Databases and SQL

Joseph Hallett

November 11, 2024

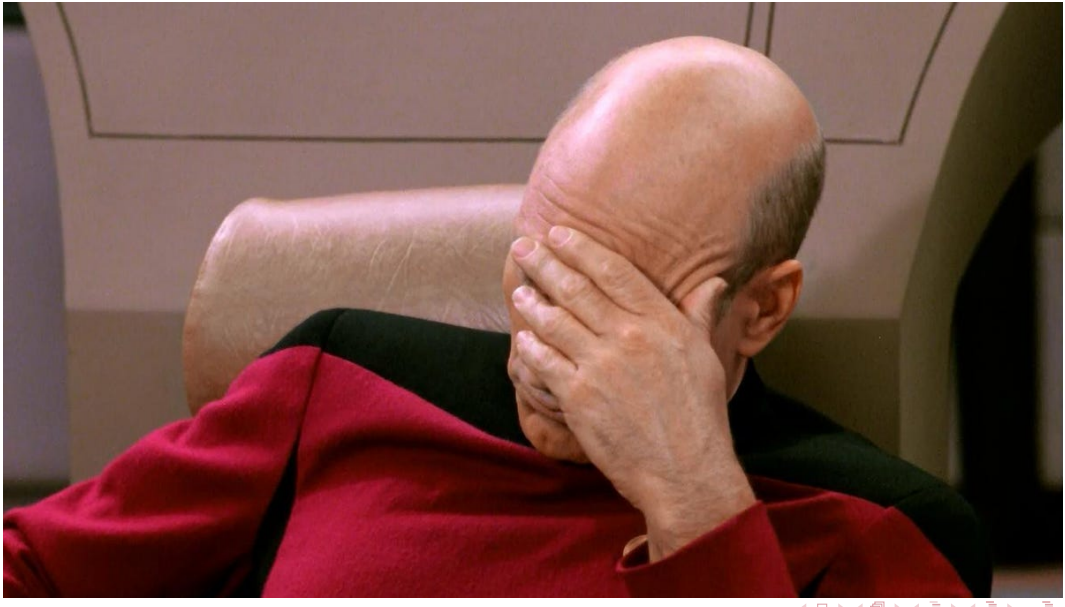
A World-beating track and trace system

During the *recent* global horribleness, then Prime Minister Boris Johnson promised the UK would have a world-beating track and trace system

- It cost £37,000,000,000
  - (84% of what Elon Musk paid for Twitter)
- It could handle 65,536 records
  - It silently forgot more than that
  - (Populatation of UK 67,000,000: ~0.1%)
- It was implemented in Microsoft Excel
  - Using the spreadsheet format from 1987
  - If they'd used the modern format it could handle 1,048,576 records
  - (~1.5% of UK population)
- Conservative estimate is that this killed 15,000 people.

`https://www.medrxiv.org/content/10.1101/2020.12.10.20247080v1`

Computer Scientists everywhere…



Scientists Are SMART

A 2016 study found that about 25% of genome datasets have errors in them

- Excel corrects SEPT2 protein to 2-September by default
- MARCH1 protein corrected to 1-March

https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-1044-7

It's okay we've fixed it in 2020

- SEPT2 now called SEPTIN2
- MARCH1 now called MARCHF1
- Button to turn off automatic conversion bit more prominent (2023)

Seriously?

EuSprig Guidance

```
=IF(use=critical,
   "Formal␣Software␣Engineering␣Process",
   IF(use=important,
     "Use␣a␣database",
     "have␣fun␣with␣spreadsheets"))
```

Dr Simon Thorne, programme chair of European Spreadsheet Risks Intrest Group (EuSpRiG)

Just Use a database…

Databases are great

- Way of storing structured data in a computer
- Once a whole separate degree
  - Now 3 weeks of a CS degree
  - (Hey it was 50 minutes last year…)

Not *wholy* different from a spreadsheet

- But no silly limitations from being a tool for double entry bookkeeping
- Instead of columns in sheets we have fields in tables with explicit types

SQLite

Loads of different database engines

Server based :: MySQL, Oracle SQL, MariaDB, etc

These run on a server and provide distributed access to a single database
Good if you want to keep your database separate from your application

- you're Running a webapp
- you Need multiple people able to connect at the same time

File Based :: SQLite, DuckDB, etc

Run in your application and provide structured storage

- We're doing SQLite
  Good If you just want to structure data
  - You're building a mobile app
  - You need to store data

But Before we go down to SQL

Lets do a bit of *theory*.

