport
    ↪     | Virtual stable Graphic Interface
18 (14 rows)
```

Page 16

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
1 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
1 CREATE [viewName] AS [queryString];
2
3 --eg
4 CREATE VIEW CUST_NAMES AS SELECT CUST_FNAME, CUST_LNAME FROM customer;
```

The view above can be executed as

```
LANGUAGE: SQL
1 SELECT * FROM CUST_NAMES;
```

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

Page 17

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

```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

```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

```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

```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

```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
```

```
11   Penelope          | Hexter             |                 9
12   York              | O'Deegan           |                10
13   ...
14  (251 rows)
```

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

LANGUAGE: SQL

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

LANGUAGE: Unknown

```
1     cust_fname      |     cust_lname     | cust_ord_id
2  -----------------+-------------------+-------------
3   Chadd             | Franz-Schoninger  |           1
4   York              | O'Deegan          |           2
5   Marie-françoise   | Currier           |           3
6   Bérengère         | Menendez          |           4
7   Bénédicte         | Dozdill           |           5
8   Bénédicte         | Dozdill           |           6
9   Chadd             | Franz-Schoninger  |           7
10  Bénédicte         | Dozdill           |           8
11  Penelope          | Hexter            |           9
12  York              | O'Deegan          |          10
13  ...
14 (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

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

LANGUAGE: Unknown

```
1     cust_fname      |     cust_lname     | cust_ord_id
2  -----------------+-------------------+-------------
3   Chadd             | Franz-Schoninger  |           1
4   York              | O'Deegan          |           2
5   Marie-françoise   | Currier           |           3
6   Bérengère         | Menendez          |           4
7   Bénédicte         | Dozdill           |           5
8   Bénédicte         | Dozdill           |           6
9   Chadd             | Franz-Schoninger  |           7
10  Bénédicte         | Dozdill           |           8
11  Penelope          | Hexter            |           9
12  York              | O'Deegan          |          10
13  ...
14 (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

```sql
1 -- use query from question 6
```

```
2 SELECT customer.cust_fname, customer.cust_lname, cust_order.cust_ord_id FROM cust_order
3 LEFT JOIN customer ON customer.cust_id=cust_order.cust_id
4 ORDER BY customer.cust_id, cust_order.cust_ord_id;
```

LANGUAGE: Unknown

```
1    cust_fname    |    cust_lname    |  cust_ord_id
2 -----------------+-----------------+-------------
3  Jobey           | Boeter          |           26
4  Jobey           | Boeter          |           34
5  Jobey           | Boeter          |           39
6  Jobey           | Boeter          |           57
7  Jobey           | Boeter          |           68
8  Jobey           | Boeter          |           71
9  Jobey           | Boeter          |           77
10 Jobey           | Boeter          |           91
11 Jobey           | Boeter          |           98
12 Jobey           | Boeter          |           99
13 Jobey           | Boeter          |          131
14 Jobey           | Boeter          |          143
15 Jobey           | Boeter          |          146
16 York            | O'Deegan        |            2
17 York            | O'Deegan        |           10
18 York            | O'Deegan        |           19
19 ...
20 (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

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

LANGUAGE: Unknown

```
1  cust_id | cust_ord_id | staff_id | staff_lname |        work_email
2 ---------+-------------+----------+-------------+---------------------------
3        4 |           1 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
4        2 |           2 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
5        6 |           3 |        2 | Shrimpton   | Nikoletta.Shrimpton@dsd.com
6        9 |           4 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
7        7 |           5 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
8        7 |           6 |        4 | Gloster     | Hanan.Gloster@dsd.com
9        4 |           7 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
10       7 |           8 |        3 | Housegoe    | Montgomery.Housegoe@dsd.com
11       3 |           9 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
12       2 |          10 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
13       7 |          11 |        6 | Clewlowe    | Aura.Clewlowe@dsd.com
14       9 |          12 |        4 | Gloster     | Hanan.Gloster@dsd.com
15       7 |          13 |        4 | Gloster     | Hanan.Gloster@dsd.com
16       7 |          14 |        4 | Gloster     | Hanan.Gloster@dsd.com
17       6 |          15 |        4 | Gloster     | Hanan.Gloster@dsd.com
18       9 |          16 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
19      10 |          17 |        5 | Gillicuddy  | Janeva.Gillicuddy@dsd.com
20       7 |          18 |        3 | Housegoe    | Montgomery.Housegoe@dsd.com
21       2 |          19 |        3 | Housegoe    | Montgomery.Housegoe@dsd.com
22 ...
23 (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

```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

```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
14  Nikoletta    | Shrimpton    | Nikoletta.Shrimpton@dsd.com |        71 |        1 | Jobey
         ↪         | Boeter           | Inverse high-level attitude
15  Montgomery   | Housegoe     | Montgomery.Housegoe@dsd.com |        98 |        1 | Jobey
         ↪         | Boeter           | Distributed uniform Graphic Interface
16  Niel         | Welsby       | Niel.Welsby@dsd.com          |       112 |        6 | Marie-
    ↪ françoise | Currier      | Integrated 24/7 interface
17  ...
18  (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
1  SELECT c.cust_fname, c.email FROM customer c
2  FULL OUTER JOIN cust_order co ON c.cust_id = co.cust_id
3  WHERE co.cust_ord_id IS NULL;
```

LANGUAGE: Unknown

```
1   cust_fname |              email
2  ------------+-----------------------------------
3   Jen        | jsettle222@google.ca
4   Fawnia     | fpetchell1@networkadvertising.org
5   Nealy      | nstanley7@arstechnica.com
6   Tine       | tclopton5@typepad.com
7   Cody       | clago8@rambler.ru
8   Lonnie     | lmacgilpatrick6@uiuc.edu
9   Evie       | 3vi3@google.wh
10  Mireielle  | mkillner2@cafepress.com
11  Falkner    | fgrouer4@dion.ne.jp
12  Kaine      | klawford3@imdb.com
13  Theadora   | tajsik9@sfgate.com
14  (11 rows)
```

Page 18

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.

Page 19

LECTURE: Database Security - Privileges

📅 26-01-23   🕐 13:00   🎓 Mark   📍 RB LT1

NB: This lecture was not delivered as scheduled due to staff sickness. Notes have been taken from the slides made available on Moodle.

Privileges

When we say 'privileges' we are referring to what someone can do. We should never allow someone to do everything in a database, except the database admin.
It is the role of the database administrator to work out what access levels users will need to the database. Deciding which privileges someone needs is complicated and often factors such as their job role or position in the company come into play. For example, what data does someone in the sales team need access to; or what access should a boss have, read only to everything? There may be multiple people within one department who have different levels of access. Ultimately, there isn't a nice 'one size fits all' rule which can be applied to giving the right level of access. Levels of access have to be considered on a case-by-case basis.

Setting Access Levels

Access can be granted on different levels and different activities. Users can be given access to entire databases, some tables or only some views. They can be given permissions to select data, insert data, update data or delete data.
Users can also be given access to create views however this is not always a good idea.

Encryption

PostgreSQL has several different types of data security, this includes: PGP (Pretty Good Privacy); and Hashing (using md5, sha1, sha225, sha256, sha348 and sha512). By default encryption is disabled, to enable it the following line of code needs to be run.

```sql
LANGUAGE: SQL
1  CREATE EXTENSION pgcrypto;
```

There are many benefits to using encryption, these include: the data is not available in clear text; and without the key the data cannot be read. However there are a number of downsides: encryption & decryption is slow. Often it is worth taking the time to do this however there will be some data in the database which does not need to be encrypted, for example product names.

Salt

Salting adds some text to the value you need to encrypt. When salting is not used and the same encryption algorithm is used, all input data will be the same when encrypted, this can lead to security issues. However, if salting is used and a salt value is added before encryption, even if two input values are the same once encrypted (permitting they have different salt values) the outputs will be completely different.
PostgreSQL has an inbuilt salt value generation function (`gen_salt()`) which produces a random salt value. The hashing algorithm used is stored in the encrypted string produced by the algorithm so that the data can be decrypted; otherwise you wouldn't be able to decrypt data as the salt generation function is random.

SQL Injection

> **SQL Injection**
>
> A web security vulnerability that allows an attacker to interfere with the queries than an application makes to its database.

This needs to be stopped both at the application and database level. This is done by sanitising user inputs at the application level (can be done in any programming language) and by using views at the database level.
There are a number of methods which can be used to prevent SQL injection: using stored procedures, enforcing least privileges, and having multiple database users.

Page 20

PRACTICAL: Security One

📅 26-01-23   🕐 14:00   🎓 Val & Co   📍 FTC 3

T1. Create a new role. Call this new role your first name. It must be given a password and the ability to login. Copy your code and response below:

```sql
LANGUAGE: SQL
CREATE ROLE thomas WITH LOGIN PASSWORD 'highlySecure1!';
```

T2. Try to use this new role by using the following code

```
LANGUAGE: Pseudocode
psql -h localhost -p 5432 -U thomas
```

Output:

```
LANGUAGE: Pseudocode
Password for user thomas:
psql: FATAL:  database "thomas" does not exist
```

T3. As your normal user, create a new database that has the same name as your new role. This needs to be owned by the new user.

```sql
LANGUAGE: SQL
CREATE DATABASE thomas OWNER thomas;
```

Outputs:

```
LANGUAGE: Pseudocode
CREATE DATABASE
```

T4. Try to use this new role by using the following code

```
LANGUAGE: Pseudocode
psql -h localhost -p 5432 -U thomas
```

Outputs

```
LANGUAGE: Pseudocode
Password for user thomas:
```

T5. What does the prompt look like when you log in with your new role? Copy it below.

```
LANGUAGE: Pseudocode
1 thomas=>
```

T6. List the databases available. Copy the output below.

```
LANGUAGE: Pseudocode
1 thomas=> \l
2                                     List of databases
3      Name       |     Owner       | Encoding | Collate | Ctype  |  Access privileges
4 ----------------+-----------------+----------+---------+--------+----------------------
5  code_test      | up2108121       | UTF8     | C.UTF-8 | C.UTF-8 |
6  customer_db    | up2108121       | UTF8     | C.UTF-8 | C.UTF-8 |
7  dsd_22         | up2108121       | UTF8     | C.UTF-8 | C.UTF-8 |
8  mongo-2021-fix | mongo-2021-fix  | UTF8     | C.UTF-8 | C.UTF-8 |
9  postgres       | postgres       | UTF8     | C.UTF-8 | C.UTF-8 |
10 template0      | postgres       | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
11                |                 |          |         |         | postgres=CTc/postgres
12 template1      | postgres       | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
13                |                 |          |         |         | postgres=CTc/postgres
14 thomas         | thomas         | UTF8     | C.UTF-8 | C.UTF-8 |
15 up2108121      | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
16 up2108121_cw   | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
17 week02         | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
18 (11 rows)
```

T7. Connect to a different database and list the tables. Copy the output below.

```
LANGUAGE: Pseudocode
1 thomas=> \c dsd_22
2 SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
3 You are now connected to database "dsd_22" as user "thomas".
4 dsd_22=>
```

T8. Select all of the data in one of the tables listed in T7. Copy the output below.

```
LANGUAGE: SQL
1 SELECT * FROM manifest;
```

```
LANGUAGE: Pseudocode
1 ERROR:  permission denied for table manifest
```

T9. As your normal user, make the new role a superuser with the following code:

```
LANGUAGE: SQL
1 ALTER ROLE thomas WITH SUPERUSER;
```

T10. Make sure your new role is logged out with \q and then log in again. What does the prompt now look like? Copy this prompt below

```
LANGUAGE: Pseudocode
1 up2108121@up2108121:~ psql -h localhost -p 5432 -U thomas
2 Password for user thomas:
3 psql (11.18 (Debian 11.18-0+deb10u1))
4 SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
5 Type "help" for help.
6
7 thomas=#
```

T11. List the databases available. Copy the output below.

```
LANGUAGE: Pseudocode

thomas=# \l
                                   List of databases
      Name        |     Owner      | Encoding | Collate | Ctype  |   Access privileges
----------------+----------------+----------+---------+---------+-----------------------
 code_test       | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
 customer_db      | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
 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
 thomas          | thomas         | UTF8     | C.UTF-8 | C.UTF-8 |
 up2108121       | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
 up2108121_cw    | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
 week02          | up2108121      | UTF8     | C.UTF-8 | C.UTF-8 |
(11 rows)
```

T12. Connect to a different database and list the tables. Copy the output below.

```
LANGUAGE: Pseudocode

\c up2108121_cw
\dt

List of relations
Schema |        Name        | Type  |   Owner
--------+------------------+-------+-----------
public | boat             | table | up2108121
public | boat_spec        | table | up2108121
public | boatyard         | table | up2108121
public | customer         | table | up2108121
public | role             | table | up2108121
public | service          | table | up2108121
public | service_contents | table | up2108121
public | service_item     | table | up2108121
public | service_staff    | table | up2108121
public | staff            | table | up2108121
public | staff_role       | table | up2108121
(11 rows)
```

T12. Select all of the data in one of the tables listed in T7. Copy the output below.

```
LANGUAGE: SQL

\c dsd_22
SELECT * FROM manifest;
```

```
LANGUAGE: Pseudocode

manifest_id | cust_ord_id | prod_id
-------------+-------------+---------
          1 |           1 |      84
          2 |           2 |       1
          3 |           3 |      91
          4 |           4 |       5
          5 |           5 |      97
          6 |           6 |      74
          7 |           7 |      88
          8 |           8 |      97
          9 |           9 |      66
         10 |          10 |      43
         11 |          11 |      78
         12 |          12 |      24
```

```
15          13 |         13 |         69
16          14 |         14 |         25
17          15 |         15 |          4
18          16 |         16 |         32
19          17 |         17 |         66
20          18 |         18 |         13
21          19 |         19 |         83
22          20 |         20 |          4
23          21 |         21 |         45
24          22 |         22 |          4
25          23 |         23 |         93
26          24 |         24 |         45
27 ...
```

Page 21

PRACTICAL: Security Two

📅 2023-02-02   🕐 14:00   🎓 Mark & Co   📍 FTC 3

T1. Create 2 new roles and give them both login ability and passwords. You can choose the role names. (This was done in last week's practical. If you can't log in, look at the error messages and fix it.)

