An Overview of Databases and Data Storage Technologies

Elise Hellwig

2024-02-29

Workshop Description

This workshop provides a broad overview of the various technologies for storing and organizing different collections of data. We will discuss how data structure and data types impact your storage options, when you should use a database, and which platforms you might consider for your research. This workshop is a general lecture with case studies and Q&A (no laptops necessary). This workshop is a prerequisite for DataLab's Intro to SQL for Querying Databases and Spatial SQL with SpatiaLite workshops as well as being part of the Research Computing micro-credential

Learning Objectives

After completing this workshop, you will be able to answer the following questions:

- What is the difference between information and data?
- What are the most common data structures people use?
- Which data structures lend themselves to answering different types of research questions?
- What is a database?
- What are the differences between relational and noSQL databases?
- When would a database help answer a research question?

Prerequisites

No prerequisites or software required. This workshop is designed for researchers with active or planned data collection and storage.

Image Credits

All images were created by Elise Hellwig unless otherwise noted.

Data

> Quantities, characters, or symbols on which operations are performed by a computer, considered collectively. Also (in non-technical contexts): information in digital form.
>
> - Oxford English Dictionary

If you are doing research, chances are you are going to be working with data in some form or another. Even if you do not use traditional rows and columns, you are certainly working with some form of information. And if you have translated information into a form that is more easily shared or stored for later use, you are working with data. Information becomes data when we give it some sort of structure. That could be the

previously mentioned rows and columns, or something more human readable, like the grammar and syntax of natural language.

One of the challenges of working with data is figuring out the best way, or at least a good way, to store it. Thankfully, we have mostly left behind the 20th century's favorite way of storing data: paper in filing cabinets. Our new methods are definitely less flammable but also have a higher barrier to entry. There are many more types of data stores available now as well. This means you will need to make a decision on which one to use. Ultimately, the best way to store your data will depend on the data you want to store and the questions you want to ask with it.

When describing data we generally talk about two main characteristics: data type and data structure.

Data Types

Your data's type tells the computer what sorts of operations make sense. Common data types include:

- integers
- decimal
- floating point numbers
- characters (text)
- Boolean values (TRUE and FALSE)
- Dates and times

Data type also influences how data is stored. Unlike other characteristics we will discuss later, an individual data point can have a type.

Data type is not necessarily inherent to a piece of information, though it is often strongly suggested. For example, you can store the value of 1 as the integer 1, the decimal 1.0, the text "1", the Boolean TRUE, or the date 1900-01-02. The underlying information is the same (1), but the computer will treat each of these pieces of data differently. It will not allow you to add "1" to another number, and it may have issues combining the integer or decimal 1.0 with text. Text itself can only be stored as characters.

Data Structures

> A data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.
>
> Wikipedia

Unlike with data types, a single piece of data cannot have a data structure. This is because data structure is about capturing relationships between data points, there isn't any meaningful information to be gained from the relationship of a data point with itself. We will discuss five of the most common data: tabular/columnar, graph, tree, categorical/relational, and spatial data. In practice, data often exhibits traits from multiple data structures.

As mentioned previously, data is information translated into a form to store for later use. The type of information you work with does not dictate the type of data structure you must use. However, some types of information lend themselves to particular data structures over others.

When trying to decide what data structure to use to represent information from your research, the key is to focus on what sorts of questions you would like to ask. Each data structure will make some types of questions easier to ask and others harder. If you want to ask disparate types of questions, you may need to represent your data using multiple structures. Because data structure has such a large impact on how you interact with your data, the structure you choose is just as important as the analysis you do afterward.

Columnar/Tabular

Tabular or columnar data is made up of rows and columns, collectively referred to as a table. By convention, each row represents an observation, and each column a measurement or aspect of that observation. Tabular

data is incredibly common, especially in the sciences. Many researchers, regardless of field store at least some of their data in tabular form. Tables work well when all of the objects in your study can be summarized with the same set of characteristics.

The table below contains data from the US magazine *Motor Trend* in 1974. In the table below, each row is an observation of a car model released between 1973 and 1974. The magazine recorded the make, model and origin of each car and measured the car's fuel efficiency (mpg), horsepower, weight, and number of cylinders in the engine.