- How do we design databases?
- What properties do we want from our designs?
- What mistakes can we avoid ahead of time?

Next week

SQL proper. Databases have a lot of theory behind them

- And There's a **lot** of Jargon
- Lessons learned from designing them

This time

Database theory and tools to get the design *right*.

- …next time how we *actually* write SQL.

Relational Modelling

Databases let us store data in tables!

- What's a table?

Tables hold data

Essentially a spreadsheet.

Rows hold data

- One row per item in the table
- There is no order within the table (rows *should be* independent)

Columns are called *attributes*

- Each attribute describes something about that row in the database

A database contains many tables

- Attributes can refer to data in other tables

Structure

But how do you *structure* your data in a table?

- What patterns are going to make our life easier?
- How can we describe what's in our tables and what the relationships are between things?

Relational modelling

Proviso!

Relational modelling is a tool for thinking about how to decompose relationships between things into tables.

- People get fussy about the syntax

Please don't!

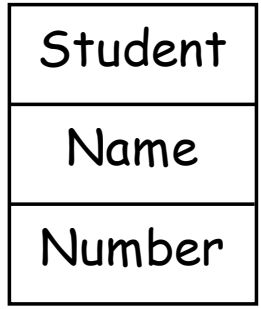
I'll try and show you various syntaxes you may encounter, but its *just a tool*

- Do whatever works for you
- So long as its clear it doesn't matter
- The diagrams are for doodling ideas *not* final implementation

Things are nouns!

Here is a student! Students have a name and a number!

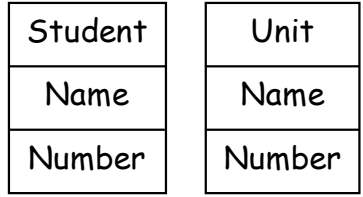
- The student is the *entity*.
- The name and number are the *attributes*.



More things are nouns!

Here is a unit! Units also have a name and a number!

- The unit is the *entity*.
- The name and number are the *attributes*.



Don't worry about names

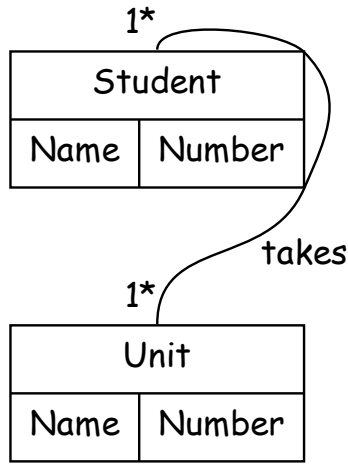
There may be many examples of different *values* that could be examples of units and students… but don't worry about that.

Student	
Name	Patrick McGoohan
Number	6

Unit	
Name	Software Tools
Number	COMS10012

Nouns can be related!

One student may take many units; and units may have many students



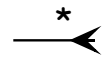
Alternative notation

Some people prefer a graphical notation for entity relationships called *crow's foot*

- I prefer to write it explicitly

Don't get too hung up on notation!

- And use a key if you're ever asked in an exam
- The point is to let *you* doodle notes
- Do whatever makes sense to you or the people you work with



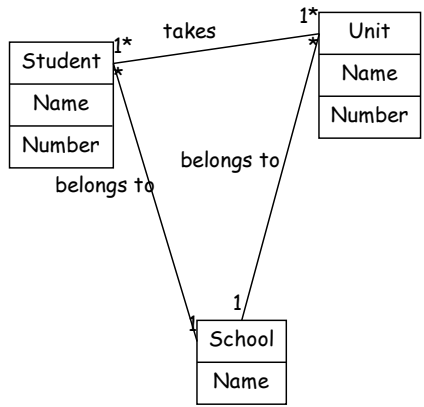
Schools are a thing!

