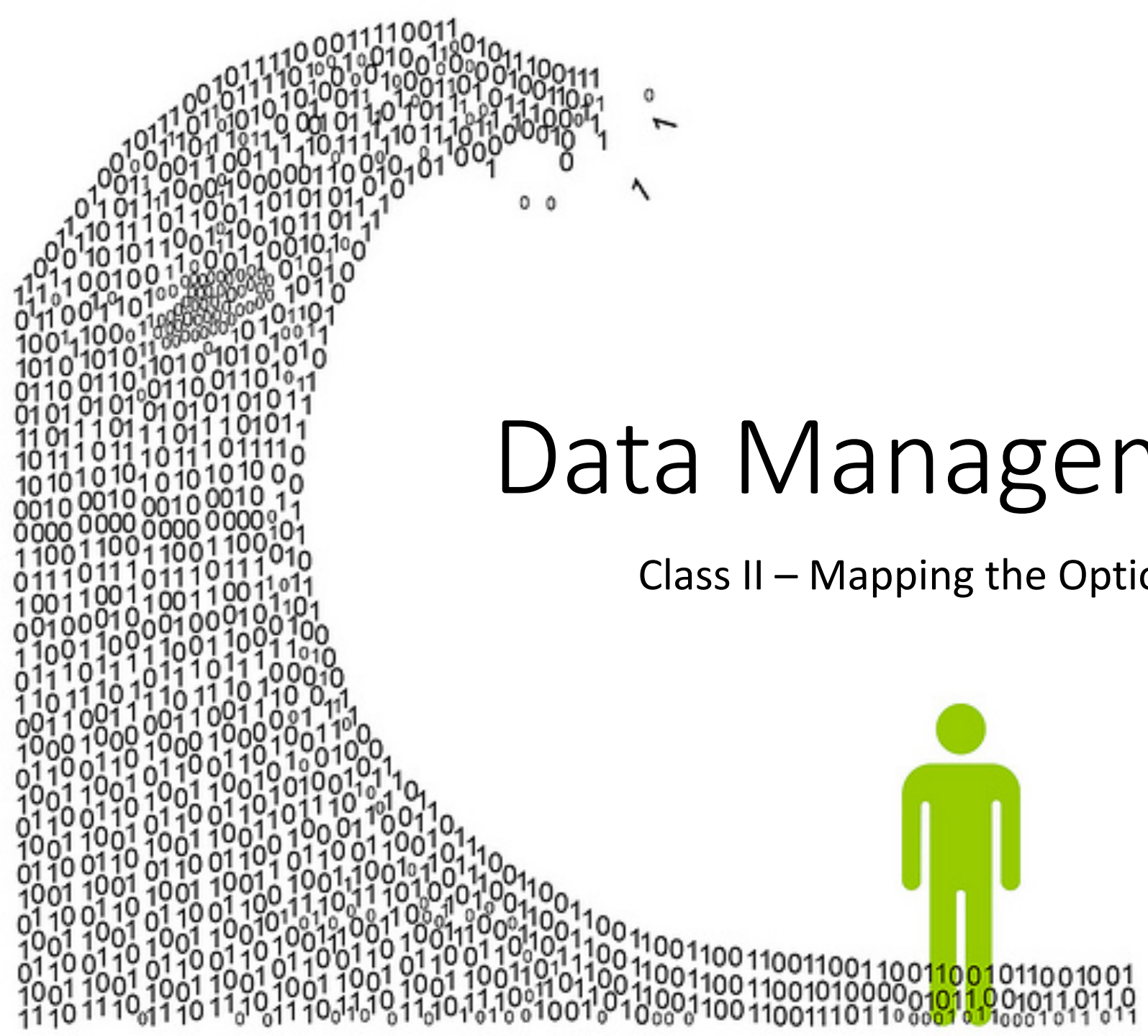
Data Management

Class II – Mapping the Options



A Map of the Options

- In the rest of this course we'll be doing practical exercises to introduce you to different types of data management tools and platforms.
- First, however, we should spend some time introducing each of these tools – showing how they fit in relative to each other, their strengths and weaknesses, and what problems you might use each of them to solve.
- It's okay if you don't remember detailed information about each tool – the objective is to grasp the broad concepts, especially about the trade-offs involved in using different types of tool.

