Relational Databases

**Introduction**

***“A database is a persistent organised store of related data”***

Learn this definition!

Persistent – Permanent

Organised – Data stored in records and fields

This could be in a single table or a collection of tables. Most programs will store data using a database. It is therefore important that we can design and program databases.

In this topic we will look at relational database design, normalisation and how SQL can be used to program and access databases.

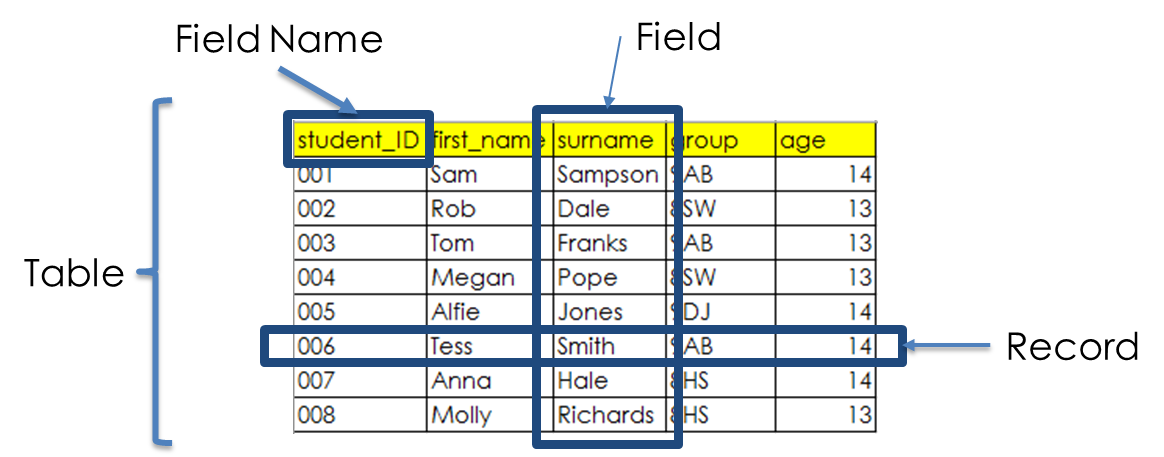
**Important Database Terminology**

TABLE: Collection of data that relates to an entity (e.g. students)

RECORD: A collection of data about a single entity (e.g. a student)

FIELD: A unique piece of data about an entity (student surnames)

FIELD NAME: An identifier for the single piece of data (e.g. ‘surnames’).



**Entity and Attributes – More DB Terminology**

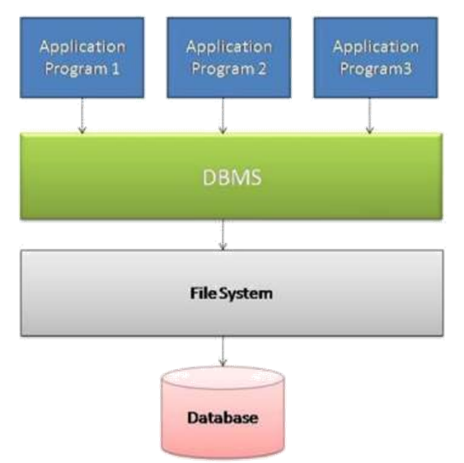
When dealing with databases we often talk about entities and attributes.

An entity is something in the real world that we store data about. A person, a car, an order or an exam subject are all examples of entities. An entity becomes a table in a database. One instance of an entity in a database is a record.

An attribute is a characteristic of an entity. It becomes a field in a database. For example, if the entity was a person, an attribute might be their surname.

**Data Handling Software**

A database on its own is pretty useless. For a database to have any real use, there needs to be software to bring the database to life. This software might be a dedicated ‘data processing’ program used to access customer details. Or it could be, for example, a ‘computer game’ which needs to update the game’s high score table. Either way, software needs to accompany a database to enable the database to be created and data in it retrieved, updated and deleted (CRUD).



**DBMS**

In fact, applications that access databases will all do so using a special ‘intermediate program’ called a Database Management System (DBMS).

The whole point of a DBMS is to separate the database from the application(s) that use the database.