There are things called *schools*:

- Schools have names
- Each unit belongs to *exactly* one school
- Each student belongs to *exactly* one school

Each school can have students and units its responsible for

- But could also be empty!



What should I call a student?

Obviously their name would be *polite* …
…but what will happen if we were to open a class on *Gallifrey*?

(The *truly* pedantic amongst you will notice I've missed John Hurt's *War Doctor*, Richard E. Grant's Doctor from *Scream of the Shalka* and all the regenerations from Rowan Atkinson to Dawn French as part of that Comic Relief sketch from the 90s. Dr Who is complicated; but consider yourself seen.)

All 13!

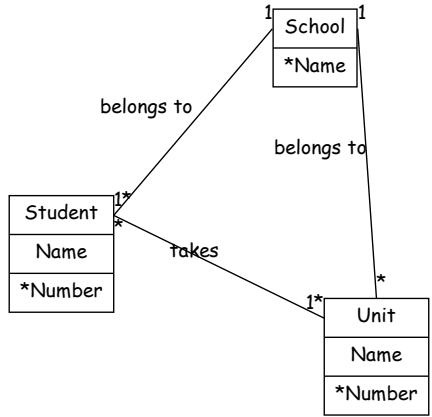
This would rapidly get too confusing for computers!

- (But not for people)

A key for an *entity* is the set of attributes needed to uniquely refer to it.

- *A candidate key* is a *minimal* set of attributes needed to uniquely refer to it.
- The *primary key* for an entity is the key we use.

If a key contains multiple attributes its called a *composite key.*
If a key is a meaningless ID column you added just for the sake of having a key its called a *surrogate key.*



Design of entities

So far we've been sketching the *relationships* between different entities.

- When we come to implement the database each *entity* would be a different table in the database
- But how should we structure the entities themselves?

Suppose we want to store in our database a *reading list*.

- Each unit will have a list of books that they recommend

Is this a good design?

Reading List	
*Unit	Software Tools
*Title	Software Tools
Author	Brian W. Kernighan
	P.J. Plaugher
...	...

Reading List	
*Unit	Software Tools
*Title	SSH Mastery
Author	Michael W. Lucas
...	...

Normal Forms

Bit of database theory designed to *minimize the problems you'll encounter when working with a database.*

- Avoid having to add and update multiple database entries
- Make it easy to delete stuff
- Make it easy to find things
  Loads of different rules

0NF, 1NF, 2NF, 3NF, 3.5NF, 4NF, 5NF, 6NF

- In practice almost everything after ~3.5NF is overkill

We're aiming for highlevel-gist of what they all are

(If you want precise mathematical definitions consult a textbook)

0NF is no rules

So everything meets it by default!

1NF

First Normal Form is each field can only contain one value

- Basically, don't nest databases within databases…

Reading List	
*Unit	Software Tools
*Title	Software Tools
Author	Brian W. Kernighan
	P.J. Plaugher
…	…

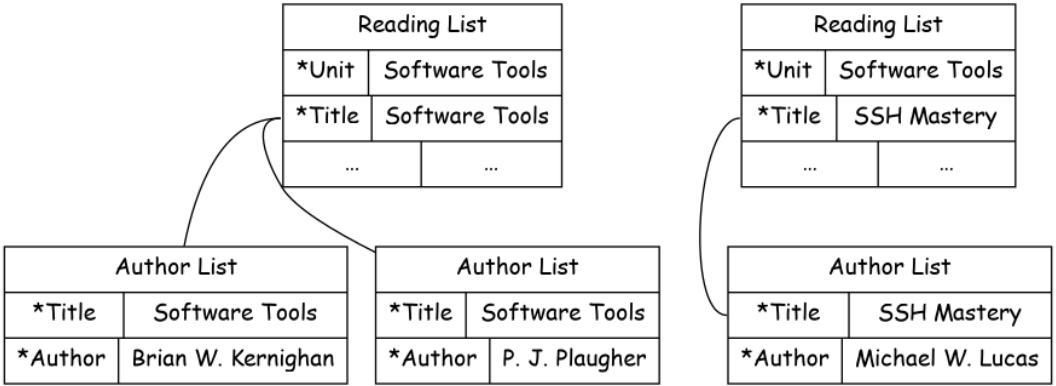
Reading List	
*Unit	Software Tools
*Title	SSH Mastery
Author	Michael W. Lucas
…	…

Why isn't this in 1NF?