The Basics: File Sharing

- If you don’t have a huge amount of data and are working with a relatively small team – there’s no need to over-complicate things. Ordinary files, properly labelled and stored in a folder, can serve perfectly well for data management.
- You still need to figure out a few things:
  - **Formats:** What kind of files will you use? Can everyone on your team read/access the formats you’ve chosen?
  - **Versioning:** How will you ensure that everyone always has an up-to-date version of the data?
  - **Backups:** How will you ensure that the data isn’t lost if something happens to your computer?

“Sneakernet”: Files on USB Drives

- This is the simplest (and oldest) way of storing and sharing data – flat data files such as text files or CSV files, kept in a folder and shared between colleagues using USB sticks (or discs, or some other physical medium).
  - (”Sneakernet” is a really old term for this – it refers to the fact that the “network” is literally your sneakers, since you’re carrying data around on foot.)
- It’s simple…
- But it’s useless for collaborations with colleagues far away, and it’s impossible to keep track of data versions.

Network Sync: Dropbox / OneDrive / etc.

- There are many commercial services that let you synchronise a shared folder between different users over the Internet.
- For many smaller projects, this kind of service (*along with a good backup plan*) is perfectly sufficient for data management!
- Dropbox (etc.) keeps everyone's copies of the data the same (so there are no versioning problems), and allows sharing over long distances.
- However: there is a practical limit to how much data you can share; and this approach only works for "flat file" data.

A Word about File Formats

- When we talk about ”sharing files” in any way, it’s important to note that this term itself hides a world of complexity.
- Data files come in many formats – plain text files, Excel files, CSV files, Stata (.dat) or SPSS (.sav) files, and many others besides.
- Moreover, many of those types of file can be different internally:
  - Text might be ‘encoded’ in various different ways (especially complex with non-European languages);
  - Excel files etc. may be specific to certain versions of the software;
  - CSV files can have different encodings and also different ‘dialects’ depending on how they were produced…

General Best Practices

- Always save text or CSV files in a Unicode (“UTF”) encoding. This ensures maximum compatibility across operating systems, software and locales.
- Where possible, use a free, open and easily readable format like CSV instead of a commercial format like Excel or Stata files – not everyone may have the software required to read those.
- Avoid complexity. The more “fancy” things you add to data files – like embedded graphs or macros in Excel files – the more likely it is that other users won’t be able to open them.

The Next Step: Databases

- With a CSV or Excel file, you just load all of your data into R, Python or SPSS, and start working with it. But if there's a huge amount of data, several problems emerge…
  - Processing very large data sets can be extremely slow, even for simple functions;
  - In some cases, you simply can't load the data set at all because it doesn't fit in memory;
  - Appending new data or altering even one line of data in a huge file requires everyone else using the file to re-download the whole thing.
- Once you start to encounter problems like this, using a **database** becomes a much more efficient strategy.

What is a Database?

- In simple terms – a database is a piece of software that's engineered to store a large amount of data and allow you to search, filter and do other operations on that data as efficiently as possible.
- Unlike normal files, which have to be either loaded all at once or accessed sequentially, databases are "random access" – you can home in and read or alter records right in the middle of the database with ease.
- Depending on how your database is set up, you and your colleagues may be able to access it simultaneously – even from different parts of the world.

Types of Database

- Many different kinds of database exist. Some of them are designed for general usage, while others have very specific usages – later on we'll briefly discuss one of those specific ones, network databases.
- Choosing a database inevitably means making a trade-off.
  - Some databases are very flexible about the kind of information they hold, but that comes at the cost of speed and efficiency.
  - Some databases are stored in the Cloud and can be easily accessed from anywhere in the world, but that comes with extra costs and overheads.
- Almost any database can be accessed directly from R, Python or any other language, so they fit well with your data workflow.