**Why is a DBMS needed?**

The reasons for the need to separate the database from the program are as follows:

1. The DBMS can be thought of as a guardian of the data and any application that needs to access the data has to do so through the guardian, which prevents applications from handling the data wrongly, which could compromise the integrity of the database.
2. It also means that if the code of programs are edited, programmers don’t need to worry about compromising the data as it is stored elsewhere.

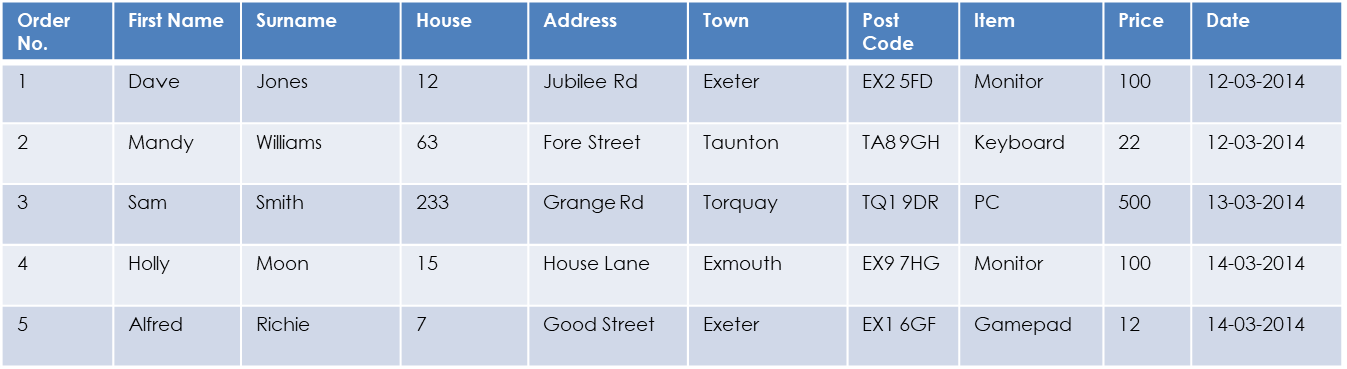
**Features of a DBMS**

* Allows different applications to access the data at the same time.
* It controls access to the data. Security features are provided to limit who can do what.
* It provides backups and the ability to restore from a backup if a disaster occurs.
* It supports a query language and other languages which can be used to extract, add and amend data and to alter the structure of the database.
* It can enforce ‘referential integrity’. This means that it prevents any operation that could damage the relationships between tables of data. For example, deleting an exam subject would not be possible if there were students linked to that subject.
* It controls ‘concurrency’. This means that it can lock data while someone is working on it so that someone else cannot change it at the same time.

As said above, DBMS supports a query language and this is one of the most important features. A query language is basically the commands that an application will use to ‘talk to’ a database. An application will use a query language (often SQL) to ‘Create’, ‘Retrieve’, ‘Update’ and ‘Delete’ data. And when it comes to ‘Retrieving’ data, an application is really ‘Querying’ the database. ‘Querying’ a database means ‘asking it questions’ to find the required data and this is at the heart of ‘working with databases’.

**Flat File & Relational Databases**

As stated earlier, a database can be a single table of data or a collection of tables and in terms of effective database design, the decision to have either a single table or multiple tables depends on the number of entities that the data is about.

Here is an example of a computer shop sales FLAT FILE database:

**Multiple Entities in a Single Table & Data Redundancy**

The issue with this database table is that it contains data about multiple entities:

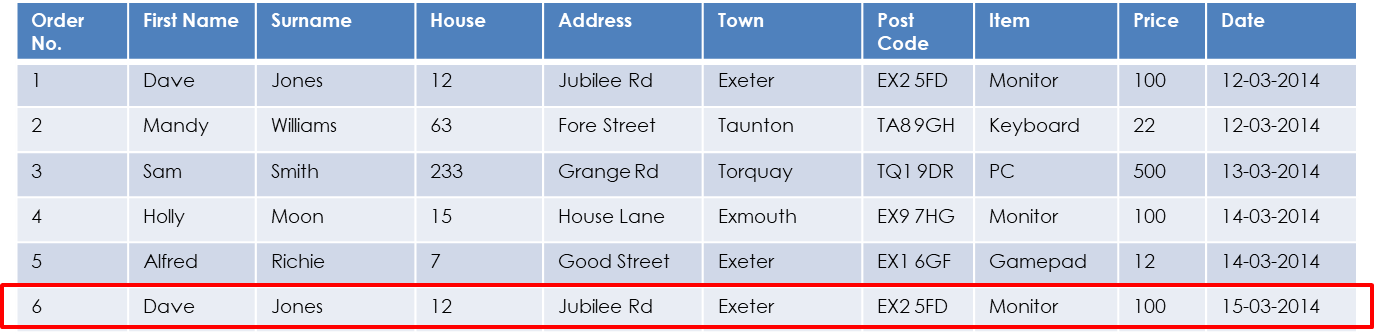
* Customers
* Products

Why is this an issue?

If a new sale occurred, a new record would need to be added to the table and in this record, ALL the data about the person and sale item, would have to be added. This is an issue for the following reasons:

1. It takes time to repeatedly enter the same data into a database
2. Humans are prone to making errors. If they have to enter the same data over and over again, they may type in some data wrongly thus making that data unsearchable.
3. If an update was made to the customer’s data, all records would need to be individually updated.

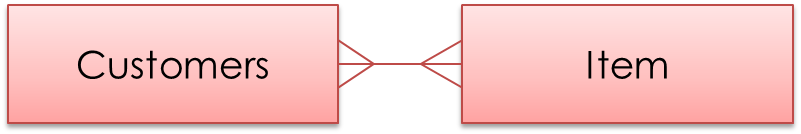
All in all this is not efficient!



**Entity Relationships and Database Design**

The entities in this database have a many to many relationship, which means:

* A Customer can buy many items.
* An item, can have many different customers.

This is shown in the following ‘Entity Relationship Diagram’ (ERD):

Many-to-Many relationships are always an issue in a flat file database, because they generally require the need to duplicate data over and over again, leading to data redundancy. They are incredibly inefficient.

**Intersection / Junction Tables**

The way around this issue is to make three different tables of data:

* A ‘Customers’ table
* A ‘Items’ table
* An ‘Orders’ table (Intersection /Junction Table)

…and link them together.

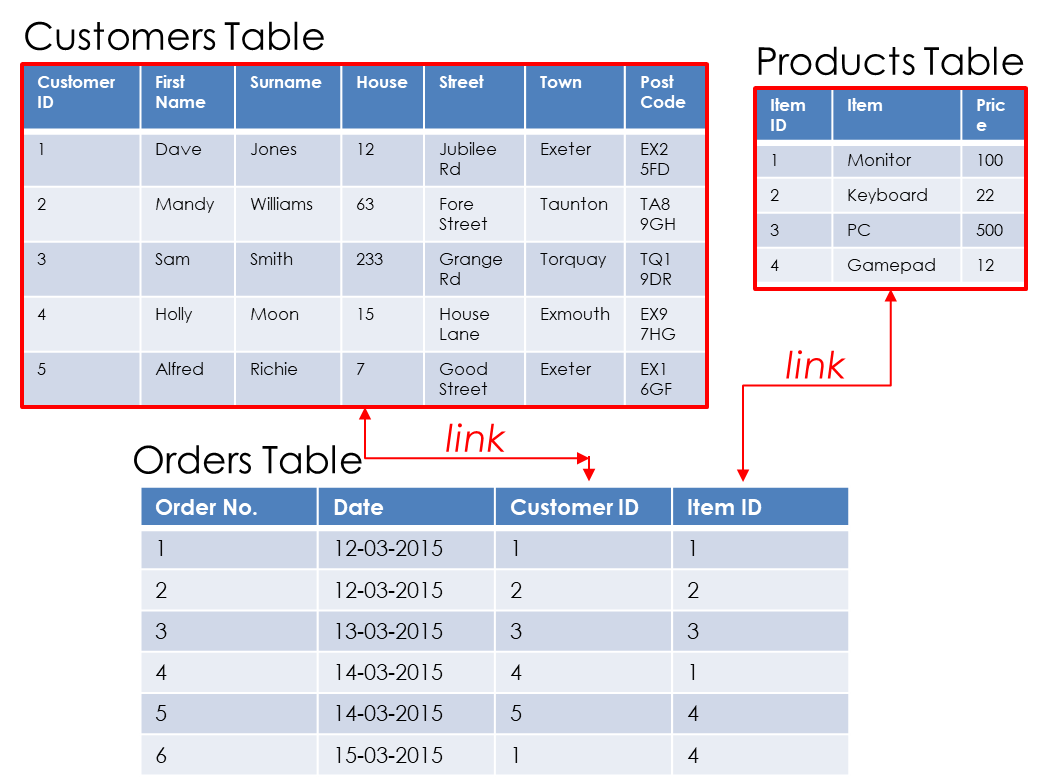
With this set up, the customer details and the product details need only be entered once, in their own separate table.

