Relational Modelling (1)

So databases provide data stored in tables

- but how do you structure this data in a table?
- can we visualise relationships and describe the data?

Relational Modelling is another tool for decomposing relationships between things into tables

The key idea in relational modelling is not to store information more than once if you can avoid it. If you have stored in several places that Fred is taking Introduction to Programming, and then Fred switches his units, you don't want to end up with a situation where this change is only reflected in some parts of the database.

Entity Relationship Diagrams

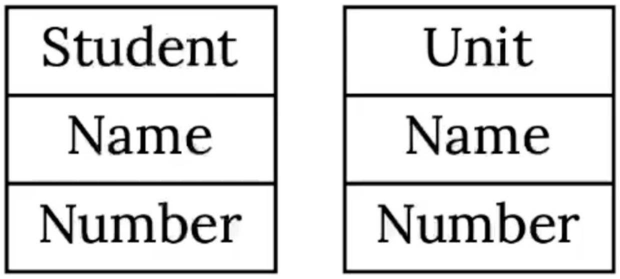
Here is an example for a database contianing students

A student has a name and a student number

- The Student is the **entity**
  - entities are usually nouns
- the name and number are **attributes**

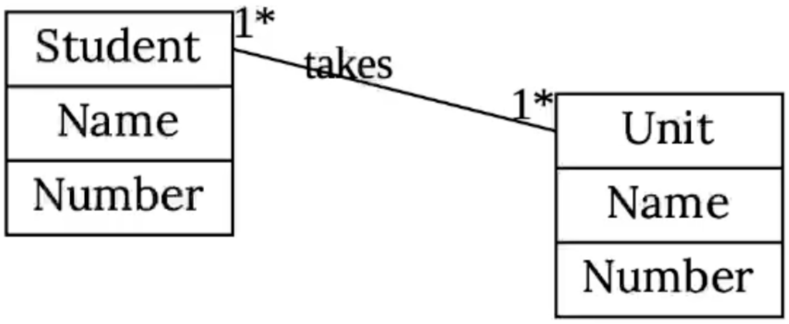
You can have numerous entities (more things are nouns)

Here is a Unit, they also have a name and number

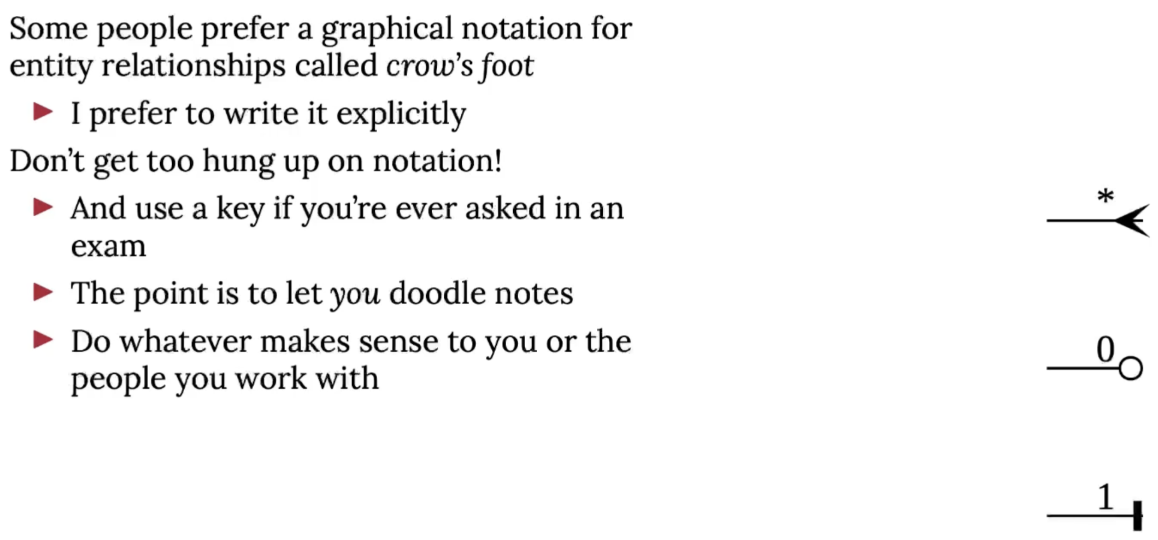


Now these entities can be related

- one student may take mutiple units, and one unit will have multiple 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

- top symbol means many (aka crows foot)
- middle 0
- bottom 1
- you can combine these things, e.g. if you had middle and bottom it could mean 0 or 1
  - 0 or crows foot == 0 or more
  - 1 and crows foot == 1 or more

The point of en entity relationship diagram is simply to doodle the relationships between entities

More additions…

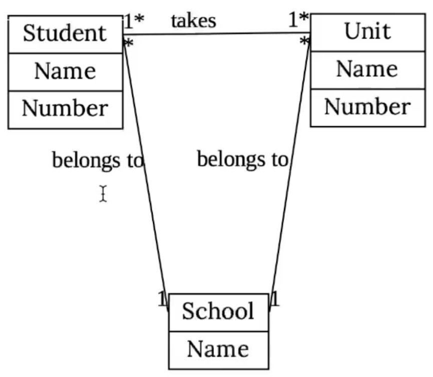
Schools are a thing!

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!