```sql
LANGUAGE: SQL
1 CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
2 CREATE DATABASE user1 OWNER user1;
3
4 CREATE ROLE user2 WITH LOGIN PASSWORD 'password2';
5 CREATE DATABASE user2 OWNER user2;
```

T2. Login with one of the new roles Get a list of all the databases with \l. Can you see other databases?

```
LANGUAGE: Pseudocode
1 up2108121@up2108121:~\$ psql -h localhost -p 5432 -U user1
2 Password for user user1:
3 psql (11.18 (Debian 11.18-0+deb10u1))
4 SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
5 Type "help" for help.
6
7 user1=> \l
8                                  List of databases
9       Name      |     Owner     | Encoding | Collate |  Ctype  |   Access privileges
10 ---------------+---------------+----------+---------+---------+----------------------
11  code_test     | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
12  customer_db   | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
13  dsd_22        | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
14  mongo-2021-fix | mongo-2021-fix | UTF8    | C.UTF-8 | C.UTF-8 |
15  postgres      | postgres      | UTF8     | C.UTF-8 | C.UTF-8 |
16  template0     | postgres      | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
17                |               |          |         |         | postgres=CTc/postgres
18  template1     | postgres      | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
19                |               |          |         |         | postgres=CTc/postgres
20  thomas        | thomas        | UTF8     | C.UTF-8 | C.UTF-8 |
21  up2108121     | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
22  up2108121_cw  | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
23  user1         | user1         | UTF8     | C.UTF-8 | C.UTF-8 |
24  user2         | user2         | UTF8     | C.UTF-8 | C.UTF-8 |
25  week02        | up2108121     | UTF8     | C.UTF-8 | C.UTF-8 |
26 (13 rows)
```

T3. Connect to `dsd_22` and list the tables with `\dt`. Can you see all the tables in the `dsd_22` database?

```
LANGUAGE: Pseudocode
1 user1=> \c dsd_22
2 SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
3 You are now connected to database "dsd_22" as user "user1".
4 dsd_22=> \dt
5           List of relations
6  Schema |     Name     | Type  |   Owner
```

```
7  --------+------------+-------+-----------
8  public | category   | table | up2108121
9  public | cust_order | table | up2108121
10 public | customer   | table | up2108121
11 public | manifest   | table | up2108121
12 public | product    | table | up2108121
13 public | role       | table | up2108121
14 public | staff      | table | up2108121
15 (7 rows)
```

T4. Run a `SELECT` statement on the product table. Use the following command:

LANGUAGE: SQL

```sql
SELECT * FROM PRODUCT WHERE PROD_ID <= 10;
```

LANGUAGE: Pseudocode

```
ERROR:  permission denied for table product
```

T5. As your normal user, the `upxxxxxx` user, grant the new role the ability to run `SELECT` commands on the product table.

LANGUAGE: SQL

```sql
GRANT select
ON product
TO user1;
```

T6. As the new role, can you now run the command you ran in step 4? Copy the response below.

LANGUAGE: Pseudocode

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

T7. Run the following code to `INSERT` a new product:

LANGUAGE: SQL

```sql
INSERT INTO PRODUCT (PROD_NAME,PROD_CAT) VALUES ('The Amazing New Thingy',3);
```

LANGUAGE: Pseudocode

```
ERROR:  permission denied for table product
```

T8. Run the following code

```
LANGUAGE: SQL
1 SELECT PROD_NAME, PROD_ID, PROD_CAT FROM PRODUCT WHERE PROD_NAME = 'The Amazing New Thingy';
```

```
LANGUAGE: Pseudocode
1  prod_name | prod_id | prod_cat
2 -----------+---------+----------
3 (0 rows)
```

T9. Give both the new roles the `UPDATE` privilege on the role table.

```
LANGUAGE: SQL
1 GRANT update
2 ON role
3 TO user1, user2;
```

T10. List the `role_names` that are stored in the role table. Copy below:

```
LANGUAGE: SQL
1 SELECT role_name FROM role;
```

```
LANGUAGE: Pseudocode
1 ERROR:  permission denied for table role
```

T11. Run the following command as the second new role. (Not the one you did the initial tests on)

```
LANGUAGE: SQL
1 UPDATE ROLE SET ROLE_NAME = 'Hygiene Expert' where role_name = 'Cleaner';
```

```
LANGUAGE: Pseudocode
1 ERROR:  permission denied for table role
```

To give permission to be able to `UPDATE`, the user must also have permission to `SELECT`. This is the same as for `DELETE`.

```
LANGUAGE: SQL
1 -- sql to update permission of user2 to be able to select
2 GRANT select ON role TO user2;
```

Now run the SQL provided again.

```
LANGUAGE: Pseudocode
1 UPDATE 1
```

T12. List the `role_names` that are stored in the role table. Do you have a new role? Is this the same `role_id` value? Copy below:

LANGUAGE: SQL

```sql
SELECT role_name, role_id from role;
```

LANGUAGE: Pseudocode

```
   role_name     | role_id
-----------------+---------
 Order Picker    |       1
 Final Packer    |       2
 Post Sales      |       3
 Customer Retain |       4
 Misc            |       5
 Pre Sales       |       7
 Hygiene Expert  |       6
(7 rows)
```

T13. As your normal user, (the superuser), create a view that selects the customer first and last names and their email addresses. Call the view `cust_email`. Copy your code, once you have run it successfully, below. (Views were covered in lecture 9). Copy your code and the response below.

LANGUAGE: SQL

```sql
CREATE VIEW cust_email AS SELECT cust_fname, cust_lname, email FROM customer;
```

LANGUAGE: Pseudocode

```
CREATE VIEW
```

LANGUAGE: SQL

```sql
SELECT * FROM cust_email;
```

LANGUAGE: Pseudocode

```
 cust_fname       |    cust_lname     |               email
------------------+-------------------+-----------------------------------
 Jobey            | Boeter            | jboeter0@mail.ru
 York             | O'Deegan          | yodeegan1@nydailynews.com
 Penelope         | Hexter            | phexter2@cbslocal.com
 Chadd            | Franz-Schoninger  | cfranzschoninger3@google.com.hk
 Vikky            | Eke               | veke4@elegantthemes.com
 Marie-françoise  | Currier           | acurrier0@economist.com
 Bénédicte        | Dozdill           | cdozdill1@amazon.de
 Görel            | Douthwaite        | edouthwaite2@feedburner.com
 Bérengère        | Menendez          | amenendez3@dell.com
 ...
(35 rows)
```

T14. As the first new role, run a SELECT on this new role. Copy the response below.

LANGUAGE: SQL

```sql
SELECT * FROM cust_email;
```

LANGUAGE: Pseudocode

```
1  ERROR:   permission denied for view cust_email
```

T15. `GRANT` the ability for the 2nd new role to run the view. Remember that you run a `SELECT *` on the view to get the data displayed.

LANGUAGE: SQL
```
1  GRANT select ON cust_email TO user2;
```

LANGUAGE: Pseudocode
```
1  GRANT
```

T16. Run the `SELECT *` on the view for both of your new roles. Copy the outputs below.
`user1`

LANGUAGE: SQL
```
1  SELECT * FROM cust_email;
```

LANGUAGE: Pseudocode
```
1  ERROR:   permission denied for view cust_email
```

`user2`

LANGUAGE: SQL
```
1  SELECT * FROM cust_email;
```

LANGUAGE: Pseudocode
```
1   cust_fname     |    cust_lname     |                email
2  ----------------+-------------------+----------------------------------
3   Jobey          | Boeter            | jboeter0@mail.ru
4   York           | O'Deegan          | yodeegan1@nydailynews.com
5   Penelope       | Hexter            | phexter2@cbslocal.com
6   Chadd          | Franz-Schoninger  | cfranzschoninger3@google.com.hk
7   Vikky          | Eke               | veke4@elegantthemes.com
8   Marie-françoise | Currier          | acurrier0@economist.com
9   Bénédicte      | Dozdill           | cdozdill1@amazon.de
10  Görel          | Douthwaite        | edouthwaite2@feedburner.com
11  Bérengère      | Menendez          | amenendez3@dell.com
12  ...
13  (35 rows)
```

T17. Using `REVOKE`, remove the ability for the new user to run `SELECT *` on the view. Copy the code used and the responses below.

LANGUAGE: SQL
```
1  REVOKE select ON cust_email FROM user1, user2;
```

LANGUAGE: Pseudocode
```
1  REVOKE
```

T18. Try running the `SELECT *` as both users again. Copy the outputs below: `user1`

LANGUAGE: SQL

```sql
SELECT * FROM cust_email;
```

LANGUAGE: Pseudocode

```
ERROR:  permission denied for view cust_email
```

`user2`

LANGUAGE: SQL

```sql
SELECT * FROM cust_email;
```

LANGUAGE: Pseudocode

```
ERROR:  permission denied for view cust_email
```

T19. When logged in as the first new role, remove the 2nd new role. Copy the responses below:

LANGUAGE: SQL

```sql
DROP ROLE user2;
```

LANGUAGE: Pseudocode

```
ERROR:  permission denied to drop role
```

T20. As your normal user, the upxxxxxx one, remove both of the new roles. Copy the responses below:

LANGUAGE: SQL

```sql
-- user1
REVOKE all ON role FROM user1;
REVOKE all ON product FROM user1;
DROP DATABASE user1;
DROP ROLE user1;

-- user2
REVOKE all ON role FROM user2;
REVOKE all ON product FROM user2;
DROP DATABASE user2;
DROP ROLE user2;
```

LANGUAGE: Pseudocode

```
DROP ROLE
DROP ROLE
```

Page 22

PRACTICAL: Encryption

📅 2023-02-09   🕐 14:00   🎓   📍 FTC 3

Normally when we use encryption within a database, we pass the responsibility of encrypting the data to the front end service. This is to prevent the *encryption seed* from being visible within the database logs where a 'super-super admin' can see the insert statements and see the unencrypted data get inserted.

Tutor Task

Copy and run the following code.

```sql
-- create a new db for demo

create database secdb;

\c secdb
-- we can't copy and paste this next line of code at the same time as previous 2 lines!

-- Turn on encryption - It is not on by default.
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- bytea is a binary datatype
-- https://www.postgresql.org/docs/current/datatype-binary.html

CREATE TABLE secDemo(id serial PRIMARY KEY, pw bytea);

 -- insert into secdemo(pw) values ( encrypt( 'data', 'key', 'aes') );

INSERT INTO secdemo(pw)
VALUES (encrypt('Holiday!lips@', '56732', 'aes'));

select * from secdemo;

-- select decrypt(pw, 'key', 'aes') FROM secdemo;

select decrypt(pw, '56732', 'aes') as "decrypted version" FROM secdemo;

-- still bytea at this point

-- select convert_from(decrypt(pw, 'key', 'aes'), 'utf-8') FROM secdemo;
-- convert_from() converts from bytea to text

select convert_from(decrypt(pw, '56732', 'aes'), 'utf-8') as "converted from decrypted" FROM
    secdemo;
```

Student Tasks

T1. Make sure you are up to date with the practicals!
T2. Create a new database called `sec3`

```sql
CREATE DATABASE sec3;
```

```
2 \c sec3
```

T3. Turn encryption on in this new database.

LANGUAGE: SQL

```sql
1 CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

T4. Using the Tutor Task above, create a new table called `member` and add 5 rows of data. This table must hold first and last names along with the member's date of birth, (stored as a `date` datatype), a postcode and an encrypted password. Copy the code and data inserted below:

LANGUAGE: SQL

```sql
1  CREATE TABLE member(
2      id serial PRIMARY KEY,
3      fname VARCHAR(25) NOT NULL,
4      lname VARCHAR(25) NOT NULL,
5      dob date NOT NULL,
6      postcode VARCHAR(8) NOT NULL,
7      password bytea
8  );
9  -- insert values now
10 INSERT INTO member(fname, lname, dob, postcode, password) VALUES ('Dave', 'Davidson', '
       ↪ 2022-01-01', 'NE1 4EQ', encrypt('cheese123', '1234', 'aes'));
11 INSERT INTO member(fname, lname, dob, postcode, password) VALUES ('Fred', 'Fredrikson', '
       ↪ 2021-03-6', 'AB12 CDE', encrypt('mouse33', '1234', 'aes'));
12 INSERT INTO member(fname, lname, dob, postcode, password) VALUES ('Sue', 'Susan', '1972-05-02',
       ↪ 'BN35 7DQ', encrypt('secrue68', '1234', 'aes'));
13 INSERT INTO member(fname, lname, dob, postcode, password) VALUES ('Jane', 'Johnson', '
       ↪ 2012-01-01', 'FE43 8GG', encrypt('cake43', '1234', 'aes'));
14 INSERT INTO member(fname, lname, dob, postcode, password) VALUES ('Sam', 'Sampson', '2016-01-03
       ↪ ', 'HE8 0NH', encrypt('camping111', '1234', 'aes'));
```

T5. Once stored, print out the data for all of the rows. Copy below

LANGUAGE: SQL

```sql
1 SELECT * FROM member;
```

LANGUAGE: Pseudocode

```
1  id | fname |   lname    |    dob     | postcode |             password
2 ----+-------+------------+------------+----------+-----------------------------------
3   1 | Dave  | Davidson   | 2022-01-01 | NE1 4EQ  | \x2054eed5d9908d4d0dbb1d11777b9f2f
4   2 | Fred  | Fredrikson | 2021-03-06 | AB12 CDE | \x7669b0433719a86b2f2736f3ccee1757
5   3 | Sue   | Susan      | 1972-05-02 | BN35 7DQ | \x05d04a89e6159a6d79f6e035a1ad1931
6   4 | Jane  | Johnson    | 2012-01-01 | FE43 8GG | \xb10a802af936a09504f73b98a66e45ff
7   5 | Sam   | Sampson    | 2016-01-03 | HE8 0NH  | \xa46293e279f31027034993007197c256
8 (5 rows)
```

T6. Decrypt the values stored in the encrypted password attribute for all 5 rows.