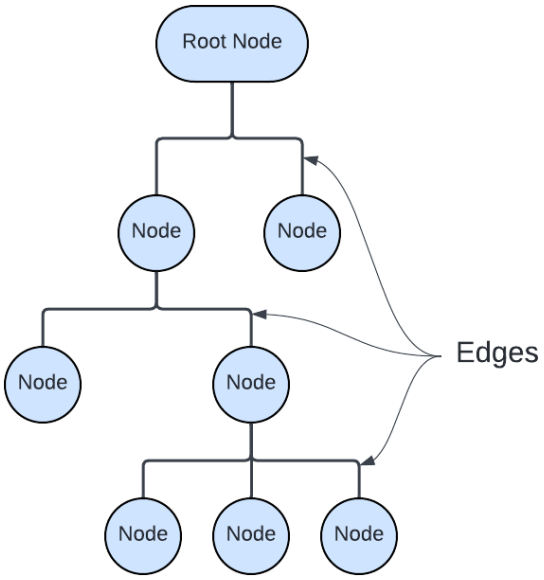
```
##                      mpg cylinders horsepower weight manual origin
## Mazda RX4           21.0         6        110  2.620      1  Japan
## Mazda RX4 Wag       21.0         6        110  2.875      1  Japan
## Mercedes-Benz 240D  24.4         4         62  3.190      0 Europe
## Mercedes-Benz 230   22.8         4         95  3.150      0 Europe
## Mercedes-Benz 280   19.2         6        123  3.440      0 Europe
## Mercedes-Benz 280C  17.8         6        123  3.440      0 Europe
## Mercedes-Benz 450SE 16.4         8        180  4.070      0 Europe
## Mercedes-Benz 450SL 17.3         8        180  3.730      0 Europe
## Mercedes-Benz 450SLC 15.2        8        180  3.780      0 Europe
## Toyota Corolla      33.9         4         65  1.835      1  Japan
## Toyota Corona       21.5         4         97  2.465      0  Japan
## Lincoln Continental 10.4         8        215  5.424      0    USA
## Chrysler Imperial   14.7         8        230  5.345      0    USA
## Dodge Challenger    15.5         8        150  3.520      0    USA
```

Other examples of data sets that lend themselves to a tabular data structure include California almond yields (lb/acre) for farms around the state, climate data over the past 50 years, snow accumulation (inches) for Lake Tahoe, and biomarkers for people participating in a study on a chronic illness.

Tabular data lends itself to answering questions about the average value or spread of a particular measurement over the entire dataset. It can also help investigate trends in a particular variable, or how two variables relate to each other. In the case of the car fuel efficiency data, we could ask about the relative fuel efficiency of cars from different parts of the world or if there is any relationship between a car's weight and its horsepower. The tabular data structure also makes sorting observations very easy.

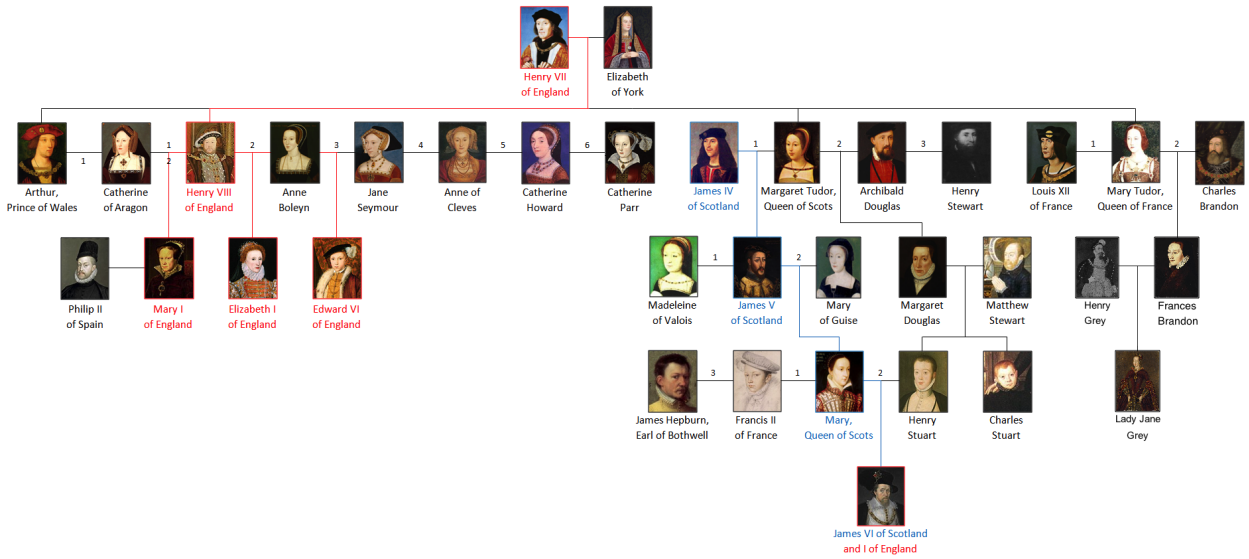
Trees

Trees are another commonly used data structure. Tree structured data explicitly capture hierarchical relationships between entities. Trees are made of up nodes (or vertices) which represent entities in the data, and edges which represent relationships between those entities. The number of edges separating an entity from a common root node defines that entity's place in the hierarchy. When viewing tree structured data on a computer, it often looks very "nested", like sections and subsections of a document. In fact, tree data is sometimes referred to as document structured data because of this.