Each time an order occurs all we do is link the appropriate customer (from the customer table) and the appropriate item (from the items table) to a record in the orders table.

If the same customer makes another order, their details are already stored and so by linking them to a new order record means that we don’t need to keep entering the same data about the customer for each sale therefore there is no time wasted entering duplicate date, reducing the chance of introducing data errors and improving efficiency if a customer or product detail needs updating.

**Relational Database Design**

Here is an example of the relational database, which contains 3 tables:



As you can see, the ‘Orders Table’ links the other two tables by containing 2 extra fields to house a copy of the ID columns from each of the other two tables.

This creates a link to the customer and the product for each order created.

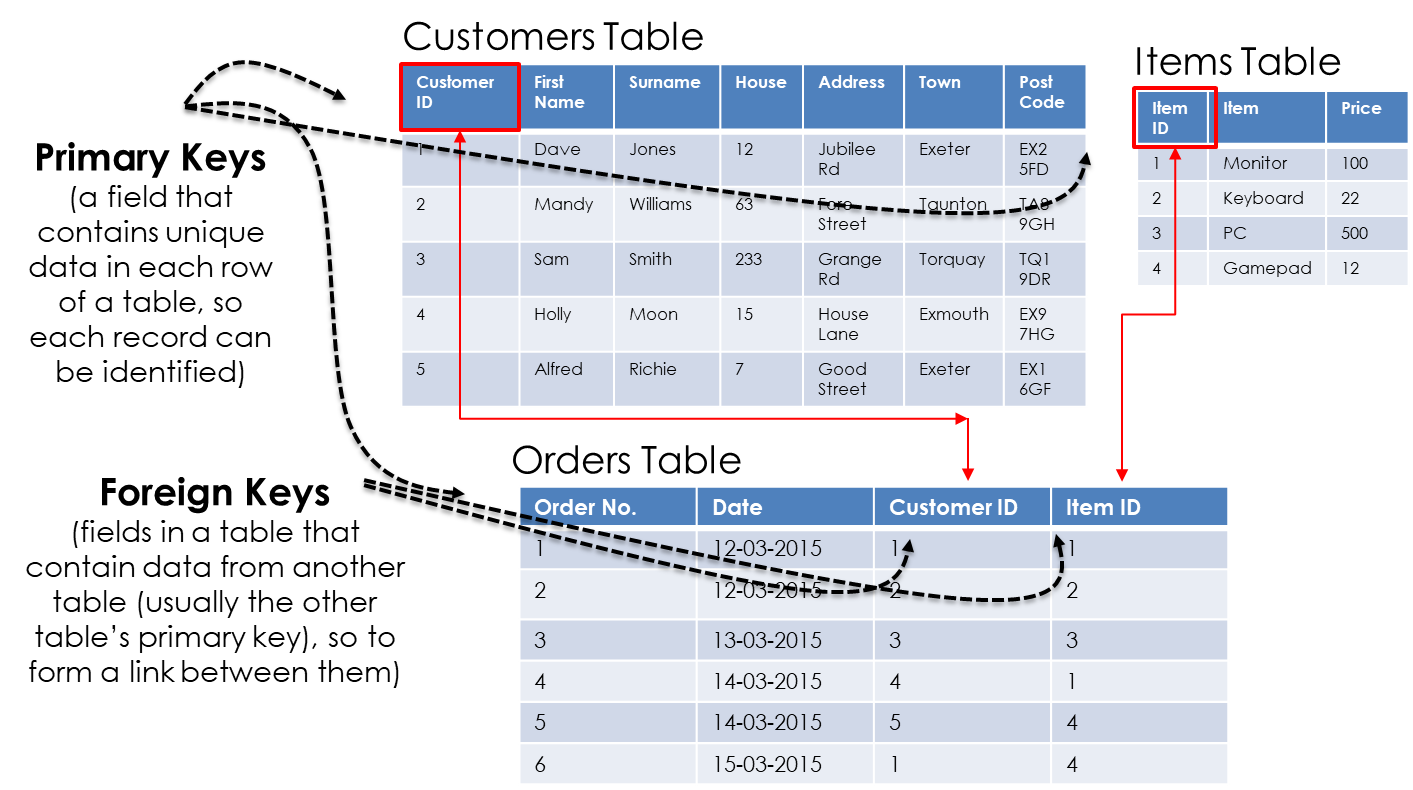
**Key Fields**

To ensure that the relationships / links between tables are fully functional and effective, each table will need to have some key fields.

A **primary key** is field which can be used to uniquely identify each record of the table. In other words, a primary key must contain data which is different in every record in the table. This is usually best achieved using an ID number.

