An Introduction to Data Normalisation

Creating a Logical Data Model

Neville Turbit

Overview

First a disclaimer. This is not absolutely technically correct. The intention of this white paper is to give people, who have little idea of data normalisation, a view of what it is all about. It is the 101 of data normalisation.

Why do you need to understand data normalisation? Project Perfect do a lot of Microsoft Access development. When talking with clients the concept of data normalisation is often seen as a black art. We have to explain what it is all about, and involve the users in the process. That involves some understanding of data normalisation and logical data modelling to make their contribution meaningful.

A secondary target is the person setting out to build their first database be it Microsoft Access, Filemaker or any other database solution. Getting the data structure wrong means that no matter what else you do, the program is basically flawed. The most beautiful, functional building will be unstable if the foundations are faulty.

Data Organisation

Thinking in terms of a spreadsheet, imagine you want to store customer details. You might have columns for:

- Customer Number (unique number)
- First Name
- Surname
- Address Line 1
- Address Line 2
- Suburb
- Postcode
- Phone
- Mobile
- Email
- Company
- Notes

This works fine until you find someone who has a business and home phone number as well as a mobile. Not a problem as you add a "Business Phone Number" column.

Then you find that someone has a general company number and also a direct company number. Add another column called "Direct Business Number". So now you have columns for phone numbers:

- Phone
- Business Phone Number
- Direct Business Number
- Mobile

There is immediately a problem because prior to adding the two new business number columns, you had entered both private and business numbers in the original "Phone" column. You now have information that will either remain incorrect, or you will need to go back and reorganise the information.

The users of the system have decided to record fax numbers. As there is no field to do this, they put them in the "Notes" column. You now decide to add another column for "Fax". You have people go through every note, and put the fax number in the new column. What you forgot was that there are business fax numbers and private fax numbers.

You can see how the basic design decisions quickly lead to data error and rework. If you get it wrong at the start, you have to keep adding new columns and resorting the data.

Another issue we have not mentioned is the amount of blank space. Maybe 10% of people have a personal fax number. The spreadsheet is storing 90% blank space. This creates a massive spreadsheet with many cells blank.

Tables and Primary Keys

In a database, information is stored in tables. Think of a table as a single worksheet. Each row, or record, has to be uniquely identified. This is almost always a number. In a personal sense, we all have account numbers, and customer numbers and tax file numbers which identify us as unique records in some database. This unique identifier is called the "PRIMARY KEY". It is typically a number and usually sequentially generated. Customer 00001 is followed by 00002 and 00003.

There are two important principles for tables:

- **Space is important.** We want to minimise the size of tables – and hence speed up processing and reduce storage – by reducing blank fields.
- **Information should not be repeated.** This will be explained in the next section in the white paper but here is an example. If "Suburb" is a field (or column in our spreadsheet), we do not want to repeat the same suburb name for every person who lives in that suburb.

Relationships

Let us explore the concept of not repeating information. I mentioned suburb. Each suburb has a postcode. For every customer from the same suburb they will have the same postcode. The example I will use is from Australia, but you can translate the example to your zip code or postcode.

The company address for Project Perfect is Abbotsford. Postcode is 2046. I will ignore the state to simplify the example. Imagine we create another table (spreadsheet) where we list all the suburbs and their postcodes.

Suburb ID	Suburb Name	Suburb Postcode
1	Abbotsford	2046
2	North Sydney	2059

The suburb ID is a sequential number that identifies the record. It is the primary key for the suburb table.

If I go back to my original example, I had the following fields or columns.

- Suburb
- Postcode

If I store the full name of each suburb and the postcode for each customer, it takes up a lot of room. I can replace both with a new field (column) called “Suburb Number”. If the customer is from Abbotsford I put in “1”. If they are from North Sydney I put in “2”. I don’t need to store postcode for every person. The program can look up suburb number and extract the postcode from the Suburb table.

Another advantage is consistency. If I enter the suburb name in full for every customer, there are likely to be inconsistencies and typos. North Sydney may be Nth Sydney, Nth. Sydney, North Syd. Etc. If I then want to find all customers in North Sydney there is no way to pick up all the abbreviations and misspellings.

If I use another table with suburbs – each of which has a unique number - I just search for all addresses that have a Suburb Number of 2. The number in the customer table that refers to the suburb table is called the “**FOREIGN KEY**”. The foreign key points to the primary key in another table. The foreign key is a cross reference to another table.