LANGUAGE: SQL

```sql
1 SELECT id, convert_from(decrypt(password, '1234', 'aes'), 'utf-8') FROM member;
```

LANGUAGE: Pseudocode

```
1  id | convert_from
2 ----+--------------
```

```
3     1 | cheese123
4     2 | mouse33
5     3 | secrue68
6     4 | cake43
7     5 | camping111
8 (5 rows)
```

From Last week's lecture

T7. Connect to `dsd_22` and add a unique constraint to the customer table on the town column. Copy the code and output below:

```
LANGUAGE: SQL

1 \c dsd_22
2
3 ALTER TABLE customer ADD CONSTRAINT table_unique UNIQUE (town);
```

```
LANGUAGE: Unknown

1 ERROR:  could not create unique index "table_unique"
2 DETAIL:  Key (town)=(La Mohammedia) is duplicated.
```

T8. Using the manifest table, how many `prod_id` are there?

```
LANGUAGE: SQL

1 SELECT count(DISTINCT prod_id) FROM manifest;
```

```
LANGUAGE: Pseudocode

1  count
2 -------
3     76
4 (1 row)
```

T9. How many distinct `prod_id` are there?

```
LANGUAGE: SQL

1 SELECT count(DISTINCT prod_id) FROM product;
```

```
LANGUAGE: Pseudocode

1  count
2 -------
3    100
4 (1 row)
```

T10. How many orders in the manifest table include the product with the id of 24?

```
LANGUAGE: SQL

1 SELECT count(manifest_id) FROM manifest
2 WHERE prod_id=24;
```

```
LANGUAGE: Pseudocode
1   count
2  -------
3       4
4  (1 row)
```

T11. How many orders in the manifest table include the product with the id of 2?

```
LANGUAGE: SQL
1  SELECT count(manifest_id) FROM manifest
2  WHERE prod_id=2;
```

```
LANGUAGE: Pseudocode
1   count
2  -------
3       0
4  (1 row)
```

T12. Again, in the manifest table, what code could be used to give the following output:

```
LANGUAGE: Pseudocode
1  prod_id
2  ---------
3       100
4        99
5        97
6        95
7        94
8        93
9        92
10       91
```

Copy your answer below:

```
LANGUAGE: SQL
1  SELECT DISTINCT prod_id FROM manifest
2  ORDER BY prod_id DESC
3  LIMIT 8;
```

T13. Using alter table, add a check constraint to the `dsd_22` staff table. The check must check that the length of a postcode is over 5 characters long. Hint: the `length()` function will find out how long a value is. Copy the code below.

```
LANGUAGE: SQL
1  ALTER TABLE staff ADD CONSTRAINT postcode_length CHECK(length(postcode)> 5);
```

T14. Now add a new staff member to the staff table using the insert code snippet below.

```
LANGUAGE: SQL
1  INSERT INTO staff (staff_fname, staff_lname, addr1, addr2, town, postcode, home_email,
       ↪ work_email, ROLE)
2  VALUES ('Tiny',
3        'Smith',
4        '85 Lilly Way',
5        'Off Pole Lane',
6        'Southsea',
7        'PO98',
```

```
8          'tsmith@smiths.com',
9          'Tiny.Smith@dsd.com',
10         5);
```

T15. Copy the output below

```
LANGUAGE: Pseudocode

1 ERROR:  new row for relation "staff" violates check constraint "postcode_length"
2 DETAIL:  Failing row contains (12, Tiny, Smith, 85 Lilly Way, Off Pole Lane, Southsea, PO98
    ↪    , tsmith@smiths.com, Tiny.Smith@dsd.com, 5).
```

Dates

We can use dates in many ways.
Download the code from the folder code for practical and run it in your NORMAL database
- the one called upxxxxxx.

```
LANGUAGE: SQL

1  create table date_check (
2      id INT primary key,
3      first_name VARCHAR(50) not null,
4      last_name VARCHAR(50) not null,
5      email VARCHAR(50) not null,
6      joined DATE not null
7  );
8  insert into date_check (id, first_name, last_name, email, joined) values (1, 'Carie', 'Harling'
    ↪    , 'charling0@yale.edu', '2022-04-28');
9  insert into date_check (id, first_name, last_name, email, joined) values (2, 'Deina', 'Brennans
    ↪    ', 'dbrennans1@slashdot.org', '2022-04-08');
10 insert into date_check (id, first_name, last_name, email, joined) values (3, 'Devon', '
    ↪    Matijasevic', 'dmatijasevic2@economist.com', '2022-09-25');
11 insert into date_check (id, first_name, last_name, email, joined) values (4, 'Wald', '
    ↪    Kleinhausen', 'wkleinhausen3@trellian.com', '2022-08-13');
12 insert into date_check (id, first_name, last_name, email, joined) values (5, 'Cammie', 'Womack'
    ↪    , 'cwomack4@who.int', '2022-06-19');
13 insert into date_check (id, first_name, last_name, email, joined) values (6, 'Cross', '
    ↪    MacCallam', 'cmaccallam5@tuttocitta.it', '2023-02-05');
14 insert into date_check (id, first_name, last_name, email, joined) values (7, 'Maris', '
    ↪    Flitcroft', 'mflitcroft6@clickbank.net', '2022-07-12');
15 insert into date_check (id, first_name, last_name, email, joined) values (8, 'Peggy', '
    ↪    Gasquoine', 'pgasquoine7@ebay.com', '2022-07-22');
16 insert into date_check (id, first_name, last_name, email, joined) values (9, 'Kermit', 'Ninnoli
    ↪    ', 'kninnoli8@smh.com.au', '2022-10-10');
17 insert into date_check (id, first_name, last_name, email, joined) values (10, 'Frieda', '
    ↪    Glassford', 'fglassford9@wufoo.com', '2022-08-26');
18 insert into date_check (id, first_name, last_name, email, joined) values (11, 'Lanie', 'Boggish
    ↪    ', 'lboggisha@comcast.net', '2022-03-31');
19 insert into date_check (id, first_name, last_name, email, joined) values (12, 'Amelie', '
    ↪    Timmons', 'atimmonsb@wp.com', '2022-11-23');
20 insert into date_check (id, first_name, last_name, email, joined) values (13, 'Portia', '
    ↪    Nielson', 'pnielsonc@wix.com', '2022-10-10');
21 insert into date_check (id, first_name, last_name, email, joined) values (14, 'Sara-ann', '
    ↪    Ellens', 'sellensd@chronoengine.com', '2022-06-15');
22 insert into date_check (id, first_name, last_name, email, joined) values (15, 'Bob', 'Larcombe'
    ↪    , 'blarcombee@dailymotion.com', '2022-06-28');
23 insert into date_check (id, first_name, last_name, email, joined) values (16, 'Celestyn', '
    ↪    Wickenden', 'cwickendenf@prnewswire.com', '2022-06-15');
24 insert into date_check (id, first_name, last_name, email, joined) values (17, 'Rina', 'Dymoke',
    ↪    'rdymokeg@discuz.net', '2022-07-19');
25 insert into date_check (id, first_name, last_name, email, joined) values (18, 'Isadora', '
    ↪    Haughey', 'ihaugheyh@sfgate.com', '2022-07-31');
26 insert into date_check (id, first_name, last_name, email, joined) values (19, 'Demetria', 'Neem
    ↪    ', 'dneemi@jiathis.com', '2022-05-08');
27 insert into date_check (id, first_name, last_name, email, joined) values (20, 'Feliza', 'Gras',
    ↪    'fgrasj@printfriendly.com', '2022-03-19');
```

T16. Select the last names and the date they joined and copy the results below.

LANGUAGE: SQL

```sql
SELECT last_name, joined FROM date_check;
```

LANGUAGE: Pseudocode

```
  last_name   |   joined
--------------+------------
 Harling      | 2022-04-28
 Brennans     | 2022-04-08
 Matijasevic  | 2022-09-25
 Kleinhausen  | 2022-08-13
 Womack       | 2022-06-19
 MacCallam    | 2023-02-05
 Flitcroft    | 2022-07-12
 Gasquoine    | 2022-07-22
 Ninnoli      | 2022-10-10
 Glassford    | 2022-08-26
 Boggish      | 2022-03-31
 Timmons      | 2022-11-23
 Nielson      | 2022-10-10
 Ellens       | 2022-06-15
 Larcombe     | 2022-06-28
 Wickenden    | 2022-06-15
 Dymoke       | 2022-07-19
 Haughey      | 2022-07-31
 Neem         | 2022-05-08
 Gras         | 2022-03-19
(20 rows)
```

T17. Now sort them into `last_name` order. Copy the results below:

LANGUAGE: SQL

```sql
SELECT last_name, joined FROM date_check
ORDER BY last_name;
```

LANGUAGE: Pseudocode

```
  last_name   |   joined
--------------+------------
 Boggish      | 2022-03-31
 Brennans     | 2022-04-08
 Dymoke       | 2022-07-19
 Ellens       | 2022-06-15
 Flitcroft    | 2022-07-12
 Gasquoine    | 2022-07-22
 Glassford    | 2022-08-26
 Gras         | 2022-03-19
 Harling      | 2022-04-28
 Haughey      | 2022-07-31
 Kleinhausen  | 2022-08-13
 Larcombe     | 2022-06-28
 MacCallam    | 2023-02-05
 Matijasevic  | 2022-09-25
 Neem         | 2022-05-08
 Nielson      | 2022-10-10
 Ninnoli      | 2022-10-10
 Timmons      | 2022-11-23
 Wickenden    | 2022-06-15
 Womack       | 2022-06-19
(20 rows)
```

T18. What happens if we sort by a column we are not displaying? Copy the output below:

LANGUAGE: SQL

```sql
1 SELECT last_name, joined FROM date_check
2 ORDER BY email;
```

```
LANGUAGE: Pseudocode
1    last_name   |   joined
2  ------------+------------
3   Timmons     | 2022-11-23
4   Larcombe    | 2022-06-28
5   Harling     | 2022-04-28
6   MacCallam   | 2023-02-05
7   Wickenden   | 2022-06-15
8   Womack      | 2022-06-19
9   Brennans    | 2022-04-08
10  Matijasevic | 2022-09-25
11  Neem        | 2022-05-08
12  Glassford   | 2022-08-26
13  Gras        | 2022-03-19
14  Haughey     | 2022-07-31
15  Ninnoli     | 2022-10-10
16  Boggish     | 2022-03-31
17  Flitcroft   | 2022-07-12
18  Gasquoine   | 2022-07-22
19  Nielson     | 2022-10-10
20  Dymoke      | 2022-07-19
21  Ellens      | 2022-06-15
22  Kleinhausen | 2022-08-13
23 (20 rows)
```

T19. How would you get a list of people who joined after October 1st 2022?

```sql
1 SELECT first_name, last_name, joined FROM date_check
2 WHERE joined > '2022-10-01';
```

```
LANGUAGE: Pseudocode
1  first_name | last_name |   joined
2  -----------+-----------+------------
3   Cross      | MacCallam | 2023-02-05
4   Kermit     | Ninnoli   | 2022-10-10
5   Amelie     | Timmons   | 2022-11-23
6   Portia     | Nielson   | 2022-10-10
7 (4 rows)
```

T20. Order the output by joined date order. Copy this output below.

```sql
1 SELECT first_name, last_name, joined FROM date_check
2 WHERE joined > '2022-10-01'
3 ORDER BY joined ASC;
```

```
LANGUAGE: Pseudocode
1  first_name | last_name |   joined
2  -----------+-----------+------------
3   Kermit     | Ninnoli   | 2022-10-10
4   Portia     | Nielson   | 2022-10-10
5   Amelie     | Timmons   | 2022-11-23
6   Cross      | MacCallam | 2023-02-05
7 (4 rows)
```

T21. Now order the output from 20 so that the joined date is the first order THEN try to order by the `last_name`. Copy this code & output below.

```sql
LANGUAGE: SQL
1 SELECT first_name, last_name, joined FROM date_check
2 WHERE joined > '2022-10-01'
3 ORDER BY joined, last_name ASC;
```

```
LANGUAGE: Pseudocode
1  first_name | last_name |   joined
2 ------------+-----------+------------
3  Portia     | Nielson   | 2022-10-10
4  Kermit     | Ninnoli   | 2022-10-10
5  Amelie     | Timmons   | 2022-11-23
6  Cross      | MacCallam | 2023-02-05
7 (4 rows)
```

T22. We can use the `between` keyword to find results that fall between two dates. Output all data for the people who joined between April 20th 2022 and November 30th 2022. Copy the output below.

```sql
LANGUAGE: SQL
1 SELECT first_name, last_name, joined FROM date_check
2 WHERE joined BETWEEN '2022-04-20' AND '2022-11-30'
3 ORDER BY joined, last_name ASC;
```

```
LANGUAGE: Pseudocode
1   first_name |  last_name  |   joined
2  ------------+-------------+------------
3   Carie      | Harling     | 2022-04-28
4   Demetria   | Neem        | 2022-05-08
5   Sara-ann   | Ellens      | 2022-06-15
6   Celestyn   | Wickenden   | 2022-06-15
7   Cammie     | Womack      | 2022-06-19
8   Bob        | Larcombe    | 2022-06-28
9   Maris      | Flitcroft   | 2022-07-12
10  Rina       | Dymoke      | 2022-07-19
11  Peggy      | Gasquoine   | 2022-07-22
12  Isadora    | Haughey     | 2022-07-31
13  Wald       | Kleinhausen | 2022-08-13
14  Frieda     | Glassford   | 2022-08-26
15  Devon      | Matijasevic | 2022-09-25
16  Portia     | Nielson     | 2022-10-10
17  Kermit     | Ninnoli     | 2022-10-10
18  Amelie     | Timmons     | 2022-11-23
19 (16 rows)
```

Page 23

LECTURE: Coursework Feedback & Functions

📅 2023-02-23   🕐 13:00   🎓 Mark   📍 RB LT1

Text Functions

ASCII()

The `ASCII()` function returns the ASCII value of a character. The function expects 1 character, any additional characters passed to it will be ignored.

```sql
LANGUAGE: SQL
1 SELECT ASCII ('A');
```

will return: 65.

CHR()

The `CHR()` function performs the inverse of `ASCII()`, it returns the character represented by the ASCII code passed in.

```sql
LANGUAGE: SQL
1 SELECT CHR(65);
```

will return: `A`.

INITCAP()

The `INITCAP()` function converts the first letter of each word in the string passed into it into a capital, this is known as *title case*.

```sql
LANGUAGE: SQL
1 SELECT INITCAP('hi my name is dave);
```

will return: `Hi My Name Is Dave`.

POSITION()

The `POSITION()` function returns the location of a substring in a string.

```sql
LANGUAGE: SQL
1 SELECT POSITION('B' IN 'A B C');
```

will return: 3. Note that the indexing is 1 based and that the function will only return the first occurrence of the target string in the search string.