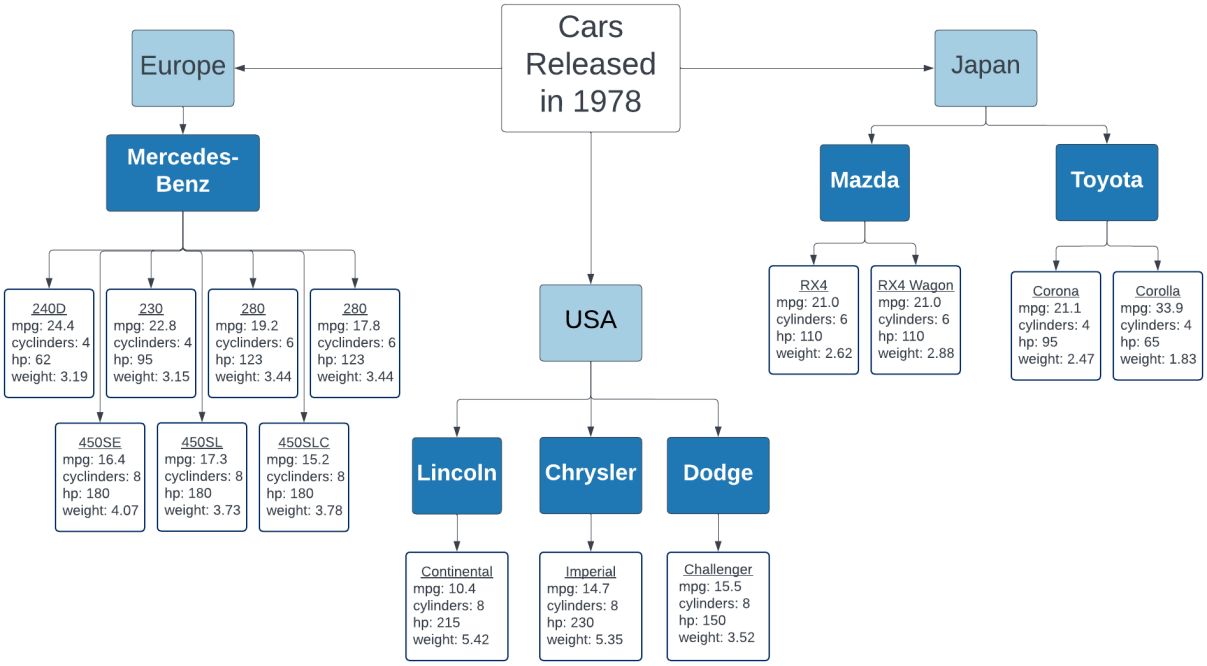


In general, tree data facilitates asking questions about particular subsets of the data. For example, you might want to know how closely two entities are related, which branches of the tree have the most nodes, or all of the observations that fall under a particular subtree.

Family trees are a common example of tree structured data. In family trees, the nodes represent people in the family. Horizontal lines represent marriages or couplings, and vertical lines represent a connection to offspring. In the family tree of the Tutor Dynasty in England below, it is easy to determine things like who had the most wives (Henry VIII), which people shared a generation and who was the last member of the dynasty (James I of England). Questions like the what was the average age at death for the Tudors would be harder to answer.

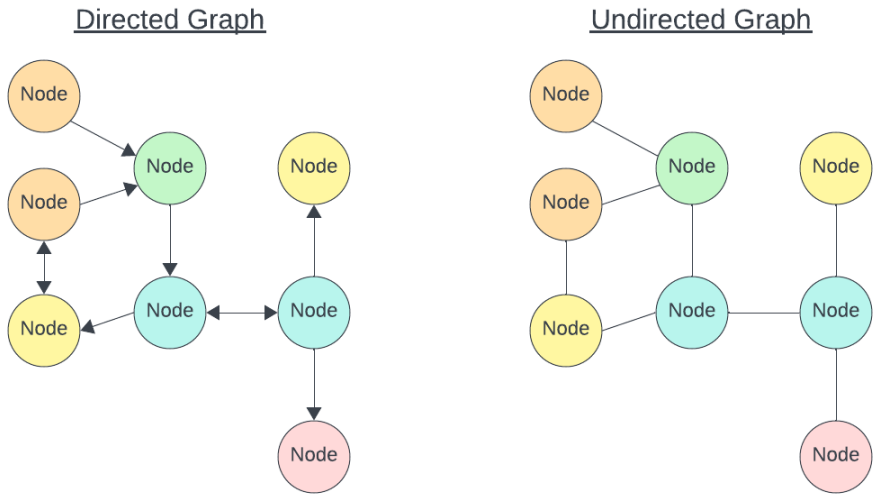


We can also represent the car fuel efficiency data with a tree data structure. This makes it much harder to answer some of the questions mentioned in the previous section. On the other hand, the tree structure makes it very easy to see which car manufacturer produced the greatest number of models in 1973-74 and how many different models came out of Japan during that time period.

