# Basics

- SQL ISN'T a database, it's a way of communicating with relational databases. It stands for Structured Query Language.
  - When you see databases with SQL in their name, its just to indicate that SQL is the primary language to be used for interacting with that database.
- A relational database is a type of Database Management System (DBMS) that stores + organises data in a structured manner + where the relationship within the data is maintained (allowing columns to be sorted + filtered etc without any of the other data messing up due to not changing w/ that filter.)
- Although other types of DBs exist, relational DBs are by far the most widely used.
- Although Excel may seem like a relational database, its not, its a tabulation software. Although it allows things like filtering, it differs from a relational DB in that it doesn't have predefined structures + relationships between tables (ie Excel is more flexible), it doesn't enforce data integrity through use of primary + secondary keys, it doesn't allow for complex queries (but rather more simple ones), and usually it is used only by 1 person at a time, whereas relational databases can be multi-user.
- SQL is good for:
  - Quickly + cleanly combining datasets
  - Automating steps for re-use
  - Data integrity (can use the data, but can't edit 'cell' of raw base data).
- A database is made up of many tables. (Excel tabs = DB tables)

GUY HOTH

Lesson 6:

- To import a database, start PGAdmin 4, start a new server (using 'Local' as the Hostname/address in the 'Connection' tab).
  - ↳ Then right click the new server > Create > Database
  - ↳ Then right click the new database > Restore
    - ↳ Browse to file, and in 'Data/Objects' tab activate all 3 options under 'Sections' (pre-data, data, post-data).
    - ↳ Then hit the 'restore' button.
- To run a Query, right click database > Query Tool.
  - ↳ After you run a Query the Query output gets printed at the bottom under 'Data Output' → you can save these results as a CSV file so you can share the Query results w/ others.

GUNY HOTH

# Lesson 12 : Syntax

SQL keywords are usually expressed in all CAPS. Will still work in lowercase, but makes it much easier to read + analyse the commands

- In the SQL database, you can view all the tables contained within it, and all the columns comprising each table.
  - ↳ Expand the database in left panel, then expand Schemas, then expand public, then expand tables.
    - ↳ expanding each table will give you the info contained within it.

- SELECT : most commonly used statement. Allows us to retrieve data from the database. SELTECT first_name FROM table
  - ↳ It is the thing returned by running the Query

- DISINCT : returns all the unique items (ie no duplicates) from a column. SELECT DISTINCT (column) FROM table
  - ↳ field can be in brackets if you want to to help with readability.

- COUNT : returns the number of input rows that match the condition of our query. As COUNT is a function, the parameters need to be in brackets. SELECT COUNT (DISTINCT names) FROM table
  - → NOTE : In this syntax, the SELECT statement is being used to retrieve the result of the Count function., it's not being used to retrieve the 'names' column for use in the function. This is be SELECT is responsible for returning the final result.
  - → NOTE 2 : It simply returns the number of rows in the result table, so it doesn't matter if you select a particular column or the entire table (*). However, good practice to use a column name as it helps trace your original logic/ reasoning back for doing the function for future reference.

GUYHOTH

- WHERE : Second most commonly used statement after SELECT. The two go hand in hand and are used almost everywhere. WHERE is the condition on the column for the row to be returned.

  e.g.
  ```
  SELECT column1, column2
  FROM table1
  WHERE conditions
  ```
  NOTE: WHERE should always follow immediately after the FROM clause.

  [Think of the WHERE as a filter]
  ↳ e.g WHERE CA = MAO1

  - Various comparison operators, e.g =, <>, <=, <, >=, >,
  - Comparison operators can be combined (AND, OR, NOT).
  - When using '=' operator, capitalisation matters, e.g dave ≠ Dave

- ORDER : Sort rows based on their column value, either in ascending or descending order.

  ```
  SELECT column1, column2
  FROM table1
  ORDER BY column2 ASC/DESC
  ```
  ↳ ORDER should always come @ end of statement, regardless of how long the statement is.

  - ORDER BY company, Sales will order by the company first, then the Sales, ie companies are sorted first, and then Sales are sorted within each company.
  - NOTE 2: you can still order by a column even if that column isn't SELECTED in your Query.

- LIMIT : Limits how many rows are returned, e.g LIMIT 5 shows the first 5 rows.
  ↳ LIMIT Integer goes at the very end of a Query, even bellow ORDER BY.
  - Useful for Questions like 'What is the top 10 earthwork movements by Road'.

GUY HOTH

- BETWEEN: match a value falling within a range.

  Essentially the same as variable >= x AND variable <= y

  BETWEEN high AND low

  Can pair with NOT for NOT BETWEEN, but note that BETWEEN is inclusive whereas NOT BETWEEN is exclusive. For example, if the range is 4 to 8, 8 would be found as a BETWEEN, but not a NOT BETWEEN.

  - Can also be used for dates ⇒ have to be formatted in the 'ISO 8601' format (YYYY-MM-DD).