FORMAT()

The `FORMAT()` function formats arguments based on an input format string. It is similar to the C function `sprintf`.

CONCAT()

The `CONCAT()` glues one string to another. Non-attribute strings (eg `' '`) can be put between attribute names to add spaces in. You have to specify the separator between each attribute.

```sql
LANGUAGE: SQL
1  SELECT CONCAT(cust_fname, ' ', cust_lname);
```

will return: `Fred Fredrikson`.

CONCAT_WS()

The `CONCAT_WS()` function works much the same as the `CONCAT()` in that it concatenates strings together. However the `CONCAT_WS()` function only requires the separator to be specified once, as the first parameter in the bracket.

```sql
LANGUAGE: SQL
1  SELECT CONCAT_WS(' ', cust_fname, cust_lname);
```

will return: `Fred Fredrikson`.

Date Functions

We have already used `NOW()` (which returns the date and time at which the command is sent). `CURRENT_DATE` returns the current date (note that it doesn't have brackets).

DATE_PART()

The `DATE_PART()` function allows us to extract part of a date, for example just the year.

```sql
LANGUAGE: SQL
1  SELECT DATE_PART('year', NOW());
```

will return 2023.
Among others, we can request: `decade`, `year`, `month`, `day`, `hour`, `minute`, `second`, `day of week`.

AGE()

This returns the difference between the dates passed as parameters. Its explored more in this weeks practical.

CURRENT_TIME

Returns the current time.

DATE_TRUNC()

The `DATE_TRUNC()` function truncates the date to a specified level (levels are the same as for `DATE_PART()`).

LANGUAGE: SQL

```sql
SELECT DATE_TRUNC('year', NOW());
```

will return 2023-01-01 00:00:00+00. This is not used particularly often.

Page 24

PRACTICAL: Security & Functions

📅 2023-02-23   🕐 14:00   🎓 Mark   📍 FTC 3

Security

T0. Create a new table in your `upxxxxxxx` database called `users` with the following columns.

```
id - int primary key (user identifier)
first_name - varchar(30) (user first name)
last_name - varchar(40) (user last name)
email - varchar(100) (user email address)
password - text (user password - Will be stored encrypted)
```

```sql
LANGUAGE: SQL
1  CREATE TABLE users(
2      id INT PRIMARY KEY,
3      first_name VARCHAR(30),
4      last_name VARCHAR(40),
5      email VARCHAR(100),
6      password text
7  );
```

T1. Transfer `users.csv` (downloaded from Moodle) to the vm.

`PS C:\Users\thoma\Downloads> scp .\users.csv up2108121@up2108121.myvm.port.ac.uk:~`

T2. Assuming you have transferred the csv into your home directory run the following code

```
LANGUAGE: Pseudocode
1  \copy users(id, first_name, last_name, email, password) from '/home/up2108121/users.csv'
   ↪ DELIMITER ',' CSV HEADER
```

T3. You should get the response COPY 500. Check that the data has been entered correctly with

```sql
LANGUAGE: SQL
1  SELECT * FROM users LIMIT 5;
```

```
LANGUAGE: Pseudocode
1   id | first_name | last_name |           email            | password
2  ----+------------+-----------+----------------------------+------------
3    1 | Tomlin     | Hardage   | thardage0@chronoengine.com | lE50Tm63
4    2 | Shea       | Bergeon   | sbergeon1@liveinternet.ru  | j5KTPP2z
5    3 | Matilde    | Jendrusch | mjendrusch2@ftc.gov        | 9J5pKR6
6    4 | Hillyer    | Machans   | hmachans3@fda.gov          | NXHF8K
```

```
7    5 | Cassaundra | Michiel   | cmichiel4@vimeo.com         | EIvy2EUtD0
8  (5 rows)
```

T4. Run the following code

```
LANGUAGE: SQL
1  CREATE EXTENSION PGCRYPTO;
2  update users set password = crypt(password, gen_salt('bf'));
3  -- line below tests lines above
4  SELECT * FROM users limit 5;
```

```
LANGUAGE: Pseudocode
1   id | first_name | last_name |            email           |                      password
2  ----+------------+-----------+----------------------------+-----------------------------------------------------
       ↪
3    1 | Tomlin     | Hardage   | thardage0@chronoengine.com | 2a06Pu5zrUTeqTxQ9/
       ↪ cvxWgugeIPN1zBwQtaPh3hYaVNjJl4.pKEmEtFy
4    2 | Shea       | Bergeon   | sbergeon1@liveinternet.ru  | 2
       ↪ a067impB1AFnhTzjhJzNnBMkekmGWJkNAtT3/pJdWVbqsvPbHUa/fnHG
5    3 | Matilde    | Jendrusch | mjendrusch2@ftc.gov        | 2a06btIlBtdBIBI9lwbDQKmfGuvp.f5.
       ↪ lursUWV6VHPV1AOlGWArAuQha
6    4 | Hillyer    | Machans   | hmachans3@fda.gov          | 2
       ↪ a063y9DJolYQU1tYk5imvTfBOMmfQzaeardWS.GLrO4JPAq60f7wV5Mi
7    5 | Cassaundra | Michiel   | cmichiel4@vimeo.com        | 2a061oo7WRPUd/TKI.
       ↪ cPjNGQQerNKOtUevebYBOcXwIssQt46EV4TohGy
8  (5 rows)
9  [dollar signs removed from above]
```

You should see that the passwords are now encrypted.
We have encrypted the passwords and we can no longer get to see the decrypted values.
The safety in this method is that there is one way hashing protecting them. Firstly, select
the details of the user with id 304;

```
LANGUAGE: SQL
1  SELECT first_name, last_name, password from users where id = 304;
```

```
LANGUAGE: Pseudocode
1   first_name | last_name |                       password
2  ------------+-----------+-------------------------------------------------------------
3   Corette    | Peaseman  | 2a06ru.N1no95BZTozd.0Hab8uCyUW8wZOXwGN2Uksga6vjsZaW.g9CI2
4  (1 row)
5  [dollar signs removed]
```

Now we select the details again but we are sending in the password that a user has en-
tered to try to log in. If the decrypted password matches the one we are sending in we
get a row of data back.

T5. Run the following command

```
LANGUAGE: SQL
1  SELECT id,
2      first_name,
3      last_name
4  FROM users
5  WHERE email = 'cpeaseman8f@simplemachines.org'
6    AND password = crypt('nr4kjyxW', password);
```

```
LANGUAGE: Pseudocode
1  id  | first_name | last_name
2 -----+------------+-----------
3  304 | Corette    | Peaseman
4 (1 row)
```

The DBMS will look at the value of the password we are sending, `nr4kjyxW`, and it will do the decryption to see if it matches. If it does it will send us back the data we requested. At no time do we see the stored unencrypted value of the password.

T6. What do we get if we send in an incorrect password?

```
LANGUAGE: SQL
1 SELECT id,
2     first_name,
3     last_name
4 FROM users
5 WHERE email = 'cpeaseman8f@simplemachines.org'
6   AND password = crypt('nr4kjyxW!', password);
```

```
LANGUAGE: Unknown
1  id | first_name | last_name
2 ----+------------+-----------
3 (0 rows)
```

T7. Add a new user to the table but send in an encrypted version of their password:

```
LANGUAGE: SQL
1 INSERT INTO users
2 VALUES(600,
3     'Flubby',
4     'Foster',
5     'f_f@fmail.com',
6     crypt('thisismypassword1', gen_salt('bf')));
```

T8. Now select the password that has just been entered:

```
LANGUAGE: SQL
1 SELECT password
2 FROM users
3 WHERE id = 600;
```

```
LANGUAGE: Unknown
1                         password
2 -----------------------------------------------------------
3  2a06UvKeG6bv6poLkpP9IXRlOeE/V7X524BmamixwIHHqMtsBhuLZmSt.
4 (1 row)
5 [dollar signs removed]
```

Add another user with the id of 601 that uses the same very bad password as Flubby Foster.

```
LANGUAGE: SQL
1     INSERT INTO users
2     VALUES(601,
```

```
3          'Freddie',
4          'Andrews',
5          'f_a@fmail.com',
6          crypt('thisismypassword1', gen_salt('bf')));
```

Now compare the encrypted passwords, by selecting just the id and passwords for users 600 and 601. Copy the output below. (They should be different, despite being the same password). This is what gen_salt() does for us. It puts a random salt value into the encrypted text. The random text is up to 128 characters long.

LANGUAGE: SQL

```
1 SELECT id, password
2 FROM users
3 WHERE id >= 600;
```

LANGUAGE: Pseudocode

```
1  id  |                           password
2 -----+-------------------------------------------------------------
3  600 | 2a06UvKeG6bv6poLkpP9IXRlOeE/V7X524BmamixwIHHqMtsBhuLZmSt.
4  601 | 2a06sGalLa0JGaFZ99LQn.S7w.1qWGSv8soO68qlcGwTQ6.bWoqDhhYqi
5 (2 rows)
```

Functions

In order to use the next set of data we need to change the date style. Use the following code:

LANGUAGE: SQL

```
1 SET DATESTYLE TO EUROPEAN;
```

This will make Postgresql expect dates to be in the DD MM YYYY format.
Now run the following code to create a new table:

LANGUAGE: SQL

```
1 create table users2 (
2     id INT primary key,
3     first_name VARCHAR(20) not null,
4     last_name VARCHAR(30) not null,
5     email VARCHAR(55) not null,
6     dob DATE not null
7 );
8
9 insert into users2 (id, first_name, last_name, email, dob) values (1, 'Zaria', 'Coot', '
      ↪ zcoot0@baidu.com', '07-11-2002');
10 insert into users2 (id, first_name, last_name, email, dob) values (2, 'Lucho', 'Holbie', '
      ↪ lholbie1@adobe.com', '09-03-2000');
11 insert into users2 (id, first_name, last_name, email, dob) values (3, 'Sherlock', 'Shoveller',
      ↪ 'sshoveller2@zdnet.com', '10-10-2002');
12 insert into users2 (id, first_name, last_name, email, dob) values (4, 'Shelba', 'Riach', '
      ↪ sriach3@xing.com', '09-11-2002');
13 insert into users2 (id, first_name, last_name, email, dob) values (5, 'Joseph', 'Lynn', '
      ↪ jlynn4@weather.com', '25-11-2003');
14 insert into users2 (id, first_name, last_name, email, dob) values (6, 'Haroun', 'De Haven', '
      ↪ hdehaven5@vistaprint.com', '23-06-2003');
15 insert into users2 (id, first_name, last_name, email, dob) values (7, 'Fidelio', 'Lindeboom', '
      ↪ flindeboom6@salon.com', '01-11-2003');
16 insert into users2 (id, first_name, last_name, email, dob) values (8, 'Sheryl', 'Kubat', '
      ↪ skubat7@fc2.com', '07-11-2001');
17 insert into users2 (id, first_name, last_name, email, dob) values (9, 'Lisha', 'Skillern', '
      ↪ lskillern8@goo.gl', '10-09-2003');
```

```
18 insert into users2 (id, first_name, last_name, email, dob) values (10, 'Aubrie', 'Sedgmond', '
   ↪ asedgmond9@nymag.com', '02-01-2004');
19 insert into users2 (id, first_name, last_name, email, dob) values (11, 'Thorvald', 'Blincko', '
   ↪ tblinckoa@mozilla.org', '21-11-2001');
20 insert into users2 (id, first_name, last_name, email, dob) values (12, 'Quincy', 'Keeltagh', '
   ↪ qkeeltaghb@multiply.com', '04-12-2002');
21 insert into users2 (id, first_name, last_name, email, dob) values (13, 'Javier', 'Camel', '
   ↪ jcamelc@weather.com', '15-11-2001');
22 insert into users2 (id, first_name, last_name, email, dob) values (14, 'Ann-marie', 'Scholtz',
   ↪ 'ascholtzd@hp.com', '03-07-2001');
23 insert into users2 (id, first_name, last_name, email, dob) values (15, 'Camel', 'Radmer', '
   ↪ cradmere@about.com', '06-02-2001');
24 insert into users2 (id, first_name, last_name, email, dob) values (16, 'Friedrich', 'Truluck',
   ↪ 'ftruluckf@soup.io', '04-09-2000');
25 insert into users2 (id, first_name, last_name, email, dob) values (17, 'Nichole', 'Rowbottam',
   ↪ 'nrowbottamg@state.tx.us', '10-09-2001');
26 insert into users2 (id, first_name, last_name, email, dob) values (18, 'Kory', 'Agglio', '
   ↪ kagglioh@i2i.jp', '20-04-2000');
27 insert into users2 (id, first_name, last_name, email, dob) values (19, 'Bella', 'O''Brallaghan'
   ↪ , 'bobrallaghani@bravesites.com', '01-10-2002');
28 insert into users2 (id, first_name, last_name, email, dob) values (20, 'Francine', 'Rantoul', '
   ↪ frantoulj@e-recht24.de', '24-08-2001');
```

You have just inserted users into a table that has a column called dob. This stores a date of birth in ISO format, `YYY-MM-DD` but the code has entered dates in UK / European format.

T9. Check the format stored in the table. Display the dob for user with id number 10

```
LANGUAGE: SQL
1 SELECT dob FROM users2 WHERE id=10;
```

```
LANGUAGE: Pseudocode
1     dob
2 ------------
3  2004-01-02
4 (1 row)
```

Age function 1

T10. How old is the user with id number 1 TODAY? Use the `age()` function. The format for this method is `age(TIMESTAMP)` where `TIMESTAMP` can be an attribute name. This takes the current date by default to calculate the age today.

```
LANGUAGE: SQL
1 SELECT first_name, AGE(dob) FROM users2 WHERE id=10;
```

```
LANGUAGE: Pseudocode
1  first_name |          age
2 ------------+------------------------
3  Aubrie     | 19 years 1 mon 21 days
4 (1 row)
```

Age function 2

T11. How old will the user be on 30th June 2035? The format for this method is `age(TIMESTAMP,TIMESTAMP` where TIMESTAMP can be an attribute name OR date.

```
LANGUAGE: SQL
1 SELECT dob, age('30-06-2035', dob) FROM users2 where id=1;
```

```
LANGUAGE: Pseudocode
1    dob     |              age
2 ------------+-------------------------
3  2002-11-07 | 32 years 7 mons 23 days
4 (1 row)
```

More on Dates

T12. Run the following code to add a new column to `users2`.

```
LANGUAGE: SQL
1 ALTER TABLE users2 ADD COLUMN joined date DEFAULT CURRENT_DATE;
```

This will add a new column called joined and it has a DEFAULT value set to `CURRENT_DATE`. This will put in a value automatically if a value is not inserted by the user.