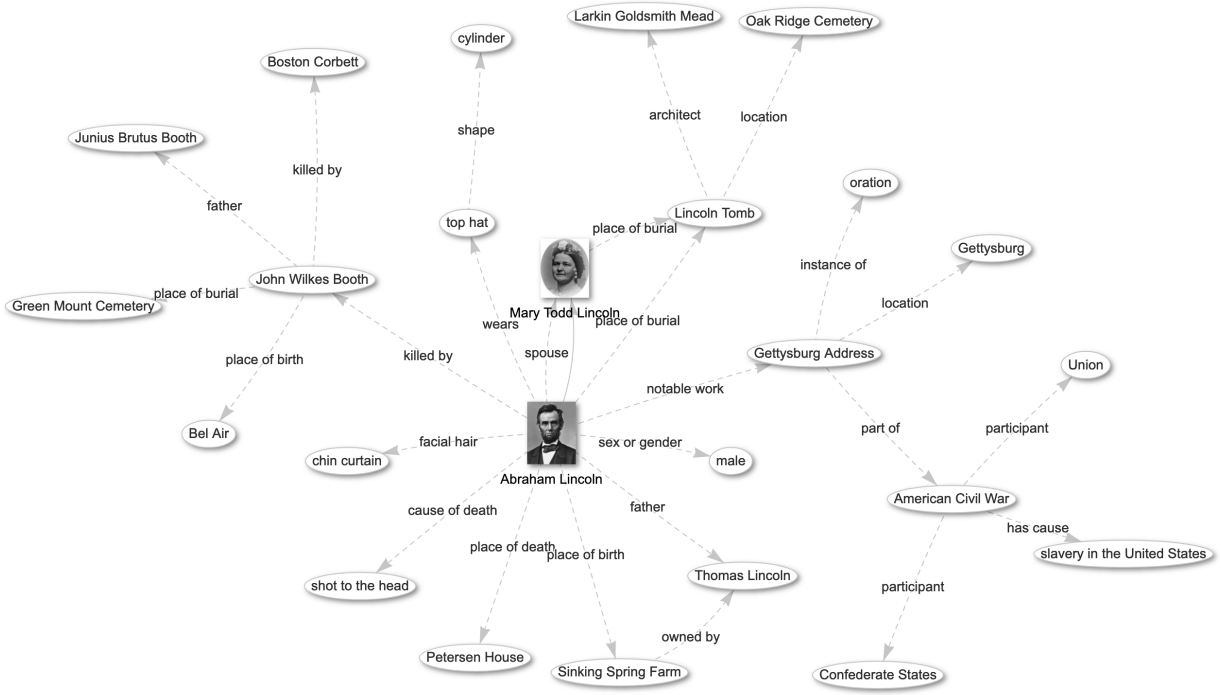


Graphs

Graph data is similar to tree data in many ways. It also explicitly captures information about relationships between entities using nodes and edges. However, unlike trees, graph data sets have no inherent center or root node, and there is no implicit hierarchy or nesting. Additionally, the relationships between the vertices can be directional, and that direction does not have to be consistent throughout the graph.



Graph structured data tends to be more common in the social sciences and humanities. One of the most ubiquitous examples of a graph is the social network. Transportation data, like roads and flight paths, also lend themselves to the graph structure. In the digital world, we can use graphs to map out the link connections between Wikipedia pages, as seen below with Abraham Lincoln.



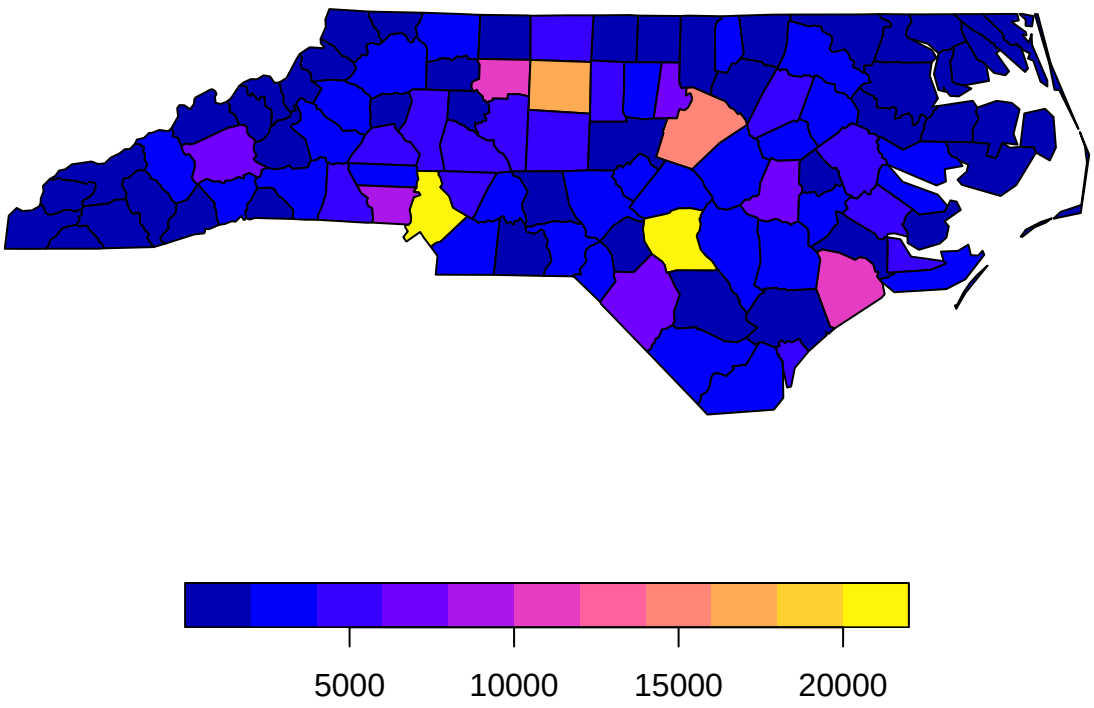
Graphs make it easy to ask questions about connectivity. For example we might want to know how many flight connections we will need to make between Honolulu, Hawaii and Cairo, Egypt, or how many degrees of separation there are between Abraham Lincoln and Hitler on Wikipedia. We can also ask about which parts of the graph are the most connected and which are the least connected.