    ↳ date BETWEEN '2020-06-30' AND '2023-06-30'

    NOTE that for the end date, if timestamps are also included in the column data, the range would only go up to the start of that day (ie when clock turns 00:00), not to the end of that day (ie when clock turns 23:59). Therefore always be careful with timestamps.

- IN: checks if a value is in a list. Essentially prevents you from having to write statements with many OR parameters.

  WHERE colour IN ('red', 'blue', 'green')

- LIKE and ILIKE: matches general patterns in strings (e.g. regex), eg if we wanted to extract all emails from a column whose domain ended in @gmail.com.

  2 wildcard types ⇒ % matches any sequence of characters

  ⇒ _ matches any single character.

  ↳ can use more than one underscore at a time

  NOTE: LIKE is case-sensitive, ILIKE is not case-sensitive.

GUYHOTH

## AGGREGATES

- Aggregates allow you to aggregate a range of data, for example, SUM, AVERAGE, MAX, MIN (range).

## GROUPING

- Group performs the query on each distinct group of data. for example,

| Category | Sales |
|---|---|
| A | 10 |
| A | 5 |
| B | 3 |
| C | 15 |
| B | 20 |
| A | 5 |

SUM (Sales)
GROUP BY category

| Category | Sales |
|---|---|
| A | 20 |
| B | 23 |
| C | 15 |

- 80/20 Rule → The specific syntax around Group by is quite involved as theres some rules that need to be followed. I don't really care how to do it, more so what can be done. Eg. get the average haul distance for each community area.

## HAVING

- HAVING allows us to filter groups after an aggregate function has been called. For example, continuing from the above, we can say:

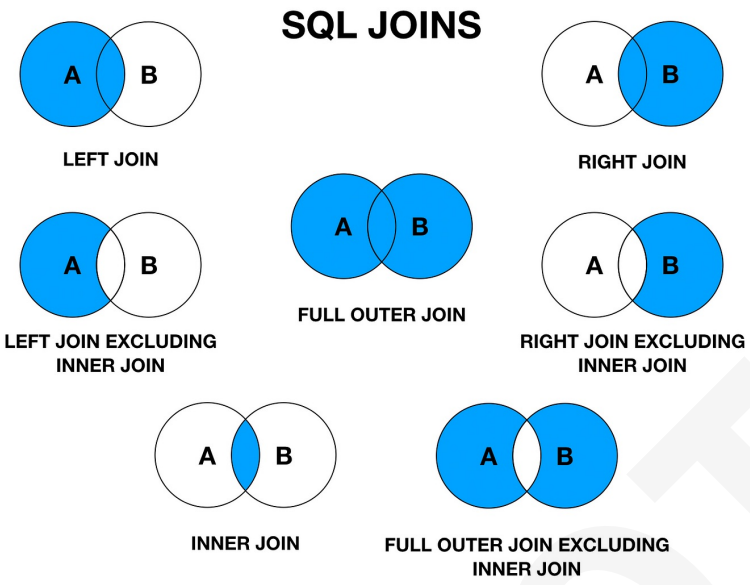
HAVING SUM (Sales) >=20 the table becomes:

| Category | Sales |
|---|---|
| A | 20 |
| B | 23 |

This is useful for when you're interested in certain things, e.g Show the average haulage distance for each community area where the averages are over 20km.

# JOINS

- Joins allow us to combine data from different tables.




- The reason different types of Joins exist is so the user can select how they want to handle data that's missing from one of the tables.

- AS ⇒ allows us to create an alias for a column or table etc.
  ↳ for purposes of making it easier to reference in future (improves readability, ie, changing 'amount' to 'Gross Profit'.)

INNER JOIN : Joins data that have keys that are common to both tables

```
SELECT * FROM tableA
INNER JOIN tableB
ON tableA.col_match = tableB.col_match
```

This part dictates which columns are matched up (ie. which columns contain the common keys).

GUNJOTH

INNER JOIN : Only combines rows which have a mutual key between them, all other rows are ignored.

LEFT JOIN : Keep all rows from left table, but only add data from right table whose key exists in left table.

FULL JOIN : Grabs everything from both tables, regardless of commonality

UNION : Essentially merges the info from two tables together by appending the data from one table below the other table. (The columns must be the same in both tables).

GH NOTE : For the sake of 80/20, this is enough for JOINS. JOINS essentially allow data from two different tables (including 1 table with itself) to be merged in several different ways. Many different merge styles exist. I just need to google them when I want to use them.

GH REFLECTION : SQL actually has so many different capable queries, that it will be easy to get what I want by just explaining my query in plain english (without trying to explain it in smaller SQL terms) to ChatGPT, and get it to build the query for me. I can even end the prompt with something like 'ASK me any clarifying questions you need to in order to construct the query, but ask the questions in plain english as I do not understand SQL, so the questions need to be in laymen terms'.

GUYHOTH