# Week 8 - SQL

DATA201/422

2024-09-13

## Overview

SQL is ubiquitous and even your phone contains a number of SQL databases (SQLite). Going into the field of Data Science, you will need to learn to process data using SQL. In this lab, we will just have a taster to understand some basics. We will interact with the SQL database using R through RStudio. In practice you sometimes have to do this but IDEs like *pgAdmin* are preferred. We have not included *pgAdmin* as our poor little hearts can't take adding new software this late in the semester!

## Use Case

In the present lab, we are budding Data Scientists working for a DVD rental company. To the Zoomers, a DVD is a little round disc that can play movies. Think Netflix with extra steps. Rather than streaming from the comfort of your own home, you used to travel to stores to lend DVDs.

We will investigate the DVD rental database to determine how we could use it to provide insight to the company.

## Deliverables

We will produce code in an R script that will achieve the following objectives:

- List the tables
- List all the fields in a table
- Pull some data using a query
- Plot the data
- Perform a join and pull the result
- Investigate an execution plan.

## Getting started

The data is on a PostgreSQL server hosted by the school. We will need the `RPostgres` package in order to connect to it. First you can connect to a SQL server using the following code. The result is an object that contains all the relevant connection information. You must reference this object whenever you want to call to the server.

```
# Create a connection object
con <- dbConnect(
  Postgres(),
  dbname = "Jeff",
  host = Sys.getenv("PG_HOST"),
  port = Sys.getenv("PG_PORT"),
  user = Sys.getenv("PG_USR"),
  password = Sys.getenv("PG_PASS")
)
```

Now pay attention to the `Sys.getenv()` function. Why have we done this ? Well, you don't want to have usernames, passwords, or connection information in plain text then send it off to GitHub. There are heaps of ways to manage this, we will use environment variables. When you call the function mentioned, it looks in an environment file and pulls the relevant variable.

## Set up environment variables

The easiest way to access your environment file for R is using the following package and function `usethis::edit_r_environ()`. In the file that opens up, paste the following variables then you must *restart R*.

```
PG_HOST = mathmads.canterbury.ac.nz
PG_PORT = 8909
PG_USR = student_data422
PG_PASS = readonly
```

You can check that your environment variables are being called and the database connection is live using `dbIsValid(con)`.

## Listing tables and fields

Here are some example functions you can run to understand the database or retrieve relevant information. There are also two examples of queries being run against the database.

```
# List all tables on the database
dbListTables(con)

# List the column names for a table not in the example
dbListFields(con, "rental")

# Pull the first ten rows of the rental table
dbGetQuery(con, "SELECT * FROM rental LIMIT 10")

# Pull the total number of rentals
query <- "SELECT COUNT(*) FROM rental"
dbGetQuery(con, query)
```

## Now lets start plotting data

The first thing we want to do is analyse the total income each day from rentals. You will need to write a query that selects the data from the correct table. You will then need to convert the datetime variable into only a date variable (we don't care about the hour payment was made, only the day).

Plot the result as a line graph with dates on the x-axis and total daily revenue on the y-axis.

## Joining across tables

The join functions we use in the `tidyverse` originated from SQL. So now we will join across multiple tables to consolidate information and report finer details about the rental business.

## Inventory stock take

The hypothetical situation is we want to do a stock take of R-rated movies across our stores. You must use SQL to get the counts for each movie category in each store. Then plot the data as a bar chart comparing the two stores. You should not do any manipulation or cleaning of the data after it is pulled into R. As a hint, your SQL query must have the following:

- Aliasing of variable names
- You *must* use a common table expression (CTE)
- *Five* joins
- A where statement
- A group by
- An aggregating function (to count up DVDs)

## Optimisation

As we discussed in class, query optimisation is as complex as it is arcane. What is important to take away from this class is merely how to identify where your queries may have trouble so you know where to start.

In PostgreSQL you can get a visual representation of the query in *pgAdmin* using analyse. However in R it is a slightly more ghetto experience. The query function you can use is `EXPLAIN` before your SQL query. This will estimate the best execution path and the cost of each step. If you use `EXPLAIN ANALYSE` it will actually execute that plan and give timings. The former is great when you have a very large query and want to optimise or check for problems before executing it.

Put `EXPLAIN ANALYSE` in front of your previous query, and determine which step you would look to first in order to optimise your query. Note that small data and queries like this would not typically require optimisation.

Answer the following questions:

- Which step would you start with for optimisation ?
- What does that step do ?

## Conclusion

Thats us for today and we finally disconnect from the server. Always be a tidy kiwi.

```
dbDisconnect(con)
```