Categorical - Relational

In categorical data, category membership is the organizing principle of the data structure. Entities can have membership in multiple categories, and relationships are defined between categories instead of the entities themselves. As such, relationships tend to represent complex processes rather than individual entities.

Spatial

In spatial data, the organizing principle is location. This can be geographic location (ex latitude/longitude) or relative location on an arbitrary grid (ex. [2, 4]). Data can either be continuous, like a sheet over a landscape, or discrete like points on a map. In fact, map making is one of the most common uses of spatial data. However, we can also ask as questions like whether points are clustered more closely than we would expect, and whether we can use nearby data to predict values for places where we don't any.



Data Stores

> A data store is repository for persistently storing and managing collections of data which include not just repositories like databases, but also simpler store types such as simple files, emails etc.
>
> Wikipedia

Once you have decided on a data structure, the next decision to make is what type of data store to use for data storage. A data store is a repository for storing and managing data. While this workshop covers exclusively digital data stores, for most of human history, data stores were analog. Even today, we still use analog data stores like libraries, museums, and filing cabinets. Data stores have two primary functions. First, they need to facilitate people accessing the data (reading) and making changes to it (writing). Second, they need to preserve this access over a period of time.

Regardless of your data's structure you have two general options for how to store your data digitally: in flat files or in a database. Probably the most common type of data store is a flat file. A flat file is stand alone file on your computer that is not linked to data elsewhere. You can store tabular data in .xlsx and .csv files, tree data in .xml and .json files, graph data in .GML and .DOT, and spatial data in .GeoJSON and .GeoTIFF files. All of these file types have applications you can use to access and modify your, or you can load the data into a programming environment like R or python. In addition, many of thse file types, like csvs and jsons are human readable, meaning you do *not* need special software to access or modify the data they contain.

In comparison, a database is an organized collection of data that users can interact with through specialized software called a database management system, often abbreviated DBMS, but that is not human readable. Databases have several key characteristics that differentiate them from other data stores like flat files.

1. **Structure** - Databases can store data with very complicated structures.
2. **Scalability** - Databases can store vastly information more than other digital data stores without

modification. Additionally, databases support substantial computing upgrades, which means you can maintain performance as your data set grows.

3. **Access** - Databases support concurrent access and have sophisticated methods for managing simultaneous user requests. They also provide fine-grained controls on who can access, modify, and delete data.

Specific types of databases have other differentiating features, like data integrity constraints, flexible data structure support. Databases also have several drawbacks:

- Data in databases is not human readable, unlike many file types including csv, xml, DOT, and GML, which are human readable
- Database software generally has a steeper learning curve than other software. You may need to learn a specialized language to interact with the data in your database.
- There may be cost associated, either for the database software or for hardware needed to run the software.
- Databases require some level of administration. Someone will need to be in charge of adding data to the database, determining who has access to what, and fixing things when they break.

If you are running into any of the following issues, a database may be the solution to your problems.

- Your data set has many repeated values
- Your data set contains information about many unrelated or loosely related studies
- You are sharing data with multiple people and everyone seems to have a different version
- Your data set is so large, your computer slows down or freezes when trying to view it
- You have to keep making the same corrections to your data over and over again
- Excel keeps converting your columns to the wrong data type
- You have so many pivot tables
- You have hundreds (or thousands) of data files you need to share with your collaborators
- You think PowerQuery is neat but wish people would stop changing the underlying data while you are using it.
- You have a hard time finding and extracting the information you need to answer a particular research question

Spreadsheets

Spreadsheets, a type of flat file, are the most ubiquitous form of digital data store. If you have used Microsoft Excel or Google Sheets, you have used a spreadsheet. In fact, spreadsheet applications transformed the use of the personal computer, particularly in business circles. Like some databases, spreadsheets impose a tabular data structure on the data they store. Despite this, a spreadsheet is not a database. Like most modern software, spreadsheet applications have graphical user interfaces, making the barrier to entry very low. Spreadsheet software also offers some built in analysis tools, so you don't have to learn a new application to do basic statistics and data visualization.

Spreadsheet software also suffers from some drawbacks. They offer limited storage capacity, making it difficult to work with large data sets. Excel and LibreOffice have a maximum row number of a little over 1 million and Google Sheets limits you to 10 million cells. Software operations will also slow down if you get close to their data capacity. This means analyzing large data sets is much more tedious than analyzing small ones.