Where an entity has more than one field that could be used as a unique identifier, if one is taken as the primary key, the others are known as **secondary keys**.

**Foreign keys** are attributes of an entity (fields of a table) that link to another entity (table). In other words, a foreign key is a primary key from another table .



**Queries**

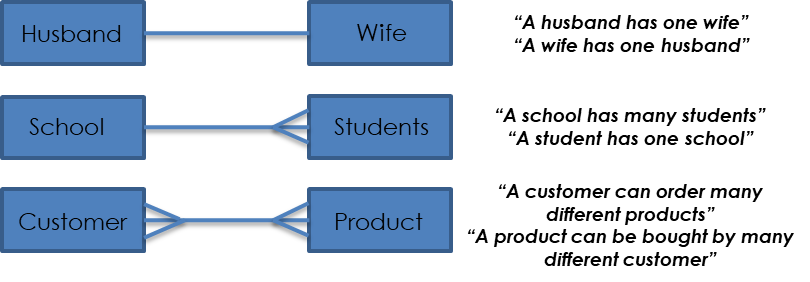
Although, the data about a particular sale is stored in several tables, because they all have a relationship (are all ‘linked up’) it means that the computer treats it like one big table. Most of the time, a database isn’t just about storing data, it is instead about using the stored data to find things out. Searching for information from a database is called a Querying. Because of the relationships, we can easily run a query and find the related data as if it were one big table.

**Entity Relationships Diagrams**

At a database design level, to avoid the issues discussed on the previous slides, it is always important to think about what entities the database is about and study their relationships.

This is often done visually using ’Entity Relationship Diagrams’ (ERDs).

Entities may have the following relationships:

