session_1

December 6, 2020

1 Working with SQL databases

Author: 'Felipe Millacura' **Date:** '6th December 2020'

1.1 Learning Objectives

- Understanding how SQL databases work
- Be able to retrieve information from an SQL database

1.2 Introduction

1.2.1 What is a database?

A database is just somewhere for us to store our data. There are many different shapes and sizes of database. SQL is a language which is often used to query these databases and that's what we will be learning today.

1.2.2 What do we do with databases?

Task - (2 mins) Have a think about it and give some examples of data you might store in a database.

Solution

The possibilities are truly vast: customers and customer orders. Climate sampling sites and climate data gathered there.

What sorts of manipulations do we make to data in databases?

- **Create** (at some point, usually before we are involved, data goes into a database)
- **Read** (we need to get data out to perform analyses)
- **Update** (occasionally, we may need to be able to change data)
- **Delete** (occasionally, we need to be able to remove data from our database)

We refer to these four operations as **"CRUD"**. As data analysts, we are naturally most interested in **Read**ing from databases: we want to import data into an analysis environment and produce useful and actionable insights. But, we'll briefly show you how to perform the other three operations, just so you have seen them in action!

1.2.3 Why databases are needed?

1. Databases *can store very large numbers of records efficiently* (they take up little space).
2. It is *very quick* and easy to find information.
3. It is *easy to add new data and to edit or delete* old data.
4. Data *can be searched easily*, eg 'find all Ford cars'.
5. Data *can be sorted easily*, for example into 'date first registered' order.
6. Data *can be imported into other applications*, for example a mail-merge letter to a customer saying that an MOT test is due.
7. *More than one person can access the same database* at the same time - multi-access.
8. *Security may be better* than in paper files.

My company uses Excel and it's just fine

Feature	Maximum limit
Open workbooks	Limited by available memory and system resources
Total number of rows on a worksheet	1,048,576 rows
Total number of columns on a worksheet	16,384 columns

SOURCE: https://www.bbc.com/news/technology-54423988

1.2.4 What is SQL?

"SQL" stands for "Structured Query Language" (pronounced either as "ess-queue-ell" or "sequel"). There are number of different 'versions' of SQL in common use: some freely available and others proprietary. You should be aware that there are lots of different versions of SQL out there: PostgreSQL, MySQL, Oracle, SQL Server etc...

The core functionality of each of the versions tends to be the same, and it's really only core functionality we'll be looking at here!

1.2.5 Database structure

In SQL, a database is a collection of **tables**. A table is a collection of **columns** and **rows**.

- A table describes the type of item that we want to store.
- A column represents some information we might find interesting about that item.
- A row is the physical data we want to save.

For example, we might have a `zoo` database with a table called `animals`. The `animals` table might have the columns `name`, `age` and `species`, and we'll also add in an `animal_id` column to keep track of individual animals. The `animals` table would then have rows of data like:

animal_id	name	age	species
1	Leo	12	Lion
2	Tony	8	Tiger

animal_id	name	age	species
3	Matilda	6	Cow
4	Winnie	12	Bear

1.2.6 Table relationships

Often the tables in a database are **related** to each other in some fashion, and in fact people often use the terms 'SQL database' and 'relational database' interchangeably, as virtually all relational databases use SQL.

Let's see an example of a relationship. Imagine we expand our `zoo` database by adding a `diets` table to help the zookeepers track what to feed each animal.

diet_id	diet_type
1	herbivore
2	carnivore
3	omnivore

So far so good. But now we need to say **which diet each animal should receive**.

Think of the two tables we have in the database so far: `animals` and `diets`. Can you see how to change the database to indicate which diet each animal in the `zoo` should receive?

Hint: We need to add something to the `animals` table. Can you think what?

Solution

We need to **add an extra column** to the `animals` table, like so

animal_id	name	age	species	diet_id
1	Leo	12	Lion	2
2	Tony	8	Tiger	2
3	Matilda	6	Cow	1
4	Bernice	12	Bear	3

This establishes a **relationship** between the two tables! Every row in the `animals` table is now linked to a row in the `diets` table.