Is this in 1NF?

No. Sometimes there's more than one *author* in a single field.

- To fix, pull the authors into a separate table…



Why is this better

Imagine we want to search for all books by *Kernighan*

- With nested structure we'd have to search *within* fields and write regular expressions
- With 1NF we just search for matching *Author List entries*

Imagine we want to change add an author (new editions sometimes do this!)

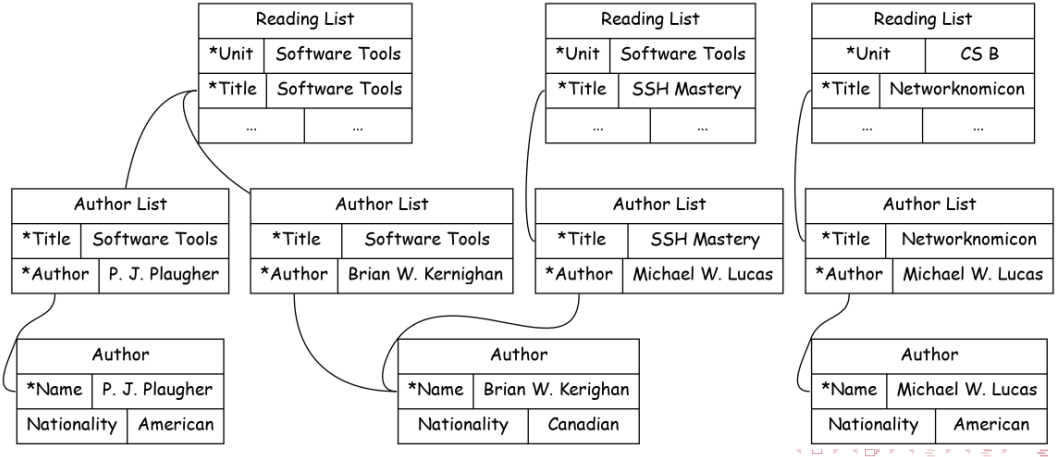
- With nested structure we have to tweak the author field
- With 1NF we just add an extra *Author List* entry

2NF

It is in 1NF AND…

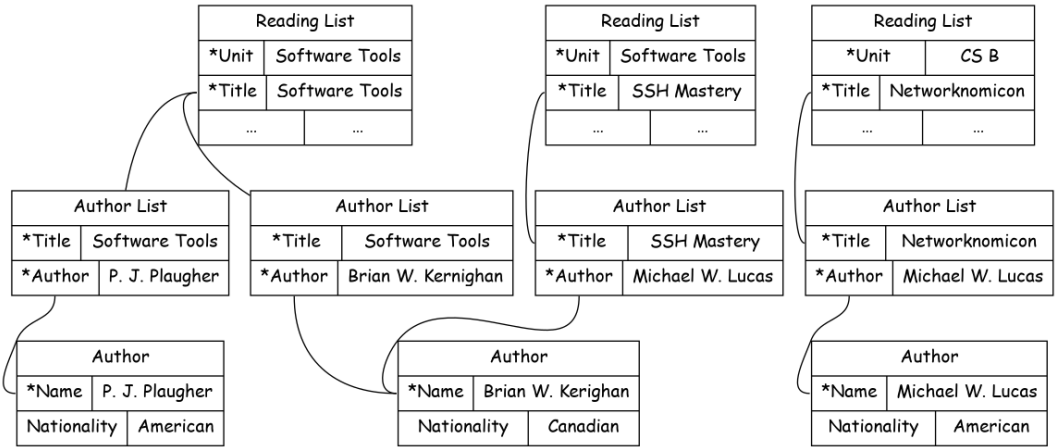
All of the non-key attributes must depend on the whole key

- If your key consists of more than one field
- Make sure everything depends on the whole key and not just part of it…



Is this 2NF?