The Classic: SQL Databases

- Simple Query Language, or SQL, is a simple-looking but very powerful language for inserting, filtering and extracting information from databases.
- It's tried and tested – used for almost every data transaction in the world since the 1970s.
- Many kinds of database software use SQL, so once you know the language, you can easily use anything from a small database on your own laptop right up to a giant cloud database handling millions or billions of records.

SQL Databases in a Nutshell

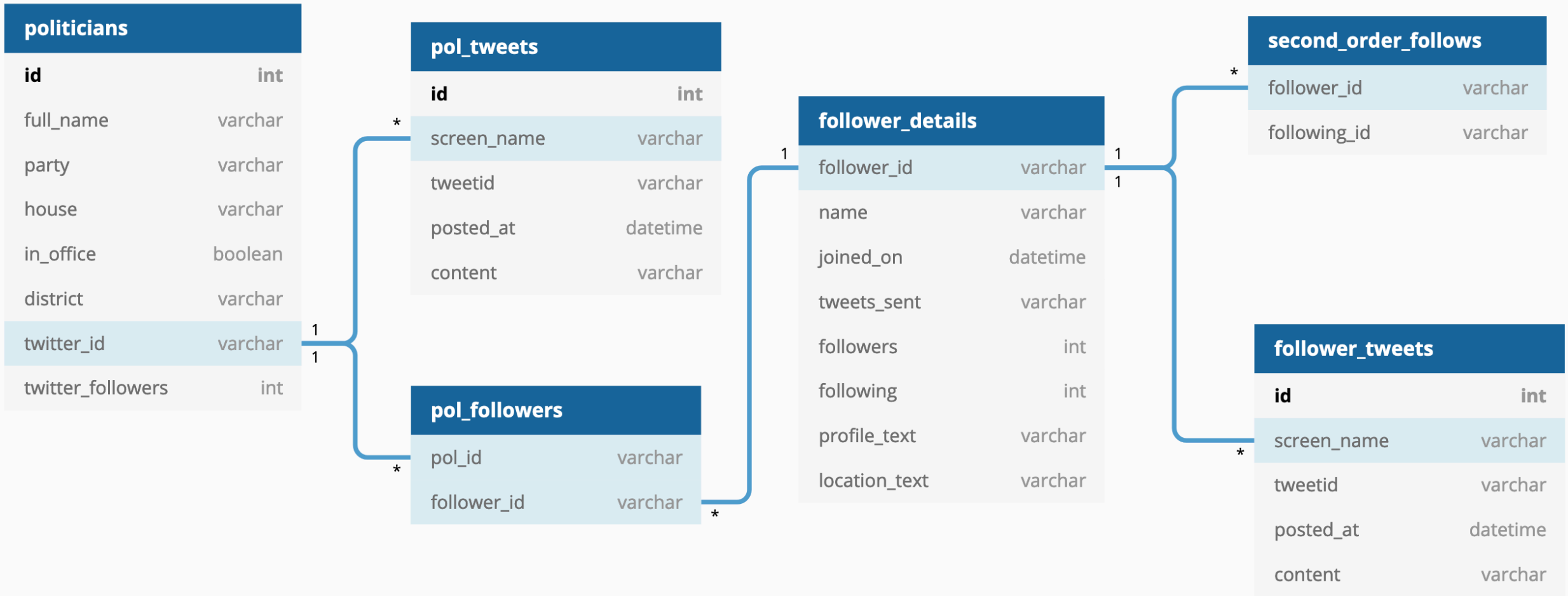
- SQL databases are made up of “tables” – two-dimensional sheets with rows (entries) and columns (variables), much like a spreadsheet or a dataframe.
- Before you put data into such a database, you have to define the table – how many columns it will have, what sort of data will be in each one, and so on.
- This means SQL databases aren’t very *flexible* – your data has to conform to a specification you decide in advance.
- However, they are *extremely fast*. You can run complex queries, matching against multiple filters, over millions of records, in seconds.

Queries?