Spreadsheets also store data in ways that cause a lot of data duplication and make data entry susceptible to typos. In the spreadsheet visualized below, we have data on the Die Hard franchise. Specifically, we have information about the actors that starred in the series and the movies in which they appeared.

```
##      title_id person_id             primary_title premiered runtime_minutes
##  1: tt0095016 nm0000246                  Die Hard      1988             132
##  2: tt0095016 nm0000614                  Die Hard      1988             132
##  3: tt0095016 nm0000889                  Die Hard      1988             132
##  4: tt0095016 nm0001817                  Die Hard      1988             132
##  5: tt0099423 nm0000246                Die Hard 2      1990             124
##  6: tt0099423 nm0040472                Die Hard 2      1990             124
##  7: tt0099423 nm0000889                Die Hard 2      1990             124
##  8: tt0099423 nm0001817                Die Hard 2      1990             124
##  9: tt0112864 nm0000246 Die Hard with a Vengeance      1995             128
## 10: tt0112864 nm0000460 Die Hard with a Vengeance      1995             128
## 11: tt0112864 nm0000168 Die Hard with a Vengeance      1995             128
## 12: tt0112864 nm0001295 Die Hard with a Vengeance      1995             128
## 13: tt0337978 nm0000246     Live Free or Die Hard      2007             128
## 14: tt0337978 nm0519043     Live Free or Die Hard      2007             128
## 15: tt0337978 nm0648249     Live Free or Die Hard      2007             128
## 16: tt0337978 nm0702572     Live Free or Die Hard      2007             128
## 17: tt1606378 nm0000246    A Good Day to Die Hard      2013              98
## 18: tt1606378 nm2541974    A Good Day to Die Hard      2013              98
## 19: tt1606378 nm0462407    A Good Day to Die Hard      2013              98
## 20: tt1606378 nm0935541    A Good Day to Die Hard      2013              98
##                         genres rating  votes                    name birth death
##  1:           Action,Thriller    8.2 936512            Bruce Willis  1955    NA
##  2:           Action,Thriller    8.2 936512            Alan Rickman  1946  2016
##  3:           Action,Thriller    8.2 936512          Bonnie Bedelia  1948    NA
##  4:           Action,Thriller    8.2 936512     Reginald VelJohnson  1952    NA
##  5:           Action,Thriller    7.1 380561            Bruce Willis  1955    NA
##  6:           Action,Thriller    7.1 380561        William Atherton    NA    NA
##  7:           Action,Thriller    7.1 380561          Bonnie Bedelia  1948    NA
##  8:           Action,Thriller    7.1 380561     Reginald VelJohnson  1952    NA
##  9: Action,Adventure,Thriller    7.6 405383            Bruce Willis  1955    NA
## 10: Action,Adventure,Thriller    7.6 405383            Jeremy Irons  1948    NA
## 11: Action,Adventure,Thriller    7.6 405383       Samuel L. Jackson  1948    NA
## 12: Action,Adventure,Thriller    7.6 405383           Graham Greene  1952    NA
## 13:           Action,Thriller    7.1 418910            Bruce Willis  1955    NA
## 14:           Action,Thriller    7.1 418910             Justin Long  1978    NA
## 15:           Action,Thriller    7.1 418910        Timothy Olyphant  1968    NA
## 16:           Action,Thriller    7.1 418910                Maggie Q  1979    NA
## 17:           Action,Thriller    5.2 214179            Bruce Willis  1955    NA
## 18:           Action,Thriller    5.2 214179            Jai Courtney  1986    NA
## 19:           Action,Thriller    5.2 214179          Sebastian Koch  1962    NA
## 20:           Action,Thriller    5.2 214179 Mary Elizabeth Winstead  1984    NA
##        role                  characters
##  1:   actor          [""John McClane""]
##  2:   actor           [""Hans Gruber""]
##  3: actress [""Holly Gennaro McClane""]
##  4:   actor        [""Sgt. Al Powell""]
##  5:   actor          [""John McClane""]
##  6:   actor             [""Thornberg""]
##  7: actress         [""Holly McClane""]
##  8:   actor             [""Al Powell""]
##  9:   actor          [""John McClane""]
## 10:   actor                [""Simon""]
```

```
## 11:   actor                  [""Zeus""]
## 12:   actor           [""Joe Lambert""]
## 13:   actor          [""John McClane""]
## 14:   actor          [""Matt Farrell""]
## 15:   actor        [""Thomas Gabriel""]
## 16: actress                  [""Mai""]
## 17:   actor          [""John McClane""]
## 18:   actor          [""Jack McClane""]
## 19:   actor               [""Komarov""]
## 20: actress                  [""Lucy""]
```

As you can see, there is a lot of data duplication, meaning our data set takes up more space on our hard drive than it needs to. Additionally, if we have to change the information about a single movie or actor, we would have to make that change in many different places, and potentially miss one or make a mistake. If we accidentally left out an "l" in Bruce Willis's last name, we would have to modify five cells to fix our data.