Database Conventions

Having explained tables and fields and how they are similar to spreadsheets and columns, we will introduce some naming conventions. These should be used in programs but less experienced programmers sometimes ignore them. If you are really interested look up Leszynski Naming Conventions on Google.

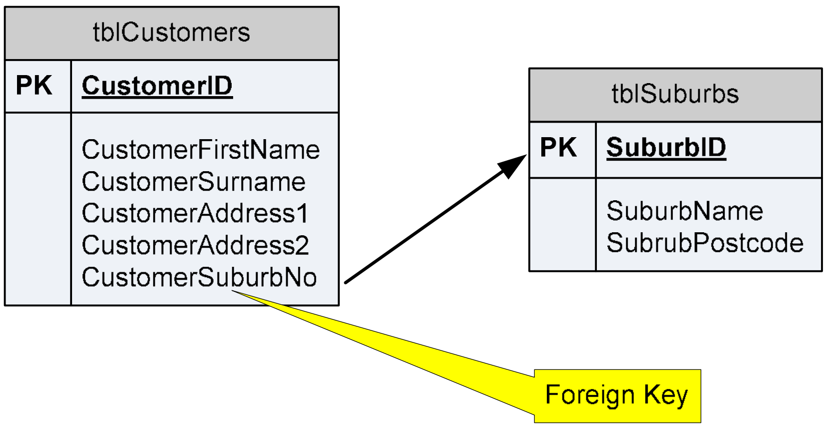
- **Tables start with “tbl”.** There are no spaces in the name, and each new word is started with a capital (Camel Case). For example, the two tables we have spoken about could be tblCustomers and tblSuburbs. I also use the plural for table names – “tblCustomers” rather than” tblCustomer”.
- **Fields start with the table name (without the “tbl”), are singular and there are no spaces. Capitalisation is used.** For example CustomerNo, CustomerSuburbNo, SuburbPostcode, SuburbName etc.

This may not seem important initially, but when you start writing programs it makes it much easier to work with consistent terminology. There are fewer errors, and it is easier to understand.

Simple Example

Here graphically is an example of our database for the relationship between tblCustomers and tblSuburbs. You can see that tblCustomers.CustomerSuburbNo is a foreign key. In other words, it is the cross reference to tblSuburbs that links the suburb to this customer.

In this example there is a primary key (unique identifier) for tblCustomers called tblCustomers.CustomerNo and a primary key for tblSuburbs called tblSuburbs.SuburbID.



Another thing to notice is the one to many relationship.

- One customer can only have one suburb.
- One suburb can be assigned to many customers.

This is important when we get to looking at entities later in the white paper.

Imagine we have a number of customers who live in the same suburb. The suburb is called "Swamp Flats". All the residents constantly get jibes about the name of the suburb so they get the government to agree to change the name to "Beauty Heights".

If we didn't have a separate table for suburbs, we would need to go through every customer record, find all the ones in "Swamp Flats" and change them to "Beauty Heights". We might write a query to list all the "Swamp Flats" suburbs but suppose there were some input incorrectly as "Swampy Flats" or "Swamp Flat". We would miss them.

If we have a tblSuburbs, we can just change one record. We change the record relating to "Swamp Flats" and every related customer record is correct. This is the value of splitting out the suburbs into a separate table.

Entities, Attributes and Relationships

Now we get down to creating a normalised data structure. Normalised means we split all the data up into separate tables with relationships between them. There are three terms we need to define. In fact we have covered them but to clarify the terminology we will define them again.

- **Entities.** The people, places and things we want to keep track of. For the purpose of this white paper, we can assume they will end up as tables e.g. Suburbs will end up as tblSuburbs.
- **Attributes.** The individual pieces of information we want to track for each entity. These are the fields within each table. For the entity customers, we want to know their first name, surname, address etc.

- **Relationships.** How the entities relate to one another. For example how customers relate to suburbs.

You might also hear the terms physical data model and logical data model. A logical data model is a representation of the entities, attributes and relationships developed by non technical people to illustrate what they want to keep track of. It is translated into a physical data model by a database administrator.

The physical data model consists of tables which evolve from the logical model, but may be modified to make the model more efficient. There may be other tables included that are not relevant to the user. For example, there may be tables for transaction logs or system access records. We will only deal with the logical data model. Leave the creation of the physical model to the database administrator.

Creating a Logical Data Model

There is another layer of complexity in our simple example that we will now explore. The examples we have used are relationships between customers and suburbs. We will now expand that example.