- * represents multiple

So say if multiple students have the same name or identifyer?

of if the saem student appeared in the school twice

- distinguishing between them would get confusing for a computer

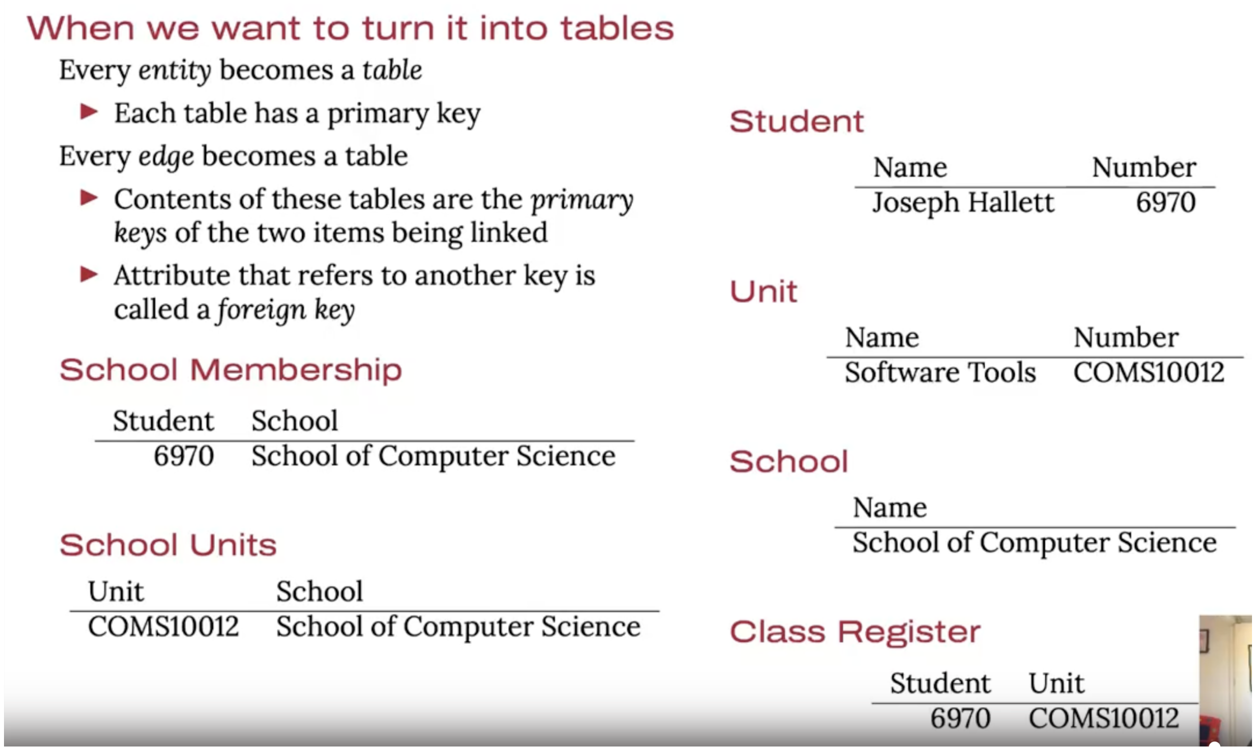
Entity Keys

- 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** is the key we actually use to refer to the object/entity
- if a key contains multiple attributes it is called a **Composite Primary Key**
  - e.g. number AND name are combined to make the unique primary key
- if a key is a meaningless ID column you added for the sake of **having a key** then it is called a **Surrogate Key**
- **Foreign Key -** used in SQL to link two tables together.
  - it is the column in one table that **references the primary key of another table**

    - purpose of foreign key is to ensure the referential integrity of the data
        - i.e. enforces a link between the data in two tables to ensure that the relationship between them **stays consistent**

So a key is just a way to uniquely referring to a specific entity inside another entity

- e.g. a specific student on a specific unit in a specific school



When we want to turn it into tables

Every *entity* becomes a *table*

- Each table has a primary key

Every *edge* becomes a table

- Contents of these tables are the *primary keys* of the two items being linked
- Attribute that refers to another key is called a *foreign key*

School Membership

Student	School
6970	School of Computer Science

School Units

Unit	School
COMS10012	School of Computer Science

Student

Name	Number
Joseph Hallett	6970

Unit

Name	Number
Software Tools	COMS10012

School

Name
School of Computer Science

Class Register

Student	Unit
6970	COMS10012

- each entity becomes a table

Key Constraints

these rules that help ensure data integrity and define the relationships between tables. They specify the unique characteristics of a column (or columns) and enforce restrictions on the data that can be stored in tables. There are several types of key constraints in SQL databases, including:

1. **Primary Key Constraint**: Ensures that each row in a table has a unique identifier. No two rows can have the same value in the primary key column(s), and a primary key column cannot contain `NULL` values. This constraint is crucial for uniquely identifying each record in a table.
2. **Foreign Key Constraint**: Used to establish a link between two tables. It is a column (or columns) in one table that refers to the primary key columns in another table. The foreign key ensures the referential integrity of the data, meaning that any value in the foreign key column(s) must exist in the corresponding primary key of the referenced table or be `NULL`.
3. **Unique Key Constraint**: Similar to the primary key, but it allows for one `NULL` value. It ensures that all values in the column(s) it is applied to are unique across the table. A table can have multiple unique constraints.
4. **Check Constraint**: Specifies a condition that must be true for all rows in a table. For example, a check constraint can enforce that a column's value must fall within a certain range.
5. **Composite Key Constraint**: A combination of two or more columns in a table that can be used to uniquely identify each row in the table. The combination of values in these columns must be unique across all rows in the table.

These constraints are critical for maintaining the accuracy and reliability of the data within relational databases. They help prevent invalid data entries and ensure the consistency of relationships across different tables, thereby supporting the integrity and trustworthiness of the database.