Finally, most spreadsheet applications store their data locally, so unless you are militant about version control outside the application, it is easy to end up with people working on different versions of the data. Google Sheets offers shared access and some version control, but still fails to deliver on storage capacity, and data integrity.

Examples

- Microsoft Excel
- Google Sheets
- Numbers
- LibreOffice Calc
- OpenRefine
- Open Science Framework

Relational Database Management System

Relational Database Management Systems (RDBMSs), or relational databases, are one type of database. When people use the word "database", nine times out of ten they are referring to a relational database. Like spreadsheets, RDBMSs impose a tabular structure on the data they contain. Unlike spreadsheets the structure of that data can be fairly complex. Relational databases are made up of multiple tables connected by shared columns, called keys.

Users interact with an RDBMS through the Standardized Query Language (SQL) instead of a graphical user interface. SQL is a programming language, but it is one specifically designed for interacting with relational databases. SQL standardizes your interactions with a database, and makes them reproducible. This means it is much easier to ensure everyone is working with the same set of data. However, SQL has a higher barrier to entry than a graphical user interface like Excel.

To learn more about using SQL in practice, check out the Intro to SQL (reader, workshop) and Spatial SQL (reader, workshop) resources.

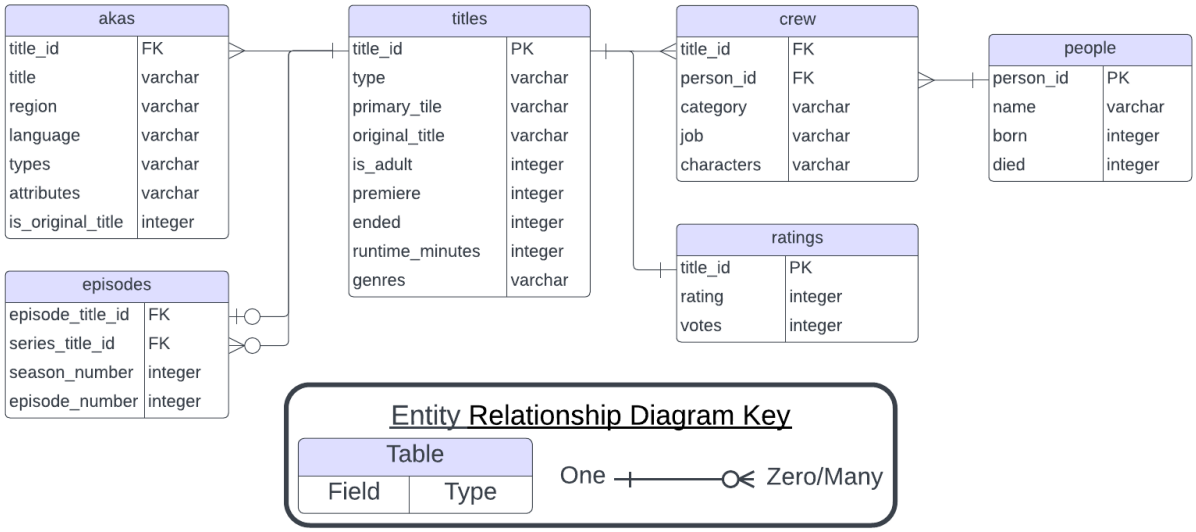
Relationship keys

A relationship key is a column or pair of columns that links two tables. There are two types of relationship keys: primary keys and foreign keys. A primary key, sometimes called an index, is a column that uniquely identifies each row in a particular table. It is generally the first column in the table and its name often contains the letters "ID". A foreign key is a column in one table whose values correspond to a primary key in another table.

Entity Relationship Diagrams

Since relational databases are not human readable, it can be helpful to have a visual summary of a database without actually needing to interact with it. That is where the Entity Relationship Diagram (ERD) comes in. ERDs are a standardized way to depict the structure of a relational database. Each box in the diagram represents a table, with the table title at the top and the column names listed below it. The column data type appears next to each column name. The

Connector lines link primary/foreign key pairs. These lines specify which tables can be combined, and which keys should be used to combine them. The connector lines also specify what type of relationship between the two tables.



The most common type of table relationship in relational databases is the one-to-many relationship. In this case, one row in table A may be linked to many rows in table B. This is the case for the relationship between the people and crew tables in the IMDB. One-to-one relationships are also fairly common in relational databases. Two tables are one-to-one when one observation in table A only corresponds to one observation in table B. This is the relationship between titles and ratings tables in the IMDB database.

As you might guess, there is a third type of relationship: many-to-many. Many-to-many relationships are the sleeping giant of database relationships. They can be very useful, but if you are not careful, they can multiply the size of your data several times over. Library book checkout transactions are one example of a many to many relationship. A single patron can check out multiple books in a single transaction, and multiple patrons can check out a book.

Why Use Relational Databases?

There are three primary reasons to use a RDBMS as a data store:

1. Reduce data duplication
2. Speed up and standardize accessing and updating the database
3. Ensure data integrity

In our Die Hard spreadsheet example, we had 260 cells. However, in a relational database, storing that same data only requires 156 cells. That is a 40% reduction in data size. For a multi-gigabyte data set, 40% makes a huge difference. Additionally, much of that information was stored as numbers instead of text, which makes its size on your hard drive even smaller. Finally, we selected a subset of columns from each table,

which means we didn't need to work with ALL of the data just because we wanted to work with some of it. Subsetting like this is much harder in traditional spreadsheet software.