First the Entities

If you are doing this as a real life example, I suggest you put each entity on a separate sheet of paper. I often pin them up around a room if doing this in a workshop situation. If you are just doing it yourself, perhaps Post It notes will suffice.

Ask the question “What are the people places and things we want to keep track of?”

- **Customers** obviously
- **Addresses**. Since customers can have multiple addresses (home, business etc.) list it as a separate entity.
- **Suburbs**. The same suburb will appear in multiple customer addresses.
- **Phone Numbers.** Same as addresses. Customers can have multiple phone numbers.
- **Phone Types.** Different types of phone number such as home, business, business direct, fax, etc.
- **Notes.** Each customer can have multiple notes. This is a decision you need to take based on the actual situation. Do you want multiple notes for each customer, or only one note? If it is only one note, make notes an attribute of Customers.
- **Companies.** Several customers may work for one company.

We now have seven entities. As you are preparing the single sheet for each, you may want to jot down some of the attributes although it is not absolutely necessary to list every attribute at this stage.

Second the Relationships

Put the seven sheets of paper in line. Start by looking at 1 and 2 which is “Customers” and “Addresses”. Ask the following questions.

> “Is there a direct relationship between “Customers” and “Addresses? “

To explain a direct relationship, there is a clear relationship between “Customers” and “Addresses” but not between “Customers” and “Suburbs”. “Customers” have “Addresses” and “Addresses” have “Suburbs”, which is an indirect link.

If there is an indirect link or no link at all, move to looking at 1 and 3. You will work from 1 to 7, then start again between 2 and 3 up to 2 and 7 until you finally get to compare 6 and 7. All comparisons have been made.

- Customers and Addresses
- Customers and Suburbs
- Customers and Phone Numbers
- Customers and Phone Types
- Customers and Notes
- Customers and Companies

Then

- Addresses and Suburbs
- Addresses and Phone Numbers
- Addresses and Phone Types
- Addresses and Notes
- Addresses and Companies

Then

- Suburbs and Phone Numbers
- etc

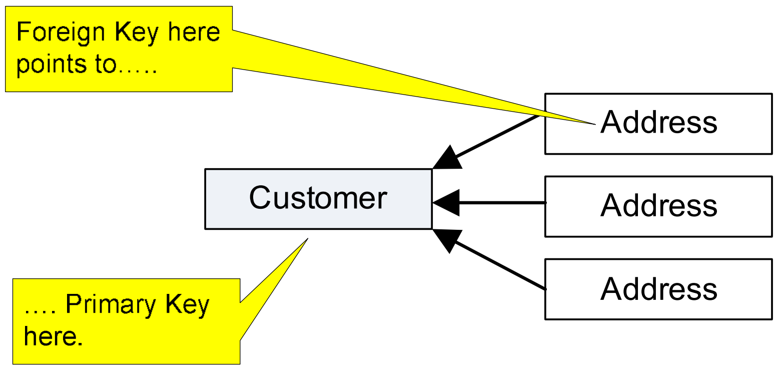
In the case of 1 and 2 ("Customers" and "Addresses") there is a direct link. We now need to ask two questions.

- "Can one customer have more than one address?" The answer is "Yes".
- "Can one address have more than one customer?" I am going to cheat a bit here and say "No".
  I know that it would be possible to have two people with the same address but it is getting too complicated for this example so I will assume the answer is "No".

If the answers are "Yes" and "No", you have a one-to-many relationship. You add a foreign key attribute to one of the entities. The question is which one?

I have heard a number of techniques described to work out which one has the foreign key, but this is my way. Think which entity will be the many side of the equation. Will one customer have many addresses or will one address have many customers?

The answer is one customer will have many addresses. Each address needs to be identified as belonging to a customer. In other words, you need a foreign key in the address record that says I belong to customer X.



If the answer is "No" and "No" you are saying that the first entity can only have one of the second entity, and the second entity can only have one of the first entity.

To give an example, let's look at Customers and Notes. We may decide that a customer can only have one note, and one note can only relate to one customer. What we have established is that "Notes" is an attribute of "Customers". Throw away the "Notes" entity sheet, and record "Notes" as an attribute on the "Customers" sheet. You only need to record one note per customer.

The third situation is the "Yes/Yes" answer. We do not have a situation like that in the simple example. Let us imagine we had another entity called "Skills". We want to see what skills apply to each customer. Perhaps we are a recruitment company and want to track the skills of customers.

