The purpose of is to briefly familiarise yourself with some of the database jargon that is used when talking about SQL and databases, and are usually the terms used by database designers.

For example, when we normally speak about data in a tabular form, like the data stored in excel we usually refer to fields and records. Fields being the columns and records being the rows.

Also in this section we will very briefly discuss how databases are modelled using business cases or use cases.

The table below will show the accepted terms used by everyone in the work place to what they could be referred to when speaking in terms of relation databases.

Worksheets, Group of tables | Tablespace, Schema

Table, File | Entity, Table, Relation (not to be confused with relationship)

Extraction ,Sampled records | Table Page

Field, Column | Column, Attribute

Facts about a certain topic | Data elements

Range of values | Domain

Character field or Numeric Field | Data type

Unique numbering | Primary key

Records, Rows | Rows, Tuples

Type of relationship between tables as well as participation | Degree

Amount of unique values compared to row count | Cardinality

Column A is before column B | Ordinal Position

Field 1 combined with Field 2 to become unique not related to another source | Candidate Key

Field 1 combined with Field 2 to become unique but related to another source | Compound Key

A unique number that exist in the real world | Natural Key (Number plates of cars, barcodes on products)

Value that cannot be split any further | Atomic Value

Data modelling and table Relationships

These technique described here will be a basic demonstration of how the relational model is

conceptualised.

Although data modelling is no part of this document scope, it will be used to give you an understanding of how relationships are formed between tables.

These type of relationships are important to understand for <a href="#Joining\_tables>Joining tables</a>.

There are 3 main types of table relationships:

1 to 1

1 to many

many to many

There is also a 4th type of relationship, it is the recursive relationship and the reason it is not part of the main 3 is because these 3 are usually between different tables and the recursive relationship occurs when a table has a relationship with itself in a kind of parent child configuration.

We will use some stories from a used car sales business.

A used car sales business named 123 Sold (pty) ltd, has many departments. Each department is managed by one person, this person appointed as manager can only be in charge of one department.

In the sales department, a salesman can advertise and market any car. Any car may also be advertised by any salesman.

123 Sold is the holding company of a body shop next door named So Custom Body and Aftermarket Parts (pty) Ltd. The body shop uses many different colors and shades, in many cases cars are painted with different colors. A car always will have a base color, but its panels can be painted in different colors, for example. The bonnet car be yellow, the roof can be green and trunk lid can be orange.

So Custom Body and Aftermarket Parts must keep track of which colors are associated with which cars.

These user stories are what set the requirements for a data structure. But how do you create a data structure from the above?

Without diving too deep looking at the above. Nouns and verbs translate into tables or entities and characteristic's normally translate into columns. But what about the relationships?

To identify the relationships you must first identify the entities, the players this can be either a person, an asset or an action. Entities are almost always something that can be represented in the real world.

Referring back to the user story:

1. A **used car sales bus****iness** named 123 Sold (pty) ltd, has many **departments**.

"Sales business" and "departments" are the entities, Company and Departments.

Company has an attribute of "name", and we can assume that departments would have a name as well.

"has many" indicates the interaction and if you have experience working you will know that a company has many departments.

From this we can model this business case:

So this is what a data structurer would look like for this business case. It is a 1 to many relationship because a company can have many departments but a department can only belong to 1 company.

In the image above, crows foot notation was used for modeling the business case. To find out more for about erd modelling notation <a href="<https://www.icloud.com/pages/0HtS5LAx3a6nYSvbfooFIlwDw#RDBMS_Concepts_%26_Semantics>" >follow this link</a>.

2. Each **department** is managed by **one person**, this person appointed as manager can only be in charge of one **department**.

Entities are manager and department.

Once again the assumption is made that department and manager has a name.

And the interaction is a 1 to 1 relationship.

From this we can model this business case:

This business case is a 1 to 1 relationship between.

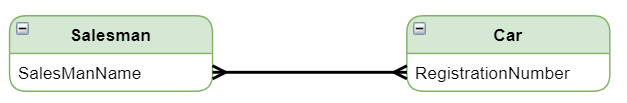
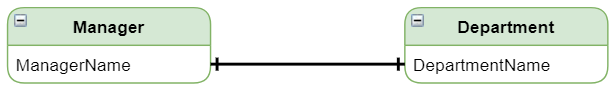
3. In the sales department, a **salesman** can advertise and market any **car**. Any **car** may also be advertised and marketed by any **salesman**.

The entities are salesman and car.

Although sales department is mentioned, this will just be to link to the department entity modeled above, but we will not model this just yet.

The interaction is a salesmen can market any car and any car can be marketed by any salesman.

This is a many to many relationship:

You can see that the crows foot is on both sides. Although we are able to model a many to many relationship, physically implementing implementing a many to many relationship is not a good idea. Many to many relationships are mostly theoretical and should be avoided.

To diffuse this situation implementing a <a href

="<http://databasemanagement.wikia.com/wiki/Database_Entities_in_E/R_Modeling>">composite entity<a/> is usually the order of the day as below:

By creating the composite entity **Advertisments** the many to many relationship is resolved and it is possible to know exactly which salesmen placed what add and for which cars adds have been placed.

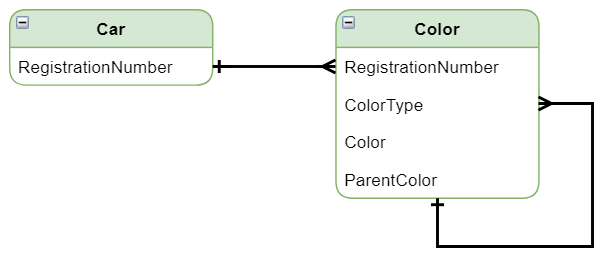
4. A car always will have a base color, but its panels can be painted in different colors.

There are various ways to model this problem, but the aim here is to demonstrate a recursive relationship.

There is only 1 entity, color. But will include, the car entity for this example.

Instead of trying to capture each panel as a column in Car, which is not a good idea to start off with because each car would not have the same number of panels. How would you know how many columns to create and furthermore what to name the columns as each car might not have the same type of panels.

With this design each panel can be named assigned to a car, a car can be assigned with as many panels as reburied and the solution is not an architectural conundrum.

But the point here is that you can see that the color table has a relationship with itself. This is what is known as a recursive relationship. The requirement states that a car must have a base color and then all other panel colors must be linked to this base color.

hence for a specific car, it would have a base color, and all the panels that are related to that car would be children of this base color.

From this we can also see that a fair amount of assumption is also required when doing data modeling.

Database designers are highly trained and experienced professionals, and are intune with what is required by a client. All applications start here.