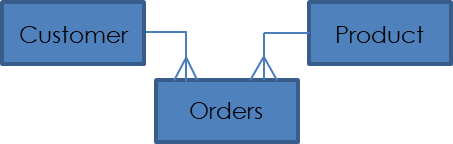


One-to-One

One-to-Many

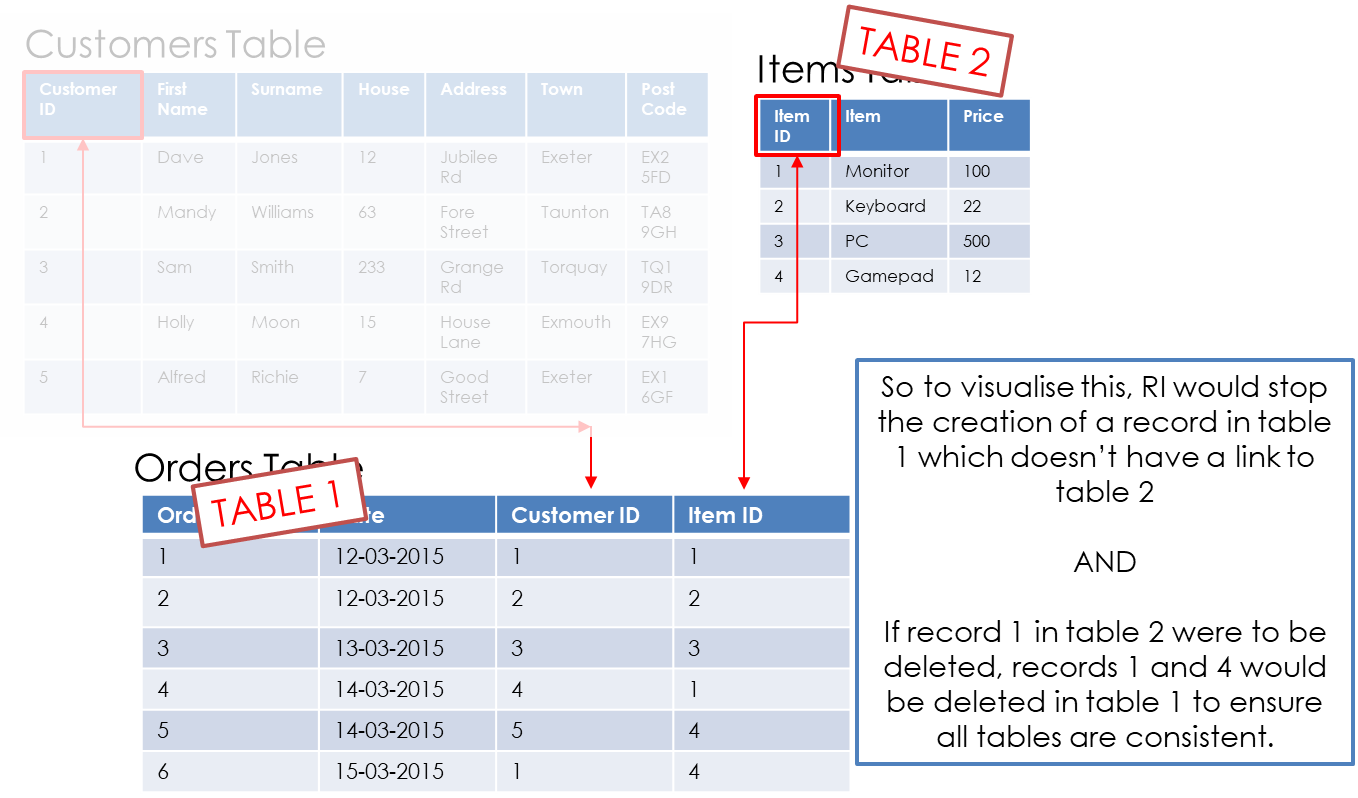
Many-to-Many

…and as we have already seen, many-to-many relationships will not work in a database and so they will always need an intersection table to satisfy their relationship in a functioning database:



**Referential Integrity**

In a relational database, as you have already seen, there will most probably be a host of tables which are all linked in some way. It is therefore very important that a given element in one of these tables, that links to another, is not deleted, as it will cause a break in the relationship and cause problems for the functionality of the database.

Referential Integrity is a concept that relates to relational databases that states that all primary key values match those of the foreign key to which it is directly related. In other words, if table 1 contained a field (foreign key) that links to a field in table 2 (primary key), referential integrity would stop the addition of a new record in table 1 which cannot be linked to table 2. Furthermore, if you wished to deleted a record in table 2, all records in table 1 that are linked to this deleted record would also be deleted.

***Keywords / Key Terms:***

**Table:** A table is simply a collection of data that relates to a person or object (often referred to as an entity (e.g. students)).

**Record:** A collection of data about a single entity (e.g. a student).

**Field:** A unique piece of data about an entity (student surnames).