- Can one customer have more than one skill? The answer is yes.
- Can one skill apply to more than one customer? Again, the answer is yes.

We now need to create a new entity called "Customer/Skills". The entity will have two foreign keys. One is for "Customer" and one for "Skill". Customer 1 may have records relating to skills 3, 9 and 11.

CustomerSkillNo **Primary Key**	**CustomerSkillCustomerNo** **Foreign Key**	**CustomerSkillSkillNo** **Foreign Key**
1	1	3
2	1	9
3	1	11

Create a new sheet of paper. Since this is a special entity to link two existing entities, it does not have to be matched.
In the process, you may decide to add an entity. In the example we decide we do not want to repeat salutations. We will create an entity called "Salutation". It will have values such as "Mr", "Mrs", "Ms", "Doc" etc.

Thirdly the Attributes

Once you have completed all the comparisons of "Entities" and the relationships are defined, fill in the attributes. It is rare to get them all listed at this stage, but you can add attributes later as you build your database. The main thing is to get the "Entities" and "Relationships" sorted at this stage.

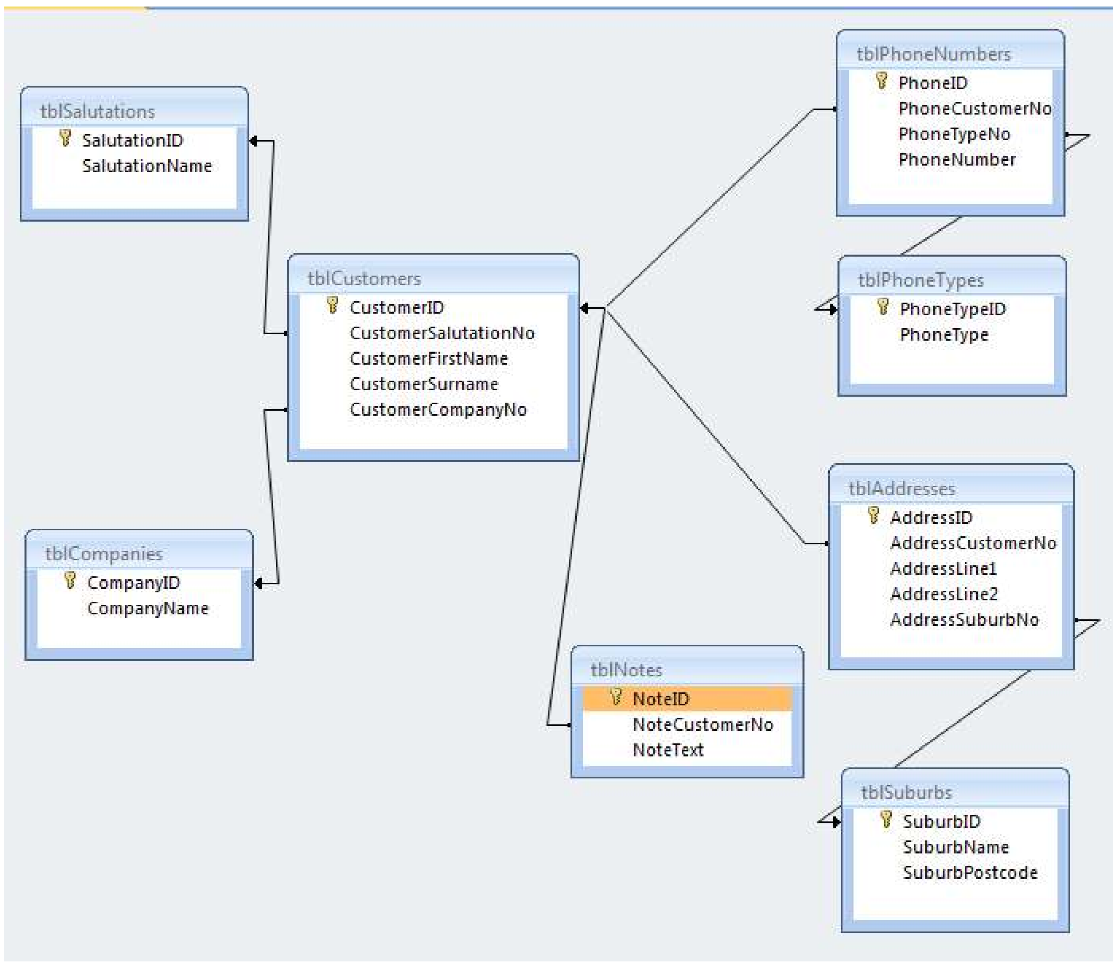
The Result

So after we have put the results together, what does it look like? Imagine the diagram below is each piece of paper arranged in a diagram. I have used the table name rather than the entity name ("tblCustomers" rather than "Customers") to show how entities match tables. The fields with the keys beside the name are the Primary Keys. The arrows point from the Foreign Keys to the relevant Primary Key.