T13. The `users2` table was created with the expectation that the `INSERT` code will provide a value for the ID, it is not set to serial. How will you find the next free id number? Copy the code and result below:

```
LANGUAGE: SQL
1 SELECT (max(id)+1) AS "NEXT ID" from users2;
```

```
LANGUAGE: Pseudocode
1  NEXT ID
2 ---------
3       21
4 (1 row)
```

T14. Add 5 new users to the `users2` table. Put a value in for the joined attribute for 2 and do not put one in for the other 3. Copy the code below:

```
LANGUAGE: SQL
1 insert into users2 (id, first_name, last_name, email, dob) values (21, 'Renell', 'Cogle', '
     ↪ rcogle0@wiley.com', '2022-02-06');
2 insert into users2 (id, first_name, last_name, email, dob, joined) values (22, 'Isabeau', '
     ↪ Gameson', 'igameson1@ucoz.com', '2023-01-25', '2022-02-04');
3 insert into users2 (id, first_name, last_name, email, dob) values (23, 'Benito', 'Celli', '
     ↪ bcelli2@xinhuanet.com', '2022-07-07');
4 insert into users2 (id, first_name, last_name, email, dob) values (24, 'Abra', 'Colbourn', '
     ↪ acolbourn3@cpanel.net', '2022-06-07');
5 insert into users2 (id, first_name, last_name, email, dob, joined) values (25, 'Paolo', 'Libby'
     ↪ , 'plibby4@unc.edu', '2022-05-04', '2022-12-13');
```

T15. Retrieve all of the data in the users2 table. How many have today's date in the joined table? How many are blank?

```
LANGUAGE: Pseudocode
1  id | first_name | last_name  |              email              |    dob     |   joined
2 ----+------------+------------+---------------------------------+------------+------------
```

```
3    1 | Zaria       | Coot        | zcoot0@baidu.com              | 2002-11-07 | 2023-02-23
4    2 | Lucho       | Holbie      | lholbie1@adobe.com           | 2000-03-09 | 2023-02-23
5    3 | Sherlock    | Shoveller   | sshoveller2@zdnet.com        | 2002-10-10 | 2023-02-23
6    4 | Shelba      | Riach       | sriach3@xing.com             | 2002-11-09 | 2023-02-23
7    5 | Joseph      | Lynn        | jlynn4@weather.com           | 2003-11-25 | 2023-02-23
8    6 | Haroun      | De Haven    | hdehaven5@vistaprint.com     | 2003-06-23 | 2023-02-23
9    7 | Fidelio     | Lindeboom   | flindeboom6@salon.com        | 2003-11-01 | 2023-02-23
10   8 | Sheryl      | Kubat       | skubat7@fc2.com              | 2001-11-07 | 2023-02-23
11   9 | Lisha       | Skillern    | lskillern8@goo.gl            | 2003-09-10 | 2023-02-23
12  10 | Aubrie      | Sedgmond    | asedgmond9@nymag.com         | 2004-01-02 | 2023-02-23
13  11 | Thorvald    | Blincko     | tblinckoa@mozilla.org        | 2001-11-21 | 2023-02-23
14  12 | Quincy      | Keeltagh    | qkeeltaghb@multiply.com      | 2002-12-04 | 2023-02-23
15  13 | Javier      | Camel       | jcamelc@weather.com          | 2001-11-15 | 2023-02-23
16  14 | Ann-marie   | Scholtz     | ascholtzd@hp.com             | 2001-07-03 | 2023-02-23
17  15 | Camel       | Radmer      | cradmere@about.com           | 2001-02-06 | 2023-02-23
18  16 | Friedrich   | Truluck     | ftruluckf@soup.io            | 2000-09-04 | 2023-02-23
19  17 | Nichole     | Rowbottam   | nrowbottamg@state.tx.us      | 2001-09-10 | 2023-02-23
20  18 | Kory        | Agglio      | kagglioh@i2i.jp              | 2000-04-20 | 2023-02-23
21  19 | Bella       | O'Brallaghan | bobrallaghani@bravesites.com | 2002-10-01 | 2023-02-23
22  20 | Francine    | Rantoul     | frantoulj@e-recht24.de       | 2001-08-24 | 2023-02-23
23  21 | Renell      | Cogle       | rcogle0@wiley.com            | 2022-02-06 | 2023-02-23
24  23 | Benito      | Celli       | bcelli2@xinhuanet.com        | 2022-07-07 | 2023-02-23
25  24 | Abra        | Colbourn    | acolbourn3@cpanel.net        | 2022-06-07 | 2023-02-23
26  22 | Isabeau     | Gameson     | igameson1@ucoz.com           | 2023-01-25 | 2022-02-04
27  25 | Paolo       | Libby       | plibby4@unc.edu              | 2022-05-04 | 2022-12-13
28 (25 rows)
```

T16. You have been asked to find out which users in the `users2` table do not have a joined date. Copy your code to find this info and the results from your code below.

LANGUAGE: SQL
```sql
SELECT id FROM users2 where joined=NULL;
```

LANGUAGE: Unknown
```
 id
----
(0 rows)
```

T17. Why do you get the result you get?
As when adding the constraint, Postgres will automatically populate all the empty values with the current date.

Challenge from Lecture

Write a query that searches through the customer email addresses in `dsd_22` database and return a list of all the email domains

LANGUAGE: SQL
```sql
SELECT substring(email, position('@' in email), length(email)) FROM customer;
```

LANGUAGE: Unknown
```
     substring
-----------------------
 @mail.ru
 @nydailynews.com
 @cbslocal.com
 @google.com.hk
 @elegantthemes.com
 @economist.com
```

```
 9  @amazon.de
10  @feedburner.com
11  @dell.com
12  @blinklist.com
13  @mail.ca
14  @tiny.cc
15  @chron.com
16  @wp.com
17  @webmd.com
18  @prweb.com
19  @wordpress.org
20  @amazon.de
21  @geocities.jp
22  @shop-pro.jp
23  @dell.com
24  @google.cn
25  @google.wh
26  @google.wh
27  @google.ca
28  @tiny.cc
29  @networkadvertising.org
30  @cafepress.com
31  @imdb.com
32  @dion.ne.jp
33  @typepad.com
34  @uiuc.edu
35  @arstechnica.com
36  @rambler.ru
37  @sfgate.com
38  (35 rows)
```

Page 25

PRACTICAL: Functions

📅 2023-03-02   👁 Self directed practical

NB: This practical is self directed due to staff sickness.

Functions

The initial tasks are to be completed in the `dsd_22` database.

T1. Write a query that will combine the customer's first name, their last name and the email address in a single column. Give this column a sensible name. Copy the code and top 5 results output below.

```sql
LANGUAGE: SQL

SELECT CONCAT_WS(' ', cust_fname, cust_lname, email) AS "Customer Information" FROM customer
    ↪ LIMIT 5;
```

```
LANGUAGE: Pseudocode

                  Customer Information
-------------------------------------------------------
 Jobey Boeter jboeter0@mail.ru
 York O'Deegan yodeegan1@nydailynews.com
 Penelope Hexter phexter2@cbslocal.com
 Chadd Franz-Schoninger cfranzschoninger3@google.com.hk
 Vikky Eke veke4@elegantthemes.com
(5 rows)
```

T2. Find all products that have the character 5 in their product name

```sql
LANGUAGE: SQL

SELECT prod_name, prod_id FROM product WHERE POSITION('5' in prod_name) > 0;
```

```
LANGUAGE: Pseudocode

                    prod_name                  | prod_id
-----------------------------------------------+---------
 Realigned 5th generation artificial intelligence |   26
 Switchable 5th generation parallelism         |   49
(2 rows)
```

T2a. For each product, find the position of the 5 in the name. Copy the code to both parts of this question below.

```sql
LANGUAGE: SQL

SELECT prod_name, prod_id, POSITION('5' in prod_name) AS "5 pos" FROM product WHERE POSITION('5
    ↪ ' in prod_name) > 0;
```

```
LANGUAGE: Pseudocode
1                       prod_name                 | prod_id | 5 pos
2 -------------------------------------------------+---------+-------
3  Realigned 5th generation artificial intelligence |      26 |    11
4  Switchable 5th generation parallelism            |      49 |    12
5 (2 rows)
```

T3. Can you write a query that combines the two queries in 2 and 2.a ? Copy the code and output below.

See above

T4. Write a query that returns the first and last names of the staff members. You need to put the word 'Dear' in front of the names.

```
LANGUAGE: SQL
1 SELECT CONCAT_WS(' ', 'Dear', staff_fname, staff_lname) AS "NAMES" FROM staff;
```

```
LANGUAGE: Pseudocode
1          NAMES
2 ------------------------
3  Dear Niel Welsby
4  Dear Nikoletta Shrimpton
5  Dear Montgomery Housegoe
6  Dear Hanan Gloster
7  Dear Janeva Gillicuddy
8  Dear Aura Clewlowe
9  Dear Nell Olsson
10  Dear Harriette Fewster
11  Dear Jillene Revitt
12  Dear Tim Illem
13  Dear Kinsley Boick
14 (11 rows)
```

Using the `users2` table from last week

T5. The company is ten years old in November and are looking to send an email to all users who also have birthdays in November to celebrate this anniversary. Using one of the date functions we discussed last week, extract the first and last names of users who have birthdays in November. The output should put the first and last names of the users together in a useful format. Copy the code and output below.

```
LANGUAGE: SQL
1 SELECT CONCAT_WS(' ', first_name, last_name) AS "Name", dob FROM users2 WHERE DATE_PART('month'
    ↪ , dob) = 11;
```

```
LANGUAGE: Unknown
1         Name       |    dob
2 ------------------+------------
3  Zaria Coot        | 2002-11-07
4  Shelba Riach      | 2002-11-09
5  Joseph Lynn       | 2003-11-25
6  Fidelio Lindeboom | 2003-11-01
7  Sheryl Kubat      | 2001-11-07
8  Thorvald Blincko  | 2001-11-21
9  Javier Camel      | 2001-11-15
10 (7 rows)
```

T6. Which function will give the ascii value of the character !

```
ASCII()
```

T6a. What is the value? Copy your code and answer below.

```sql
LANGUAGE: SQL
1 SELECT ASCII('!');
```

```
LANGUAGE: Pseudocode
1  ascii
2 -------
3     33
4 (1 row)
```

T7. What character does the ascii value 300 define? Copy your code and the answer below.

```sql
LANGUAGE: SQL
1 SELECT CHR(300);
```

```
LANGUAGE: Pseudocode
1  chr
2 -----
3  Ĭ
4 (1 row)
```

T8. What character does the ascii value 5000 define? Copy your code and the answer below. A screenshot of the output will be needed for this one!

```sql
LANGUAGE: SQL
1 SELECT CHR(5000);
```

```
LANGUAGE: Pseudocode
1  chr
2 -----
3
4 (1 row)
```

T9. Calculate the age of all the users in the users2 table. Write a query that will return their first and last name along with their age. Put the results into age order with the eldest at the top.

```sql
LANGUAGE: SQL
1 SELECT CONCAT_WS(' ', first_name, last_name) AS "NAME", AGE(NOW(), dob) AS "AGE"
2 FROM users2
3 ORDER BY AGE(NOW(), dob) DESC;
```

```
LANGUAGE: Pseudocode
1        NAME        |              AGE
```

```
2  -------------------+---------------------------------------
3   Lucho Holbie       | 22 years 11 mons 24 days 14:04:46.41464
4   Kory Agglio        | 22 years 10 mons 12 days 14:04:46.41464
5   Friedrich Truluck  | 22 years 5 mons 28 days 14:04:46.41464
6   Camel Radmer       | 22 years 24 days 14:04:46.41464
7   Ann-marie Scholtz  | 21 years 7 mons 30 days 14:04:46.41464
8   Francine Rantoul   | 21 years 6 mons 9 days 14:04:46.41464
9   Nichole Rowbottam  | 21 years 5 mons 22 days 14:04:46.41464
10  Sheryl Kubat       | 21 years 3 mons 25 days 14:04:46.41464
11  Javier Camel       | 21 years 3 mons 17 days 14:04:46.41464
12  Thorvald Blincko   | 21 years 3 mons 11 days 14:04:46.41464
13  Bella O'Brallaghan | 20 years 5 mons 1 day 14:04:46.41464
14  Sherlock Shoveller | 20 years 4 mons 23 days 14:04:46.41464
15  Zaria Coot         | 20 years 3 mons 25 days 14:04:46.41464
16  Shelba Riach       | 20 years 3 mons 23 days 14:04:46.41464
17  Quincy Keeltagh    | 20 years 2 mons 29 days 14:04:46.41464
18  Haroun De Haven    | 19 years 8 mons 9 days 14:04:46.41464
19  Lisha Skillern     | 19 years 5 mons 22 days 14:04:46.41464
20  Fidelio Lindeboom  | 19 years 4 mons 1 day 14:04:46.41464
21  Joseph Lynn        | 19 years 3 mons 7 days 14:04:46.41464
22  Aubrie Sedgmond    | 19 years 2 mons 14:04:46.41464
23  Renell Cogle       | 1 year 24 days 14:04:46.41464
24  Paolo Libby        | 9 mons 29 days 14:04:46.41464
25  Abra Colbourn      | 8 mons 25 days 14:04:46.41464
26  Benito Celli       | 7 mons 26 days 14:04:46.41464
27  Isabeau Gameson    | 1 mon 8 days 14:04:46.41464
28  (25 rows)
```

T10. What is the current time according to Postgresql?

LANGUAGE: SQL

```sql
SELECT current_time;
```

LANGUAGE: Pseudocode

```
    current_time
--------------------
 14:05:59.733555+00
(1 row)
```

T11. What is the current data AND time according to Postgresql?