No, *authors nationality* depends on the *author* but not the *title* of the book, so not the whole key



Why is this better?

Reduces duplicated data.

- So if you need to update something you only need to do it in one place

3NF

It is in 2NF AND

No non-prime attribute is transitively dependant on the primary key

- If some attribute is dependent on the primary key via some other attribute pull it out

Reading List	
*Unit	Software Tools
*Title	Software Tools
Edition	1st
Published	1976

Reading List	
*Unit	Software Tools
*Title	SSH Mastery
Edition	2nd
Published	2017

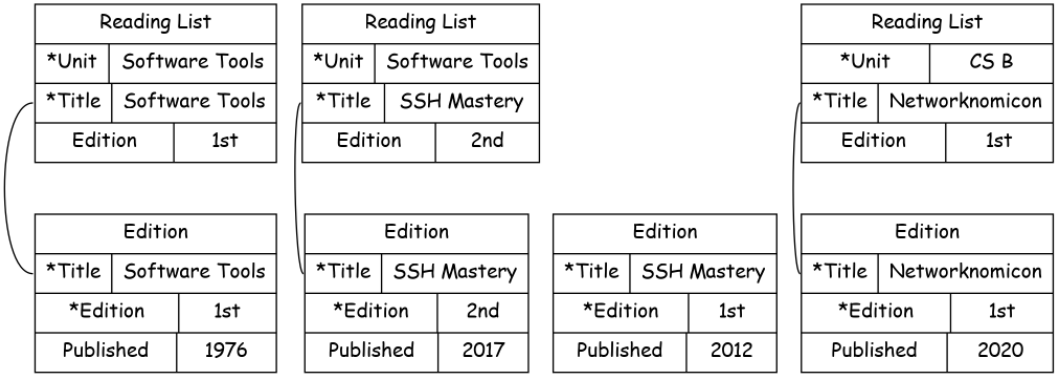
Reading List	
*Unit	CS B
*Title	Networknomicon
Edition	1st
Published	2020

Why isn't this in 3NF?

Is this in 3NF?

No.

- Its in 2NF because *edition* and *published* both depend on the key.
  - You want that edition of the book, or the one published in that year.
- But the year published *could* depend on the edition and the *title*.
  - It works because rarely do authors publish multiple editions in the same year.
    - But it'll become problematic if I set two editions of the same book from the same year.



3.5NF (Boyce-Codd Normal Form)

Actually theres an even stronger version of 3NF…
The database contains no functional dependencies

- If you ensure that there can *never* be a key that wouldn't meet 3NF then its 3.5NF
  - This really only applies if you have multiple things that could be the primary key and one of the things you didn't pick wouldn't meet 3NF.

"Each attribute must represent a fact about the key, the whole key and nothing but the key. So help me Codd."

Why are these better?

Honestly, I'm not *really* sure…

- In practice it seems to work quite well
- Theoretically they *reduce redundancy*…
- The idea is that if you want to make a database access fast you'll create an index via the primary key and these transitive dependencies mess up some of the optimizations you can do and make modifying records messy and you end up duplicating things

Get to 3NF and stop!

And now for ~~smugness~~ bonus points

4NF

Every non-trivial multivalued dependency is a superkey

- If you select every attribute in a row of your table that could be the key

5NF

Every non-trivial join dependency is implied by the candidate key

- You're not joining to something that isn't part of the key?

6NF

Every table contains only key and **at most** one other attribute

- Means you'll have lots of tables and your SQL will be mostly joins
- Used in some data centers, but you're not likely to need this
- (I sometimes write databases like this just because it saves arguments over which normal form it is in)

Recap

We went over entity relationship diagrams

- Lots of arrows, treat them as a doodle if they're helpful to you
- Don't get bogged down in semantics

We went over normal forms

- So long as *everything depends on the key, the whole key, and nothing but the key (so help us, Codd)* you'll be fine
- Enjoy getting bogged down in mathematical semantics!

Next time

Lets actually write some code?

Now

Let's practice together