- Let's step back a moment, and discuss what we mean by "queries".
- At the heart of every database is the ability to *ask your data a question*.
  - A simple question might be, "which records in my database are dated between May 2017 and September 2017, and a certain column matches this keyword?"
  - Much more complex queries are also possible!
  - For example, in a database containing a table of tweets and a table of users who sent them… We could ask for all the tweets sent on certain dates, by users who have more than a certain number of followers, and who have tweeted about certain keywords.

SQL Tables

- The second example touched on a powerful aspect of SQL databases – you can store *multiple tables*, and those tables can refer to each other. (SQL databases are sometimes called “relational databases”, because the tables have relationships to one another.)
- In this example, every user would have one entry in the *users* table, but this would be linked to many entries in the *tweets* table. We might also create a *followers* table so we can create queries about tweets sent by users who follow specific other users, for example.
- The ability to link different kinds of data stored in different tables, and write queries using all of them, is a huge advantage of SQL databases.


politicians
id int
full_name varchar
party varchar
house varchar
in_office boolean
district varchar
twitter_id varchar
twitter_followers int
pol_tweets
id int
screen_name varchar
tweetid varchar
posted_at datetime
content varchar
pol_followers
pol_id varchar
follower_id varchar
follower_details
follower_id varchar
name varchar
joined_on datetime
tweets_sent varchar
followers int
following int
profile_text varchar
location_text varchar
second_order_follows
follower_id varchar
following_id varchar
follower_tweets
id int
screen_name varchar
tweetid varchar
posted_at datetime
content varchar
1
1
*
*
1
*
1
1
*
*

SQL Queries

- SQL itself is written in a way that, at a basic level, looks a bit like English…
  - ```
    SELECT * FROM articles
    WHERE timestamp BETWEEN ‘2017-05-01’ AND ‘2017-09-30’
    AND category = ‘newspaper’;
    ```
- More complex queries, however, can get hard to decipher!
  - ```
    SELECT t.body FROM tweets t
    INNER JOIN users u ON u.userid = t.userid
    WHERE u.followers > 500
    AND t.timestamp BETWEEN ‘2017-05-01’ AND ‘2017-09-30’;
    ```

SQL Software

- ”SQL” is a standard that’s widely used by a lot of different software – that’s what really makes it worth knowing.
- For regular use on your own laptop or similar, you can download MySQL (which we’ll use in the class) or PostgreSQL for free. These are among the most popular databases in the world, and widely used commercially as well as in research.
- Other SQL databases include expensive platforms like Oracle or Microsoft SQL Server.
- The databases offered by cloud services can handle vast amounts of data, but they also use a form of SQL to query and search that data.

The Newcomer: “NoSQL”