1.2.7 Creating a database

Before we can do anything though, we need to create a table to store our records in. But before we can create a table, we have to create a database to put it in!

```
-- sql terminal
CREATE DATABASE database_name; -- REMEMBER SEMI COLON
```

For example

```
-- udacity_students.sql

CREATE DATABASE udacity_students;
```

1.2.8 Creating tables

By convention, we will name our database tables as the plural of the thing we are creating. So rows of animal data would be stored in a table called `animals`. Sheep would be stored in a table called... well, `sheep`, but you might want to call it `sheeps` to make it clear it's a table holding data on multiple sheep. Sometimes grammar has to suffer for technical clarity!

We can use the following template every time we create a new database table:

```
CREATE TABLE table_name (
  column_name1 DATA_TYPE,
  column_name2 DATA_TYPE,
  column_name3 DATA_TYPE
)
```

So before we run off and create lots of shiny tables, we need to talk about datatypes. You'll be glad to hear they roughly match up to what we have already seen in R. There are many data types we can use in SQL - the most common we will be using are:

- VARCHAR - fixed length text (string)
- INT - integer numerical data (4-byte integer)
- REAL - continuously valued numerical data (8-byte floating point)
- BOOLEAN - true / false data (TRUE, FALSE, booleans)
- TIMESTAMP - date and time information

We can pass arguments to VARCHAR to say how large we want the data in the field to be as a maximum.

Let's continue with the previous example

```
-- udacity_students.sql
CREATE TABLE students (
  name VARCHAR(255),
  on_track BOOLEAN,
  points  INT,
  connected TIMESTAMP

)
```

1.2.9 Reading (C-R-UD)

This is the R in CRUD. We give only a brief introduction here, as a more detailed lesson is coming up!

We have been 'reading' records with the `SELECT` command.

```
-- udacity_students.sql
SELECT *
FROM students;
```

The star tells SQL that we want **all** of the fields returned. If we instead said:

```
-- udacity_students.sql
SELECT name
FROM students;
```

then only the `names` will be returned by the query.

You can also subset a table based on a condition that **must** be met

```
-- udacity_students.sql
SELECT name
FROM students
WHERE points > 500;
```

or order the results in `ASC` or `DESC` way

```
-- udacity_students.sql
SELECT name
FROM students
WHERE points > 10
ORDER BY points; -- ASC by default
```

what if we want just the top 5?

```
-- udacity_students.sql
SELECT name
FROM students
WHERE points > 10
ORDER BY points DESC
LIMIT 5;
```

There are multiple operators you can use

```
> --greater than
< --less than
>= --greater or equal to
<= --less or equal to
= --equal to
!= --not equal to
<> --also not equal to (American National Standard Institute - ANSI)

LIKE -- used in a WHERE clause to search for a specified pattern in a column
AND -- displays a record if all the conditions separated by AND are TRUE.
OR -- displays a record if any of the conditions separated by OR is TRUE
IN --  allows you to specify multiple values in a WHERE clause
NOT -- displays a record if the condition(s) is NOT TRUE
```

1.2.10 Updating (CR-U-D)

This is the U in CRUD.

We use the `UPDATE` command to change the values in existing records.

Template:

```
UPDATE table_name SET column_name1 = new_value1;
```

Let's update the `on_track` column to `TRUE`!

```
-- udacity_students.sql
UPDATE students SET on_track = TRUE;
```

2 Deleting (CRU-D)

This is the D in CRUD.

To delete records we use the `DELETE` clause. But **be careful**, there's no undo! When a record is deleted from a DB it's gone for ever. "Undelete" in the database world is "restore from last night's backup" (if there *was* a backup...)

Template:

```
DELETE FROM table_name WHERE column_name = target_value;
```

SPOILERS Let's delete some students (mean I know!)

```
-- udacity_students.sql
DELETE FROM students WHERE name = 'Felipe Millacura';
SELECT * FROM students;
```

WARNING: If you don't specify the row(s) with a WHERE clause, it will delete *everything* in that table!

```
DELETE FROM students; -- DELETES EVERYTHING FROM STUDENTS
```