LANGUAGE: SQL

```sql
SELECT NOW();
```

LANGUAGE: Pseudocode

```
              now
------------------------------
 2023-03-02 14:06:38.621407+00
(1 row)
```

Using `dsd_22` complete the following tasks

When joining tables we can use the keywords JOIN ... ON or we can use JOIN ... USING.
We put the matching primary and foreign key in brackets. An example follows:

LANGUAGE: SQL

```sql
SELECT cust_id,
    cust_ord_id
```

```sql
3 FROM customer
4 JOIN cust_order USING (cust_id)
5 ORDER BY customer.cust_id;
```

T12. Run the following code

LANGUAGE: SQL

```sql
1 SELECT cust_id,
2     cust_ord_id,
3     manifest_id,
4     prod_name,
5     cat_name
6 FROM customer
7 JOIN cust_order USING (cust_id)
8 JOIN manifest USING (cust_ord_id)
9 JOIN product USING (prod_id)
10 JOIN category USING (cat_id)
11 ORDER BY customer.cust_id;
```

T12a. Copy the response below

LANGUAGE: Unknown

```
1 ERROR:  column "cat_id" specified in USING clause does not exist in left table
```

T12b. Fix the code so that you get a result. Look at the ERD to help clarify the issue. Rewrite the code to get a working query.

LANGUAGE: SQL

```sql
1 SELECT cust_id,
2     cust_ord_id,
3     manifest_id,
4     prod_name,
5     cat_name
6 FROM customer
7 JOIN cust_order USING (cust_id)
8 JOIN manifest USING (cust_ord_id)
9 JOIN product USING (prod_id)
10 JOIN category ON product.prod_cat=category.cat_id
11 ORDER BY customer.cust_id;
```

LANGUAGE: Pseudocode

```
1  cust_id | cust_ord_id | manifest_id |                    prod_name                     |
      ↪ cat_name
2 ---------+-------------+-------------+--------------------------------------------------+------------
      ↪
3       1 |          77 |          77 | Realigned homogeneous hub                        | Sport
4       1 |          71 |          71 | Inverse high-level attitude                      | Outdoor
5       1 |         143 |         143 | Re-engineered cohesive methodology               | Men's
      ↪ Wear
6       1 |          98 |          98 | Distributed uniform Graphic Interface            | Sport
7       1 |         146 |         146 | Fundamental global archive                       | Kid's
      ↪ Wear
8       1 |          91 |          91 | Configurable analyzing solution                  | Kid's
      ↪ Wear
9       1 |          39 |          39 | Switchable tangible product                      | Outdoor
10      1 |          68 |          68 | Streamlined asynchronous functionalities         | Sport
11      1 |          57 |          57 | Persistent demand-driven complexity              | Sport
12      1 |         131 |         131 | Seamless optimal leverage                        | Health
13      1 |          26 |          26 | Switchable tangible product                      | Outdoor
14      1 |          99 |          99 | Fundamental global archive                       | Kid's
      ↪ Wear
15 ...
16 (150 rows)
```

Page 26

LECTURE: JSON in PostgreSQL

📅 2023-03-23   🕐 13:00   🎓 Mark   📍 RB LT1

26.1   PostgreSQL and JSON

PostgreSQL can deal with JSON files. There are a number of differences in table creation & other queries which have to be observed however it is not much more complex than normal database. The ability to store and query JSON in PostgreSQL was added in 2012, which means it was added after PSQL was released so JSON processing is not as native as for another NOSQL database.

26.2   JSON

JavaScript Object Notation (JSON) is an open format standard which consists of key & value pairs. An example is shown below.

```Pseudocode
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

The user decides what the keys are and what each value is. This includes the data types of the values.
The main usage of JSON is to transfer data between servers and web applications, it could also be used to transfer data between the server and a desktop app or mobile app.
A JSON record can exist within another JSON record. See `menuitems` in the example above.

26.2.1   Main Differences Between JSON Type Data and Traditional Data

When creating a JSON data structure, we do not know the structure of each record. This is also the case in a PostgreSQL database, we do not have a known schema or know what the data types of each value will be.

26.3   Creating a Table with JSON data

A table containing JSON data still has to conform to standard rules of a PostgreSQL table. This means we have to have a primary key. This can simply be done with a ID column. An example of a simple table with an ID column and a JSON data column being created is shown below.

```sql
LANGUAGE: SQL
1 CREATE TABLE json_data(
2     id SERIAL PRIMARY KEY,
3     data JSON NOT NULL
4 );
```

26.4   Inserting data

```sql
LANGUAGE: SQL
1 INSERT INTO json_data (data) VALUES ('{"fname" : "John", "lname" : "Doe", "order" :{"Item" : "
  ↪ IPA", "QTY" : 6}}');
```

We can insert more rows using the same syntax. Note that this isn't any different from any other data-type other than the data we insert.

26.5   Reading Data from table

We can read data in the same way that we would any other table.

```sql
LANGUAGE: SQL
1 SELECT data FROM json_data;
```

```
LANGUAGE: Pseudocode
1 {"fname" : "John", "lname" : "Doe", "order" :{"Item" : "IPA", "QTY" : 6}}
2 { "customer" : "Lily Smith", "items" : {"product" : "Napkins","qty" : 24, "Colour" : "Red"}}
3 { "customer" : "Jade Davies", "items" : {"product" : "iPAD","qty" : 1}}
4 { "customer" : "Josh Green-Gardner", "items" : {"product" : "Toy Train","qty" : 2, "product" :
      ↪ "Model Station"}}
5 (4 rows)
```

This is all well and good, we have data. We can send this off to our frontend who will know how to process it. Except, we *can* get specific data from the JSON documents. PostgreSQL includes two operators to help get data from JSON documents. The --> operator returns data by key and -> returns data by text. Which one to use depends on what you are planning on doing with the data once you get it from the database.

26.5.1   Raw Values

If you want the raw values returned, you need to use the -> operator.

```sql
LANGUAGE: SQL
1 SELECT data -> 'customer'  AS customer FROM json_data;
```

```
LANGUAGE: Unknown
1 customer
2 -----------"
3
4 Lily "Smith"
5 Jade "Davies"
6 Josh Green-"Gardner
7 (4 rows)
```

Note that the row which doesn't have a `customer` key is outputted as a blank row and its existence is included in row count at the end.

26.5.2 Text version of the data

The ->> operator returns the text value of the data, which will still return an empty row where the key doesn't exist.

```sql
LANGUAGE: SQL
1 SELECT data ->> 'customer' AS customer FROM json_data;
```

```
LANGUAGE: Pseudocode
1 customer
2 -----------
3
4 Lily Smith
5 Jade Davies
6 Josh Green-Gardner
7 (4 rows)
```

26.5.3 Which To Use?

Exactly which of the operators you want to use depends on what you want to do with the data. The -> operator can give a JSON result and the ->> operator can be used to search inside it. An example can be seen below.

```sql
LANGUAGE: SQL
1 SELECT data -> 'items' ->> 'product' as product
2 FROM json_data ORDER BY product;
```

```
LANGUAGE: Pseudocode
1 product
2 ---------
3 Model Station
4 Napkins
5 iPAD
```

It is important to remember that a JSON number is not a PSQL integer. We need to cast them to integers before we can compare them.

Page 27

PRACTICAL: Better Queries

📅 2023-03-23   🕐 14:00   🎓   📍 FTC 3

T1. Write a query that puts the genre, the author's full name and the titles of the books they have written for all of the action books. The name and title must be displayed in a single column with the heading "Title and Author".

```sql
LANGUAGE: SQL
SELECT CONCAT_WS(' ', author.authorfname, author.authorlname, book.title) AS "Title and Author"
    ↪ , genre.genre FROM author
JOIN wrote ON author.authorid = wrote.authorid
JOIN book ON wrote.wroteisbn = book.isbn
JOIN bookgenre ON bookgenre.isbn = book.isbn
JOIN genre ON genre.genreid = bookgenre.genreid
WHERE genre.genre = 'Action';
```

```
LANGUAGE: Pseudocode
               Title and Author                | genre
-----------------------------------------------+--------
 Gaylord Croom OPTIONAL VALUE-ADDED OPEN SYSTEM | Action
 Odelle Cannaway RIGHT-SIZED LOCAL INTRANET     | Action
 Brendin Amberger RIGHT-SIZED LOCAL INTRANET    | Action
 Sara Hurll RIGHT-SIZED LOCAL INTRANET          | Action
 Linet Aberhart BALANCED ACTUATING INSTRUCTION SET | Action
 Bobbye Valois BALANCED ACTUATING INSTRUCTION SET  | Action
(6 rows)
```

T2. There are some common first names in the author table. Write a query that would return only a single row of data for each individual name.

```sql
LANGUAGE: SQL
SELECT DISTINCT authorfname FROM author;
```

```
LANGUAGE: Pseudocode
 authorfname
-------------
 Roxie
 Fleming
 Faina
 Linet
 Serena
 Kearney
 Collen
 Phyllis
 Sherlock
 Noach
 Zacharias
 Bobbye
 Odelle
 Sholom
 Dyana
```

```
18  Kaitlin
19  Sara
20  Shoshana
21  Malinda
22  Gerti
23  Melany
24  Bear
25  Wilmette
26  Brendin
27  Corbie
28  Gayelord
29  Clayton
30  Vera
31  (28 rows)
```

T3. How many rows of data would be returned by the query in Q2?

`28 rows`

T4. How many authors are there in total?

```sql
LANGUAGE: SQL
1 SELECT COUNT(*) FROM author;
```

```
LANGUAGE: Pseudocode
1  count
2 -------
3     30
4 (1 row)
```

T5. How many customers have a middle name?

```sql
LANGUAGE: SQL
1 SELECT COUNT(mname) FROM libuser
2 WHERE mname IS NOT NULL;
```

```
LANGUAGE: Pseudocode
1  count
2 -------
3     14
4 (1 row)
```

T6. Do we have any authors in our system that do not appear to have written any book that we have on the shelves? List their full names, surname first with a comma as a separator using concatenation, giving the column the title of "Author but no books".

```sql
LANGUAGE: SQL
1 SELECT CONCAT_WS(',', author.authorlname, author.authorfname) AS "Author but no books" FROM
      ↪ author
2 LEFT JOIN wrote ON wrote.authorid = author.authorid
3 WHERE wrote.authorid IS NULL;
```

```
LANGUAGE: Pseudocode
1  Author but no books
2 --------------------
3  Trude,Roxie
```

```
4   Burgan,Fleming
5   Youens,Malinda
6   Findlow,Wilmette
7   O'Carroll,Phyllis
8   Boxhall,Faina
9  (6 rows)
```

T7. List any library users full names and their email addresses that have never taken a book out of the library. Show another piece of data that proves that your query is giving exactly the answer you are asked for.

LANGUAGE: SQL

```sql
1 SELECT libuser.fname, libuser.emailaddress FROM libuser
2 LEFT JOIN loan ON loan.loanlibrarynumb = libuser.librarynumber
3 WHERE loan.loanlibrarynumb IS NULL;
```

LANGUAGE: Pseudocode

```
1    fname    |         emailaddress
2  -----------+----------------------------
3   Germain   | aremmers9@google.pl
4   Quincey   | fhazlea@gmpg.org
5   Julieta   | ahardisonb@deliciousdays.com
6   Gordon    | ifaradyc@usgs.gov
7   Sheelagh  | tganforthed@angelfire.com
8   Konstanze | gtongee@techcrunch.com
9   Cassie    | jdowgillf@plala.or.jp
10  Marshall  | gyeudallg@ezinearticles.com
11  Rodolfo   | zpinksh@multiply.com
12  Drake     | hnewi@cdc.gov
13  Arron     | cloukesj@ftc.gov
14  Madelina  | asinkinsk@zimbio.com
15  Elana     | jmatthewesl@springer.com
16  Zorine    | bsucrem@imgur.com
17  Stewart   | sskilln@jiathis.com
18  Gibb      | aburgino@youku.com
19 (16 rows)
```

T8. List the author's full names putting them into last name reverse alphabetical order.

LANGUAGE: SQL

```sql
1 SELECT authorfname, authorlname FROM author
2 ORDER BY authorlname DESC;
```

LANGUAGE: Pseudocode

```
1   authorfname | authorlname
2  -------------+--------------
3   Malinda     | Youens
4   Corbie      | Varga
5   Bobbye      | Valois
6   Roxie       | Trude
7   Gerti       | Shirtcliffe
8   Zacharias   | Ransley
9   Bear        | Oliphand
10  ...
11 (30 rows)
```

T9. List the author's full names whose first name begins with the letter B. Sort the results into the same order as in T8.

```
LANGUAGE: SQL

1  SELECT authorfname, authorlname FROM author
2  WHERE SUBSTRING(authorfname, 1, 1) = 'B'
3  ORDER BY authorlname DESC;
```

```
LANGUAGE: Pseudocode

1   authorfname | authorlname
2  -------------+-------------
3   Bobbye      | Valois
4   Bear        | Oliphand
5   Brendin     | Amberger
6  (3 rows)
```

T10. List the books that have a genre. Sort the books into alphabetical genre order.

```
LANGUAGE: SQL

1  SELECT book.title, genre.genre FROM book
2  LEFT JOIN bookgenre ON bookgenre.isbn = book.isbn
3  JOIN genre ON bookgenre.genreid = genre.genreid
4  WHERE bookgenre.isbn IS NOT NULL
5  ORDER BY book.title ASC;
```

```
LANGUAGE: Pseudocode

1                       title                      |   genre
2  -------------------------------------------------+-------------
3   BALANCED ACTUATING INSTRUCTION SET              | Action
4   DE-ENGINEERED ZERO TOLERANCE GRAPHIC INTERFACE  | Comedy
5   DEVOLVED EXUDING APPROACH                       | Thriller
6   FRONT-LINE EVEN-KEELED APPROACH                 | Sci-Fi
7   FULLY-CONFIGURABLE OPTIMAL FUNCTION             | Romance
8   FUNDAMENTAL BIFURCATED ARTIFICIAL INTELLIGENCE  | Romance
9   IMPLEMENTED INTERMEDIATE METHODOLOGY            | Horror
10  MONITORED MULTI-STATE CONGLOMERATION            | Music
11  MULTI-TIERED INTERACTIVE HELP-DESK              | Non-Fiction
12  MULTI-TIERED RESPONSIVE UTILISATION             | Thriller
13  OPTIMIZED BANDWIDTH-MONITORED FIRMWARE          | Noir
14  OPTIONAL VALUE-ADDED OPEN SYSTEM                | Action
15  PROFIT-FOCUSED OBJECT-ORIENTED METHODOLOGY      | Anime
16  PROGRAMMABLE CLEAR-THINKING PORTAL              | Non-Fiction
17  RE-ENGINEERED SYSTEM-WORTHY CORE                | Comedy
18  REDUCED COMPOSITE OPEN ARCHITECTURE             | Non-Fiction
19  RIGHT-SIZED LOCAL INTRANET                      | Action
20  SECURED 24/7 PRODUCT                            | Crime
21  SECURED MOTIVATING CONCEPT                      | Adult
22  VIRTUAL NEUTRAL CAPACITY                        | Action
23  (20 rows)
```

T11. Now sort the output for Q10 into descending `genreid`.