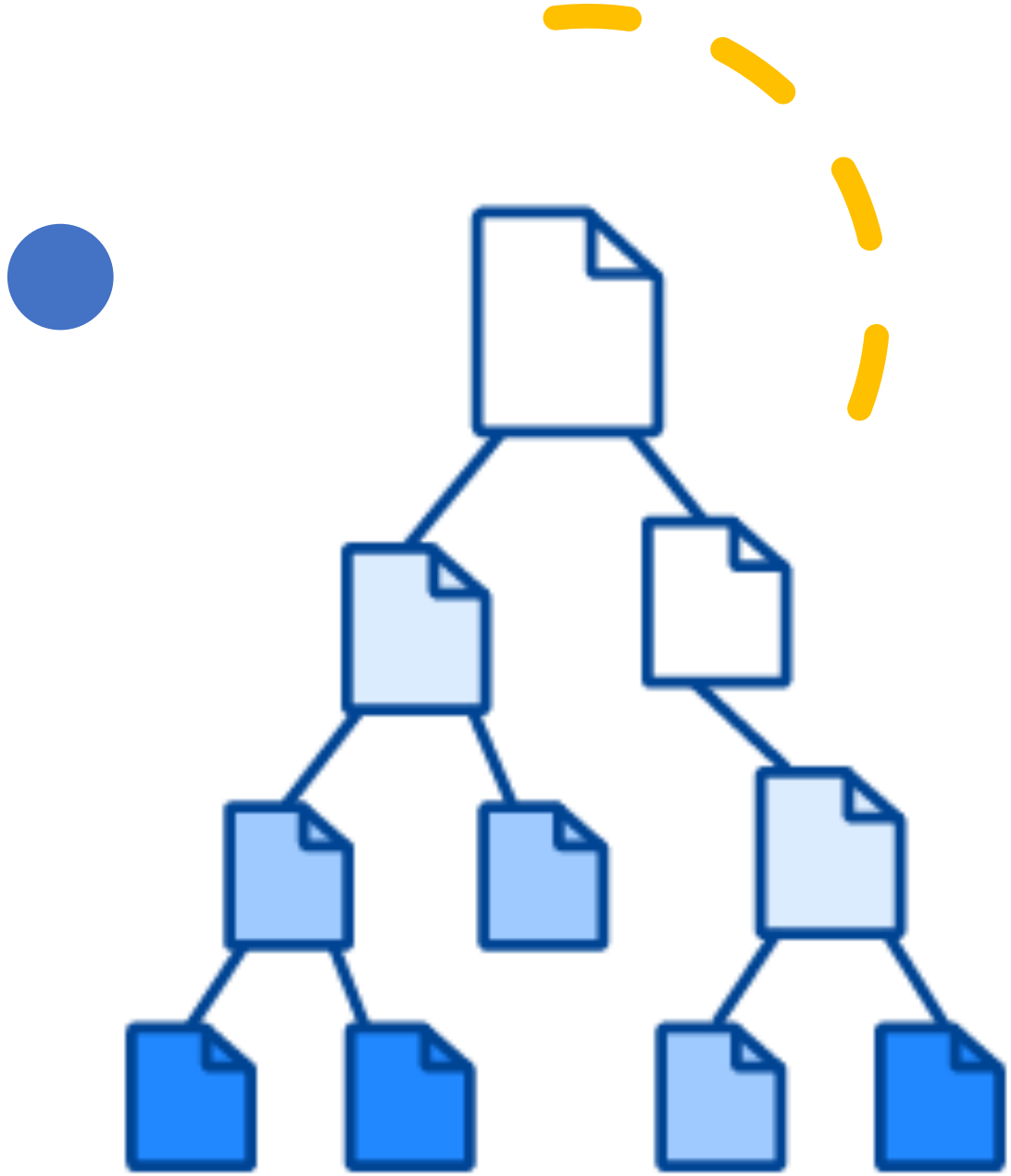
- Recently, a new class of database has become popular – and has powerful applications for researchers.
- These databases are called “NoSQL” databases, because they break the SQL paradigm that’s been popular since the 1970s.
- They’re also sometimes called “document stores”, rather than “databases”.
- The basic idea is that they can store data which doesn’t fit neatly into tables of rows and columns. Each “document” in the database can be a different size and can even contain a different structure of information.

“Document Stores”