The source database of our Die Hard data, IMDB, is massive relative to spreadsheet software capabilities. The titles table alone has 10 million observations, and the crew table has over 60 million. Despite this, extracting the Die Hard movie franchise information using an SQL query took about half a second.

The SQL query used to extract the Die Hard data is also reproducible. If I want to collaborate with a colleague on my analysis of Die Hard, I don't have to send the data set to my collaborator. Instead I can provide them access to to the IMDB database and my SQL query. If I find problems in the data at a later time, I don't need to send a whole new data set, which could introduce confusion about which data to use. All I need to do is update the database, and tell my colleague to rerun their SQL query. No additional computer storage necessary.

With an RDBMS, even though the data you analyze may have duplicates, you don't need to modify each duplicate value, in the case that one requires a correction. All changes to the data can be made to the singleton values in their original tables. Correcting Bruce Willis's misspelled last name only requires fixing his name once in the "people" table. Then, any time you extract data from the database table, that change will automatically propagate to the new data. While it may be easy to successfully fix 5 errors, it will be much harder to fix 50, or 5,000. This is especially true if they are scattered throughout your data set and not just in a single column.

RDBMS Software

Because of relational databases' widespread utility, there are many software options to choose from when creating a database. While all of them work off the same basic SQL, each one puts its own particular spin on the language in terms of additional functionality. All of the SQL database software options in the table below have wide community and/or professional support. There are many more RDBMS applications available for more specific use cases (ex. for use with Amazon AWS), but these are the most common and widely supported. All of them provide some level of support for spatial data. However, that level of support varies, so if you have specific requirements, it's best to do additional research before making a decision.

Software	Ease of Use	Documentation/ Support	Cost	License
SQLite	Easier	Extensive documentation and community support	$0	Public Domain
PostgreSQL	Harder	Spotty documentation, but with robust community support	$0 + cost of hosting	Open Source
MySQL	Easier	Large community base, No professional support without paid subscription	$0 - Low Cost	Partially Open Source
Microsoft SQL Server	Easier	Robust professional and community support	High Cost	Proprietary
Oracle	Harder	Robust professional and some community support	High Cost	Proprietary

NoSQL Databases

Not Only SQL, or NoSQL, databases are another type of database. They are primarily defined by what they do not do, namely store data in relational tables, as a RDBMSs would. Instead, NoSQL databases do not necessarily impose external structure on the data they contain. In fact, NoSQL databases can contain data from multiple structures, including

- documents
- key-value pairs

- graphs
- trees
- wide column tables

This makes NoSQL databases are much better at storing tree and graph data, as those data structures do not fit well into relational tables.

Why Use NoSQL Databases?

1. Compatible with multiple data non-tabular structures
2. Capable of storing vast amounts of data with impressive speed
3. Adding more computing power is easier

Because NoSQL databases do not have to adhere to the more rigid structure of relational databases, they can store more data and access and modify it more quickly than relational databases. This is particularly important when working with textual data, which can easily take up many terabytes of storage. It is also generally easier to upgrade NoSQL databases, which is important when you are working on the edge of what is possible data storage-wise.

Example Software

Not every NoSQL database supports every data structure. Some support multiple structures while others specialize in one. Unlike RDBMSs, all of the most common NoSQL software is open source, though the MongoDB license is slightly more restrictive than most open source licenses. Because of this they are also low cost to implement, though you can always pay for additional features and support. Similar to RDBMSs, all the most common NoSQL database software provides some support of spatial data. However, that support is limited to vector data and the functionality available varies fairly widely.

Software	Data Structure	Spatial Support
MongoDB	Tree/Document	vector
Couchbase	Tree/Document	vector
Cassandra	Column family	vector (DataStax)
Redis	key-value	point only
Neo4J	Graph	vector
ArangoDB	Tree/Document, graph, key-value	vector

Choosing the Right Data Store

You've just learned a bunch of information about data structures and data stores. Now the challenge is applying that information to your own data. When you are trying to decide what type of data storage to use, consider these ideas:

1. Different data structures facilitate different research questions.
2. When we convert information to data, we encode it in a structure.
3. All data stores impose structure on your data.
4. The best data storage technology for any research question matches the structure of the data with the structure of the data store.

All information needs to be encoded in a data structure in order for it to be useful in research. When you have identified your research question, you can use that to determine what data structures will make it easiest for you to carry out your research. Finally you can match the structure, size, and use case of your data to the data store(s) that best fit your needs.

It is important to remember that as you ask different questions of your data, you may need to change how you structure and store that data. Reproducibility experts will sometimes require that you only store one "definitive" version of your data. However, that plan rarely survives first contact with a research project. If you want understand a subject deeply, you need to ask many different types of questions about it. And this necessitates storing your data in multiple forms.

Assessment