```
LANGUAGE: SQL

1  SELECT book.title, genre.genre FROM book
2  LEFT JOIN bookgenre ON bookgenre.isbn = book.isbn
3  JOIN genre ON bookgenre.genreid = genre.genreid
4  WHERE bookgenre.isbn IS NOT NULL
5  ORDER BY book.title DESC;
```

```
LANGUAGE: Pseudocode

1                       title                      |   genre
2  -------------------------------------------------+-------------
```

```
 3  VIRTUAL NEUTRAL CAPACITY                       | Action
 4  SECURED MOTIVATING CONCEPT                     | Adult
 5  SECURED 24/7 PRODUCT                           | Crime
 6  RIGHT-SIZED LOCAL INTRANET                     | Action
 7  REDUCED COMPOSITE OPEN ARCHITECTURE            | Non-Fiction
 8  RE-ENGINEERED SYSTEM-WORTHY CORE               | Comedy
 9  PROGRAMMABLE CLEAR-THINKING PORTAL             | Non-Fiction
10  PROFIT-FOCUSED OBJECT-ORIENTED METHODOLOGY     | Anime
11  OPTIONAL VALUE-ADDED OPEN SYSTEM               | Action
12  OPTIMIZED BANDWIDTH-MONITORED FIRMWARE         | Noir
13  MULTI-TIERED RESPONSIVE UTILISATION            | Thriller
14  MULTI-TIERED INTERACTIVE HELP-DESK             | Non-Fiction
15  MONITORED MULTI-STATE CONGLOMERATION           | Music
16  IMPLEMENTED INTERMEDIATE METHODOLOGY           | Horror
17  FUNDAMENTAL BIFURCATED ARTIFICIAL INTELLIGENCE | Romance
18  FULLY-CONFIGURABLE OPTIMAL FUNCTION            | Romance
19  FRONT-LINE EVEN-KEELED APPROACH                | Sci-Fi
20  DEVOLVED EXUDING APPROACH                      | Thriller
21  DE-ENGINEERED ZERO TOLERANCE GRAPHIC INTERFACE | Comedy
22  BALANCED ACTUATING INSTRUCTION SET             | Action
23 (20 rows)
```

T12. Sort the output from Q10 into alphabetical genre order and alphabetical reversed title order in a single query.

LANGUAGE: SQL

```sql
1 SELECT book.title, genre.genre FROM book
2 LEFT JOIN bookgenre ON bookgenre.isbn = book.isbn
3 JOIN genre ON bookgenre.genreid = genre.genreid
4 WHERE bookgenre.isbn IS NOT NULL
5 ORDER BY genre.genre ASC, book.title DESC;
```

LANGUAGE: Pseudocode

```
 1                     title                      |    genre
 2 ------------------------------------------------+-------------
 3  VIRTUAL NEUTRAL CAPACITY                       | Action
 4  RIGHT-SIZED LOCAL INTRANET                     | Action
 5  OPTIONAL VALUE-ADDED OPEN SYSTEM               | Action
 6  BALANCED ACTUATING INSTRUCTION SET             | Action
 7  SECURED MOTIVATING CONCEPT                     | Adult
 8  PROFIT-FOCUSED OBJECT-ORIENTED METHODOLOGY     | Anime
 9  RE-ENGINEERED SYSTEM-WORTHY CORE               | Comedy
10  DE-ENGINEERED ZERO TOLERANCE GRAPHIC INTERFACE | Comedy
11  SECURED 24/7 PRODUCT                           | Crime
12  IMPLEMENTED INTERMEDIATE METHODOLOGY           | Horror
13  MONITORED MULTI-STATE CONGLOMERATION           | Music
14  OPTIMIZED BANDWIDTH-MONITORED FIRMWARE         | Noir
15  REDUCED COMPOSITE OPEN ARCHITECTURE            | Non-Fiction
16  PROGRAMMABLE CLEAR-THINKING PORTAL             | Non-Fiction
17  MULTI-TIERED INTERACTIVE HELP-DESK             | Non-Fiction
18  FUNDAMENTAL BIFURCATED ARTIFICIAL INTELLIGENCE | Romance
19  FULLY-CONFIGURABLE OPTIMAL FUNCTION            | Romance
20  FRONT-LINE EVEN-KEELED APPROACH                | Sci-Fi
21  MULTI-TIERED RESPONSIVE UTILISATION            | Thriller
22  DEVOLVED EXUDING APPROACH                      | Thriller
23 (20 rows)
```

T13. List all of the genres that the author Corbie Varga have written.

LANGUAGE: SQL

```sql
1 SELECT genre.genre FROM genre
2 JOIN bookgenre ON bookgenre.genreid = genre.genreid
3 JOIN book ON book.isbn = bookgenre.isbn
4 JOIN wrote ON wroteisbn = book.isbn
5 JOIN author ON author.authorid = wrote.authorid
6 WHERE author.authorfname = 'Corbie' AND author.authorlname = 'Varga';
```

```
LANGUAGE: Pseudocode
1     genre
2 ------------
3  Noir
4  Adult
5  Non-Fiction
6  Comedy
7  Anime
8 (5 rows)
```

T14. Write a view that will allow a user to list the user's first and last names, the book titles, the author last names of all books that were loaned out between Feb 28th 2022 and Oct 30th 2022. Call the view loans.

```sql
LANGUAGE: SQL
1 CREATE VIEW loans AS
2 SELECT lu.fname, lu.lname, book.title, author.authorlname FROM libuser lu
3 JOIN loan ON loan.loanlibrarynumb = lu.librarynumber
4 JOIN book ON loan.isbn = book.isbn
5 JOIN wrote ON book.isbn = wrote.wroteisbn
6 JOIN author on author.authorid = wrote.authorid
7 WHERE loan.loanstart BETWEEN '2022-02-28' AND '2022-10-30';
```

T15. Create a new role in your DBMS called libadmin and give them the ability to login and set a password for this new role.

```sql
LANGUAGE: SQL
1 CREATE ROLE libadmin WITH LOGIN PASSWORD 'imasecurepasswordipromise';
2 CREATE DATABASE libadmin OWNER libadmin;
```

T16. Give the new admin user access to the view created in Q14.

```sql
LANGUAGE: SQL
1 GRANT select
2 ON loans
3 TO libadmin;
```

T17. For this new role, run the view from Q14 and copy the output below.

```sql
LANGUAGE: SQL
1 SELECT * FROM loans;
```

```
LANGUAGE: Pseudocode
1    fname   |     lname     |                       title                        |  authorlname
2 ----------+---------------+----------------------------------------------------+---------------
3  Olympia  | Chasle        | SECURED MOTIVATING CONCEPT                          | Varga
4  Rossy    | Studd         | SECURED MOTIVATING CONCEPT                          | Varga
5  Ethel    | Calcott       | FRONT-LINE EVEN-KEELED APPROACH                     | Amberger
6  Baryram  | Postlethwaite | DEVOLVED EXUDING APPROACH                           | Aslett
7  Rossy    | Studd         | DEVOLVED EXUDING APPROACH                           | Aslett
8  Olympia  | Chasle        | SECURED MOTIVATING CONCEPT                          | Harley
9  Rossy    | Studd         | SECURED MOTIVATING CONCEPT                          | Harley
10  Olympia  | Chasle        | FUNDAMENTAL BIFURCATED ARTIFICIAL INTELLIGENCE     | Bing
11 ...
12 (53 rows)
```

Page 28

PRACTICAL: SQL Summary

📅 2023-03-30   🕐 14:00   🎓 Mark   📍 FTC 3

T1. Connect to `lib22` database

```
LANGUAGE: SQL
1  \c lib22
```

T2. Run the following code to enter a new row of data then run through the tutor tasks above

```
LANGUAGE: SQL
1  INSERT INTO LIBUSER VALUES ('AAA87857654', 'Lesya', NULL, 'Harrison', 'bharrison66@gov.uk', '1
    ↪ The Avenue', 'Fratton', 'PO99 5GG');
```

```
LANGUAGE: Pseudocode
1  lib22=# Select count(fname) from libuser;
2   count
3  -------
4      26
5  (1 row)
6
7  lib22=# select fname, count(fname) from libuser GROUP BY fname;
8    fname    | count
9  -----------+-------
10  Arron     |     1
11  Millard   |     1
12  Sheelagh  |     1
13  Ethel     |     1
14  Drake     |     1
15  Julieta   |     1
16  Anastasia |     1
17  Quincey   |     1
18  Lesya     |     2
19  Gibb      |     1
20  Madelina  |     1
21  Rossy     |     1
22  Stewart   |     1
23  Zorine    |     1
24  Elana     |     1
25  Cassie    |     1
26  Baryram   |     1
27  Gordon    |     1
28  Emmeline  |     1
29  Konstanze |     1
30  Rodolfo   |     1
31  Marshall  |     1
32  Fernanda  |     1
33  Germain   |     1
34  Olympia   |     1
35  (25 rows)
36
37  lib22=# select fname, count(fname) from libuser group by fname order by fname;
38    fname    | count
```

```
39  -----------+-------
40  Anastasia |    1
41  Arron     |    1
42  Baryram   |    1
43  Cassie    |    1
44  Drake     |    1
45  Elana     |    1
46  Emmeline  |    1
47  Ethel     |    1
48  Fernanda  |    1
49  Germain   |    1
50  Gibb      |    1
51  Gordon    |    1
52  Julieta   |    1
53  Konstanze |    1
54  Lesya     |    2
55  Madelina  |    1
56  Marshall  |    1
57  Millard   |    1
58  Olympia   |    1
59  Quincey   |    1
60  Rodolfo   |    1
61  Rossy     |    1
62  Sheelagh  |    1
63  Stewart   |    1
64  Zorine    |    1
65  (25 rows)
66
67  lib22=# select fname, count(fname) as number_of_names from libuser group by fname order by
        ↪ count(fname) desc;
68    fname   | number_of_names
69  -----------+-----------------
70  Lesya     |               2
71  Millard   |               1
72  Sheelagh  |               1
73  Ethel     |               1
74  Drake     |               1
75  Julieta   |               1
76  Anastasia |               1
77  Quincey   |               1
78  Gibb      |               1
79  Madelina  |               1
80  Rossy     |               1
81  Stewart   |               1
82  Zorine    |               1
83  Elana     |               1
84  Cassie    |               1
85  Baryram   |               1
86  Gordon    |               1
87  Emmeline  |               1
88  Konstanze |               1
89  Rodolfo   |               1
90  Marshall  |               1
91  Fernanda  |               1
92  Germain   |               1
93  Arron     |               1
94  Olympia   |               1
95  (25 rows)
96
97  lib22=# select fname, count(fname) from libuser group by fname where fname = 'Lesya';
98  ERROR:  syntax error at or near "where"
99  LINE 1: ...t fname, count(fname) from libuser group by fname where fnam...
100                                                                    ^
101 lib22=# select fname, count(fname) from libuser group by fname having fname = 'Lesya';
102  fname | count
103 -------+-------
104  Lesya |     2
105 (1 row)
106
107 lib22=# select fname,mname, lname, count(fname) from libuser group by fname, mname,lname having
        ↪  fname = 'Lesya';
108  fname | mname  |    lname     | count
109 -------+--------+-------------+-------
110  Lesya |        | Harrison    |     1
111  Lesya | Bidget | Shackleford |     1
```

```
12 (2 rows)
```

T3. Draw an ERD, (entity with primary and foreign keys only, attributes NOT needed), to cover the following business rules

- A car can be owned by 1 person

- 1 person may own 1 or more cars

- A car may have many services

- Each service is for a single car
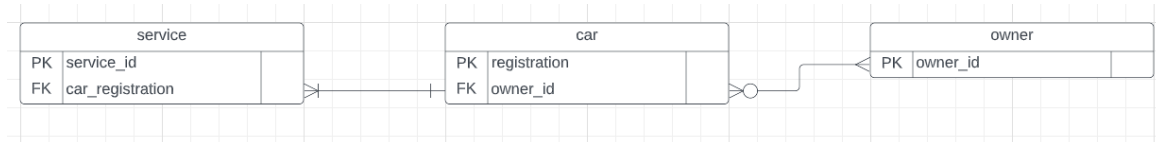
T4. Using `lib22`, how many loans have been made by the library?

```
LANGUAGE: SQL
1 SELECT COUNT(*) FROM LOAN;
```

```
LANGUAGE: Pseudocode
1  count
2 -------
3     30
4 (1 row)
```

T5. How many individual books have been lent out? (Think about the individual ISBNs in the loan table).

```
LANGUAGE: SQL
1 SELECT COUNT(DISTINCT isbn)FROM loan;
```

```
LANGUAGE: Pseudocode
1  count
2 -------
3     12
4 (1 row)
```

T6. What is the latest date that we have data for in the loan table?

```
LANGUAGE: SQL
1 SELECT loanstart FROM loan ORDER BY loanstart DESC LIMIT 1;
```

```
LANGUAGE: Pseudocode
1  loanstart
2 ------------
3  2022-11-27
4 (1 row)
```

T7. How many books were loaned on the date from Q.6?

```
LANGUAGE: SQL
1  SELECT COUNT(isbn) FROM loan
2  WHERE loanstart = '2022-11-27';
```

```
LANGUAGE: Pseudocode
1   count
2  -------
3       1
4  (1 row)
```

T8. List the book titles that were loaned between 4th October 2022 and 10th October 2022 (inclusive).

```
LANGUAGE: SQL
1  SELECT title FROM book
2  JOIN loan ON book.isbn = loan.isbn
3  where loanstart BETWEEN '2022-10-04' AND '2022-10-10';
```

```
LANGUAGE: Pseudocode
1               title
2  -----------------------------------
3   FULLY-CONFIGURABLE OPTIMAL FUNCTION
4  (1 row)
```

T9. How many books were loaned out between the dates in Q.8? Write a query, don't just count how many results you see.