- A “document” in database terms is a collection of “keys” and “values”. For example, here’s a simple ‘document’…
  - ```
    {‘userid’: 13854686,
     ‘username’: ‘some_guy’,
     ‘firstname’: ‘John’,
     ‘familyname’: ‘Smith’,
     ‘married’: True,
     ‘spousename’: ‘Carol’,
     ‘children’: [{‘name’: ‘Mary’, ‘age’: 7},
                  {‘name’: ‘Tom’, ‘age’: 4}]
    }
    ```
- This format is called ‘JSON’ – ‘JavaScript Object Notation’ - and is widely used to store and share non-tabular data.
    ```

The Document Tree

- The previous example includes something SQL can't do easily… The "children" field contained a list of children, which could contain any number of entries, each with their own variables and information.
- In SQL, you'd need to make a whole new kind of table to store data about children, and link it to the parent's entry… But in NoSQL, you can just make a list of them.
- That's because NoSQL documents *can contain other documents* and have no fixed format or size.



NoSQL Queries

- To query a document store, you generally just ask for documents that match certain criteria. For example:
  - `db.politicians.find( {‘party’: ‘M5S’, ‘in_office’: True } )`
- You can also do more complex searches using operators – for example, let’s get every person who has two children and is over 40:
  - `db.people.find( { ‘children’: { $size: 2 }, ‘age’: { $gt: 40 } } )`

NoSQL Advantages and Trade-Offs

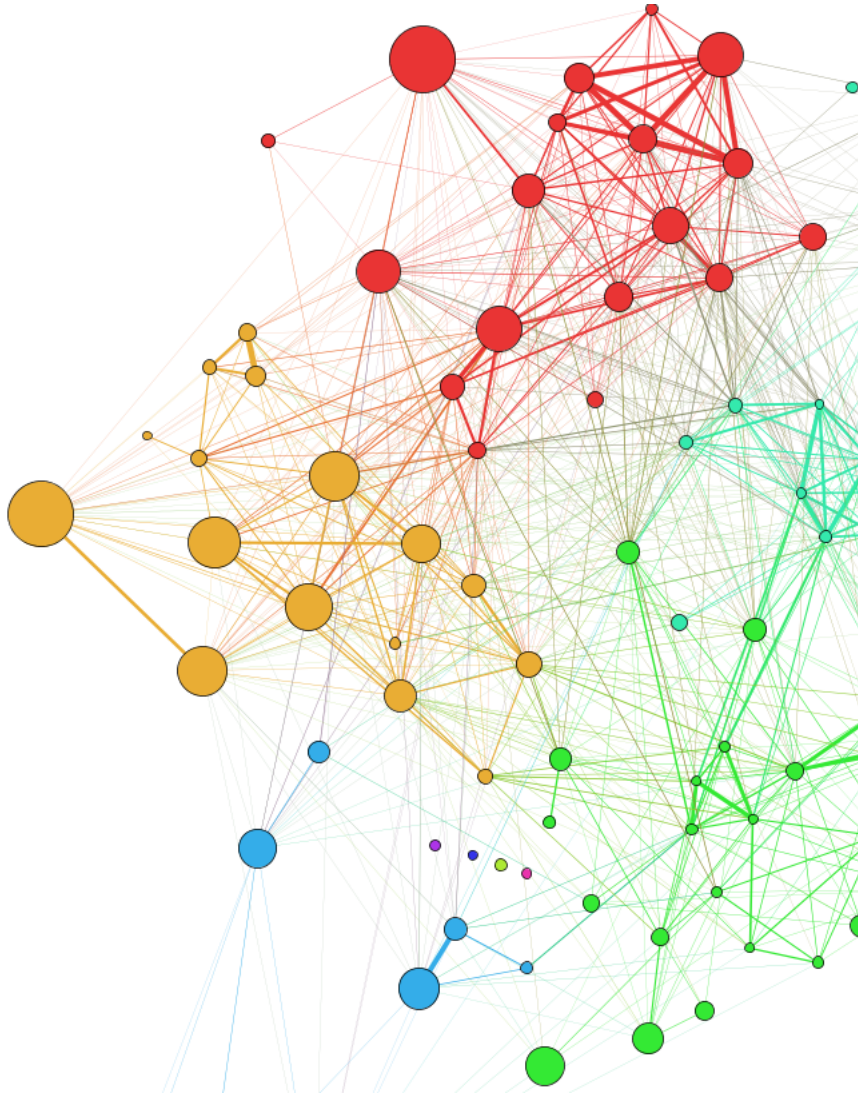
- NoSQL document stores are much more flexible than MySQL; they're great for storing unstructured data, or data that may not follow a very strict format.
- <u>However…</u>
- NoSQL is slower and less efficient than MySQL. It can store huge amounts of data, but in many cases won't be very fast at searching through it all.
- A lot of the powerful things you can do inside SQL by combining data from related tables and so on, will need to be done with your own Python or R program if you're using a NoSQL database.

NoSQL Software

- **MongoDB**, which we'll use in this class, is probably the most popular NoSQL software right now.
- Most Cloud service providers have their own version of a NoSQL document store, which usually works pretty similarly to MongoDB (but on a larger scale)
  - Amazon Web Services: **DynamoDB**
  - Google Cloud Platform: **Cloud Datastore**, **Cloud Firestore**
  - Microsoft Azure: **Cosmos DB**

The Unusual: Network Databases

- Network data is important to many different fields of research – from physics and biology, to medicine and social science.
- A network is made up of “nodes” (points in the network) and “edges” (connections between nodes) . Each node or edge can have data associated with them.
- You *can* store network data in a normal database – but it’s a bad fit. Network queries (e.g., *“show me Twitter users within two degrees of separation from this person”*) aren’t what regular databases are designed for.



Querying Networks

- To query a network (or a “graph”, as they’re commonly called in this field), you need a special kind of query language. Here’s an example in Cypher, which is the language used by the Neo4J graph database.
  - ```
    MATCH (a:TwitterAccount)-[:FOLLOWS]-(b:Politician)
    WHERE b.Party = “M5S”
    RETURN a
    ```