In undertaking the model, we start with entities of Customers, Salutations, Companies etc. We ask the question:

- "Is there a direct relationship between Salutations and Customers?" – "Yes"
- "Can one salutation apply to more than one customer?" – "Yes"
- "Can one customer have more than one salutation?" – "No"

Now we have a relationship. We add a foreign key in Customers to enable us to determine the salutation applies to each customer record.



To illustrate how the relationships would work in a database, if you look at the top left we have tblSalutations and tblCustomers. The tblCustomers has a Foreign Key called CustomerSalutationNo which points to the SalutationID field in tblSalutations. Imagine tblSalutations had records

SalutationID	SalutationName
1	Mr
2	Mrs
3	Ms

Imagine that tblCustomers had the following records:

Customer ID	**Customer SalutationNo**	**CustomerFirst Name**	**Customer Surname**	**Customer Company**
1	1	John	Smith	5
2	3	Jane	Doe	
3	1	Bill	Jones	11

You can see that the customer records with CustomerID 1 and 3 have the salutation "Mr". CustomerSalutationNo is 1 which in tblSalutations is "Mr". Record 2 has the salutation "Ms" as the CustomerSalutationNo is 3.

The Difficult Questions

There are a number of questions that may arise. Mostly they can be answered by applying some common sense and remembering the two initial goals of setting up a data structure:

- Space is important
- Information should not be repeated

Unfortunately sometimes the question raises a contradiction. You might do something that stops information being repeated, but takes up a lot of space.

Here are some questions:

- In a few cases, the same postcode can apply to two or three suburbs. Should we have an entity for suburbs and another for postcodes? In a purist form the answer may be yes, but to be pragmatic, if less than 5% of postcodes apply to more than one suburb, I would say forget two entities. You would need to add an additional field to the "Suburbs" entity to hold a foreign key to "Postcodes" so it is more space, and more complication. I would not create two entities.
- We have a separate entity for "Companies". Is it worth recording it as a separate entity if there are few customers working for the same company?
  My response would be to ask what happens if a company changes name? Do you want to go through each customer and change each customer individually or do you want to change one "Company" record?
  Do you ever want to search for people who work for a particular company? If so, it is easier to look for customers with "CustomerCompanyNo" = X. Once again it depends on what exactly you are doing with the database.
- Should you have another entity called "Address Types"? These may be home, business, delivery etc. I would say yes. I didn't include it in the example for the sake of simplicity but in a real situation I would have a separate entity to hold "Address Type". Think about what you might want to see in a drop down list. If you think you would like a list of something, it will probably be an entity.

Conclusion

We have explained entities, attributes and relationships. We have covered why you need to normalise a data structure. We have also provided a technique you can apply with users, or as a user to put together a logical data model. Getting the data structure right is critical to building a solid working application. If you get the structure wrong, you will certainly find limitations in what you are trying to do.

It can be a tedious process to build a logical data model, but it needs to be done. In the process, you will be faced with lots of decisions as to how you want the system to operate.

- Do you want to be able to record lots of notes, or will one note do?
- Do you want to record salutations in your application?
- We set up a place to store postcodes but have not looked at country. Do you want to record county?
- Will you have customers outside your own country?

All these questions will be discussed as the model evolves. In the process you need to document the decisions you take, and also raise issues to be resolved later. Just like building a house, it is tedious to get the foundations in place, and nothing seems to be happening for a long time. The problem is that if the foundations are wrong, the end result will always be a disappointment.

The Author

Neville Turbit has over 20 years experience as a Project Management and IT consultant and almost an equal time working in Business. He is the principal of Project Perfect. Neville can be contacted at turbit@projectperfect.com.au

About Project Perfect

Project Perfect is a project management software and consulting organisation based in Sydney Australia. Project Perfect sell “Project Administrator” software, which is a tool to assist organisations better manage project risks, issues, budgets, scope, documentation planning and scheduling. They also sell a complete web based methodology for software package selection. In addition they are specialists in Microsoft Access development.