```
LANGUAGE: SQL
1  SELECT COUNT(isbn) FROM loan
2  where loanstart BETWEEN '2022-10-04' AND '2022-10-10';
```

```
LANGUAGE: Pseudocode
1   count
2  -------
3       1
4  (1 row)
```

T10. Who wrote the book De-Engineered Zero Tolerance Graphic Interface?

```
LANGUAGE: SQL
1  SELECT CONCAT_WS(' ', authorfname, authorlname) FROM author
2  JOIN wrote ON author.authorid = wrote.authorid
3  JOIN book on wroteisbn = isbn
4  WHERE UPPER(title) = UPPER('De-Engineered Zero Tolerance Graphic Interface');
```

```
LANGUAGE: Pseudocode
1    concat_ws
2  ---------------
3   Corbie Varga
```

```
4  Sara Hurll
5  Linet Aberhart
6 (3 rows)
```

T11. How many times has the book in Q.10 been loaned out of the library?

LANGUAGE: SQL

```sql
1 SELECT count(loan.isbn) FROM loan
2 JOIN book ON book.isbn = loan.isbn
3 WHERE UPPER(book.title) = UPPER('De-Engineered Zero Tolerance Graphic Interface');
```

LANGUAGE: Pseudocode

```
1  count
2 -------
3      0
4 (1 row)
```

T12. List all users who have NOT loaned books out of the library. (It is up to you what data you need to display)

LANGUAGE: SQL

```sql
1 SELECT fname, lname FROM libuser
2 FULL OUTER JOIN loan ON loan.loanlibrarynumb = libuser.librarynumber
3 WHERE loanlibrarynumb IS NULL;
```

LANGUAGE: Pseudocode

```
1     fname   |    lname
2 -----------+-----------
3  Germain   | Remmers
4  Konstanze | Tonge
5  Gibb      | Burgin
6  Arron     | Loukes
7  Drake     | New
8  Cassie    | Dowgill
9  Quincey   | Hazle
10  Marshall  | Yeudall
11  Stewart   | Skill
12  Zorine    | Sucre
13  Elana     | Matthewes
14  Julieta   | Hardison
15  Lesya     | Harrison
16  Gordon    | Farady
17  Madelina  | Sinkins
18  Sheelagh  | Ganforthe
19  Rodolfo   | Pinks
20 (17 rows)
```

T13. Which keyword forces an attribute to only have one version of a value in a table?

LANGUAGE: SQL

```sql
1 UNIQUE
```

T14. Change the following code to enforce the behaviour in Q.13 on the email attribute.

LANGUAGE: SQL

```sql
1 create table test_table (
2     test_id int primary key,
```

```
3      fname varchar(30) not null,
4      name varchar(30),
5      lname varchar(50) not null,
6      email varchar(70) UNIQUE not null
7 );
```

T15. Using the following attribute names, constraints and datatypes, create a table that connects to the table in Q.14. Call this table `test_table2`

- `test_id2` - int primary key

- `linking_att` - int foreign key

- `notes` - text

LANGUAGE: SQL
```
1 CREATE TABLE test_table2(
2      test_id2 INT PRIMARY KEY,
3      linking_att INT REFERENCES test_table(test_id),
4      notes text
5 );
```

Page 29

PRACTICAL: Foreign Keys & Joins Practice

📅 2023-05-04   🕐 14:00   🎓 Val   📍 FTC 3

Create a new database called hobbies

```
LANGUAGE: SQL
1 CREATE DATABASE hobbies;
```

create a new table called game:

```
LANGUAGE: SQL
1 CREATE TABLE IF NOT EXISTS game (
2   id      VARCHAR(10)  PRIMARY KEY,
3   vendor  INT          NOT NULL,
4   name    VARCHAR(20)      NOT NULL,
5   price   DECIMAL(5,2) NOT NULL
6 );
```

insert 3 records into the game table:

```
LANGUAGE: SQL
1 INSERT INTO game (id, vendor, name, price)
2     VALUES ('371/2209', 1, 'Scrabble', 14.50);
3 INSERT INTO game (id, vendor, name, price)
4     VALUES ('373/2296', 2, 'Jenga', 6.99);
5 INSERT INTO game (id, vendor, name, price)
6     VALUES ('303/1199', 22, 'D&D', 26.99);
```

add three more rows of data that match the following:

```
Id = 360/9659
Vendor = 1
Name = Uno
Price = 11.99


Id = 373/5372
Vendor = 3
Name = Connect
Price = 5.99


Id = 370/9470
Vendor = 3
Name = Bingo
Price = 8.99
```

LANGUAGE: SQL

```sql
INSERT INTO game (id, vendor, name, price)
VALUES ('360/9659', 1, 'Uno', 11.99);

INSERT INTO game(id, vendor, name, price)
VALUES('373/5372', 3, 'Connect', 5.99);

INSERT INTO game (id, vendor, name, price)
VALUES('370/9470', 3, 'Bingo', 8.99);
```

create a table called `vendor`

LANGUAGE: SQL

```sql
CREATE TABLE IF NOT EXISTS vendor (
  id       INT      PRIMARY KEY,
  name     VARCHAR(20) NOT NULL,
  location VARCHAR(20) NOT NULL
);
```

insert 4 records into the `vendor` table

LANGUAGE: SQL

```sql
INSERT INTO vendor (id, name, location)
   VALUES (1, 'Mattel Inc', 'El Segundo, Ca, USA'),
    (2, 'Hasbro Inc', 'Pawtucket, RI, USA'),
    (3, 'J.W.Spear Plc', 'Enfield, Middx, UK'),
    (4, 'Hornby', 'Margate, Kent, UK');
```

We need data from both tables to get all the information about who sells each game. Run the following SQL and look at the output:

LANGUAGE: SQL

```sql
SELECT game.id     AS Product_Code,
       game.name   AS Game,
       vendor.name AS Vendor,
       game.price  AS Price
FROM game, vendor;
```

LANGUAGE: Pseudocode

```
product_code |   game   |    vendor     | price
-------------+----------+---------------+-------
371/2209     | Scrabble | Mattel Inc    | 14.50
371/2209     | Scrabble | Hasbro Inc    | 14.50
371/2209     | Scrabble | J.W.Spear Plc | 14.50
371/2209     | Scrabble | Hornby        | 14.50
373/2296     | Jenga    | Mattel Inc    |  6.99
373/2296     | Jenga    | Hasbro Inc    |  6.99
373/2296     | Jenga    | J.W.Spear Plc |  6.99
373/2296     | Jenga    | Hornby        |  6.99
303/1199     | D&D      | Mattel Inc    | 26.99
303/1199     | D&D      | Hasbro Inc    | 26.99
303/1199     | D&D      | J.W.Spear Plc | 26.99
303/1199     | D&D      | Hornby        | 26.99
360/9659     | Uno      | Mattel Inc    | 11.99
360/9659     | Uno      | Hasbro Inc    | 11.99
360/9659     | Uno      | J.W.Spear Plc | 11.99
360/9659     | Uno      | Hornby        | 11.99
373/5372     | Connect  | Mattel Inc    |  5.99
373/5372     | Connect  | Hasbro Inc    |  5.99
373/5372     | Connect  | J.W.Spear Plc |  5.99
373/5372     | Connect  | Hornby        |  5.99
370/9470     | Bingo    | Mattel Inc    |  8.99
370/9470     | Bingo    | Hasbro Inc    |  8.99
```

```
25  370/9470      | Bingo     | J.W.Spear Plc |  8.99
26  370/9470      | Bingo     | Hornby        |  8.99
```

Look at the vendor for each game. What is wrong with the resulting data from this query? Who makes each game? *every game is made by every vendor - this has produced a cartesian product.*

What is the keyword AS doing to the output? *giving the columns nice human readable names*
What has happened to the case of the words after the AS keyword? *it is lost as the table alias is not in ""*

If you join tables together you MUST have a WHERE or JOIN Clause! Now run the SQL again with the where clause added and look at the result.

LANGUAGE: SQL

```sql
SELECT game.id     AS "Product Code",
       game.name   AS "Game",
       vendor.name AS "Vendor",
       game.price  AS Price
FROM game, vendor
WHERE  vendor.id = game.vendor;
```

LANGUAGE: Pseudocode

```
Product Code |   Game    |     Vendor      | price
-------------+-----------+-----------------+-------
 371/2209    | Scrabble  | Mattel Inc      | 14.50
 373/2296    | Jenga     | Hasbro Inc      |  6.99
 360/9659    | Uno       | Mattel Inc      | 11.99
 373/5372    | Connect   | J.W.Spear Plc   |  5.99
 370/9470    | Bingo     | J.W.Spear Plc   |  8.99
(5 rows)
```

What has happened to the case of the words after the AS keyword? *it is maintained as the alias is now in speech marks*
Where is the D&D game? Why does this not appear in this printout? *the vendor does not exist, as we have not created a relationship between the two tables (ie a foreign key) the database doesn't know we want it.*
This should now be showing you data that reflects reality. Notice that this works without the foreign key being created.
In preparation for the next step, delete these sample tables:

LANGUAGE: SQL

```sql
DROP TABLE game;
DROP TABLE vendor;
```

Now try and re-create the tables with a foreign key constraint in the way shown below.
It should fail, we want you to work out why this is happening and how you can sort it out.
This is usually the biggest problem students face when trying to create their own tables and data!

LANGUAGE: SQL

```sql
CREATE TABLE game (
  id     VARCHAR(10)  PRIMARY KEY,
  vendor INT          NOT NULL REFERENCES vendor(id),
```

```sql
4   name    CHAR(20)     NOT NULL,
5   price   DECIMAL(6,2) NOT NULL
6 );
```

Why does this fail? *as the table vendor which its trying to reference doesn't exist*
Run the following code

```sql
LANGUAGE: SQL
1 CREATE TABLE vendor (
2   id       INT       PRIMARY KEY,
3   name     CHAR(20) NOT NULL,
4   location CHAR(20) NOT NULL
5 );
6
7 INSERT INTO vendor (id, name, location)
8    VALUES (1, 'Mattel Inc', 'El Segundo, Ca, USA'),
9     (2, 'Hasbro Inc', 'Pawtucket, RI, USA'),
10    (3, 'J.W.Spear Plc', 'Enfield, Middx, UK'),
11    (4, 'Hornby', 'Margate, Kent, UK');
12
13 INSERT INTO game (id, vendor, name, price)
14 VALUES ('371/2209', 1, 'Scrabble', 14.50), ('373/2296', 2, 'Jenga', 6.99),('360/9659', 1, 'Uno'
    ↪ , 11.99), ('373/5372', 3, 'Connect', 5.99), ('370/9470', 3, 'Bingo', 8.99), ('303/1199',
    ↪  22, 'D&D', 26.99);
```

What does the system response mean?
What happens when you try to insert the games?
When trying to insert into the game table without creating the table first, the database complains as it doesn't know where to insert the data. To fix this, the game table would need to be created first

```sql
LANGUAGE: SQL
1 CREATE TABLE game (
2   id      VARCHAR(10)  PRIMARY KEY,
3   vendor  INT          NOT NULL REFERENCES vendor(id),
4   name    VARCHAR(20)  NOT NULL,
5   price   DECIMAL(5,2) NOT NULL
6 );
7
8 INSERT INTO game (id, vendor, name, price)
9 VALUES ('371/2209', 1, 'Scrabble', 14.50), ('373/2296', 2, 'Jenga', 6.99),('360/9659', 1, 'Uno'
    ↪ , 11.99), ('373/5372', 3, 'Connect', 5.99), ('370/9470', 3, 'Bingo', 8.99), ('303/1199',
    ↪  22, 'D&D', 26.99);
```

Display the data stored in the game table. What has gone wrong?

```
LANGUAGE: Pseudocode
1 hobbies=# select * from game;
2  id | vendor | name | price
3 ----+--------+------+-------
4 (0 rows)
```

No data has been inserted as there was a vendor value which isn't present in the vendor table.
Now run this code

```sql
LANGUAGE: SQL
1 INSERT INTO game (id, vendor, name, price)
2 VALUES ('371/2209', 1, 'Scrabble', 14.50), ('373/2296', 2, 'Jenga', 6.99),('360/9659', 1, 'Uno'
    ↪ , 11.99), ('373/5372', 3, 'Connect', 5.99), ('370/9470', 3, 'Bingo', 8.99), ('303/1199',
    ↪  2, 'D&D', 26.99);
```

When you have the tables and data inserted run the queries again:

```sql
SELECT  game.id      AS ProductCode,
        game.name    AS Game,
        vendor.name  AS Vendor,
        game.price   AS Price
FROM game, vendor;
```

How many records are displayed in each query now? *24*
How many records are returned for:
JW Spear? *6*
Hornby? *6*
Why? *we have generated a cartesian product as we haven't joined the tables correctly*

Now run the following code:

```sql
SELECT  game.id      AS "Product Code",
        game.name    AS "Game",
        vendor.name  AS "Vendor",
        game.price   AS "Price"
FROM game, vendor
WHERE  vendor.id = game.vendor;
```

How many records are displayed in each query now? *6*
How many records are returned for:
JW Spear? *2*
Hornby? *0*

Add new games into the game table that match the following details. Take a note of the responses you get when entering the data:

```
Id = 360/9659
Vendor = 1
Name = Risk
Price = 15.99
```

```
Id = 361/9688
Vendor = 10
Name = Monopoly
Price = 19.99
```

```
Id = 366/6661
Vendor = 2
Name = Goal!
Price = 1121.99
```

```sql
INSERT INTO game(id, vendor, name, price)
VALUES('360/9659', 1, 'Risk', 15.99);
```

```
ERROR:  duplicate key value violates unique constraint "game_pkey"
```

```
2  DETAIL:  Key (id)=(360/9659) already exists.
```

LANGUAGE: SQL

```sql
1  INSERT INTO game(id, vendor, name, price)
2  VALUES('361/9688', 10, 'Monopoly', 19.99);
```

LANGUAGE: Pseudocode

```
1  ERROR:  insert or update on table "game" violates foreign key constraint "game_vendor_fkey"
2  DETAIL:  Key (vendor)=(10) is not present in table "vendor".
```

LANGUAGE: SQL

```sql
1  INSERT INTO game(id, vendor, name, price)
2  VALUES('266/6661', 2, 'Goal!', 1121.99);
```

LANGUAGE: Pseudocode

```
1  ERROR:  numeric field overflow
2  DETAIL:  A field with precision 5, scale 2 must round to an absolute value less than 10^3.
```