- This query will return the Twitter account of every person who follows a politician in the M5S party. The unique thing about graphs databases is the ability to search based on the relationship defined in the first line of this query.
    ```

Network Database Software

- Network / graph databases are a pretty new type of software. The market leader is **Neo4J**, which you can download and use for free; it's arguably the easiest graph database to learn.
- Its main competitors are **OrientDB** and **ArangoDB** - these are more like general-purpose NoSQL databases with some network graph functions.
- If you want to use a graph database in the Cloud, Amazon offers one called **Neptune**.

The Buzzword: Cloud Platforms

- Cloud services like Amazon Web Services, Google Cloud Platform or Microsoft Azure basically let you move a big part of your data management (and maybe your analysis etc. as well) off your own laptop and onto the Internet.
- "The Cloud is just someone else's computer" is a popular saying in IT circles, and it's partially true – "the Cloud" is just a massive set of warehouses full of powerful server computers owned by Amazon, Google etc.

How the Cloud works

- To use the Cloud you rent storage space and processing power from one of those warehouses. In general, you only pay for what you use – for example, to use a Cloud database you often just pay a tiny amount for storing your data, and then pay a little each time you do a search (which uses CPU power and memory).
  - When you're not using the data, you only pay for the storage cost.
- This means you can use resources you can't afford to buy. Your research project almost certainly can't afford to buy a supercomputer, but you can "rent" supercomputer-level processing from Google or Amazon for just the few minutes it takes to do your analysis task.

Other Cloud Advantages

- There are two other major advantages to Cloud services…

1. You can access them from anywhere in the world, and it's easy for you and your research collaborators to all work on the same data sets and code together.

2. Backups and data security are taken care of for you – data stored in the cloud is backed up regularly, and often you can go back through months of previous backups if you need to restore an earlier version of your data.

3. Cool toys: image and voice recognition APIs, machine learning tools…

So why *not* use the Cloud?

- Steep learning curve: to use Cloud services, you have to learn how to operate the Cloud platform and its tools, which are mostly designed to be used by programmers and IT professionals. You also need to be a relatively accomplished programmer to use most services.

- Significant overhead: even when you know how to use the Cloud, it will add an extra layer of complexity to many things you do. Your workflow will often involve moving data and code from your local computer to the Cloud and back.

The Cloud and Big Data

- When your data reaches a size that's too big for an ordinary computer, you have no choice but to use the Cloud for it.
- Tools like Google's "BigQuery", which we'll look at next week, use a variation of the SQL language that's designed to let you search through gigantic databases – many terabytes of information.
- You may end up storing your "big data" in the Cloud, using a tool like BigQuery to extract the specific bits of data you want to analyse, and putting *that* data into another database on your own laptop temporarily…
- Or just doing the whole analysis process using Cloud services.

This afternoon…

- We'll be starting to work with Python in this afternoon's class, so please don't forget to bring your laptop along with you.

- See you at 14.30!