An Approach to ER and Designing Database Tables

© Mark O'Reilly, Last Updated: 2016

Detailed below is a recipe for creating Entity Relationship diagrams for designing database tables. The process is offered as a step-wise introductory approach and may require more formal refinement as your experience with database design proceeds. The steps would be generally addressed in the order given, although you may need to return to previous steps to make adjustments as you proceed with your design. You may need to loop through part or all of the process several times.

The suggested steps are:

1	**Entities**	List any entities that are evident within the system.
2	**Relationships**	Identify the relationships between the entities. Include additional entities if and as they become apparent. **2.1** **Links**: Draw the entities with preliminary links on paper. **2.2** **Cardinality**: Check the cardinality between each linked pair of entities. **2.3** **Modality**: Check the modality associated with each relationship. **2.4** **Labels**: Add a label to each link or relationship.
3	**Attributes**	Add the various attributes to each entity. **3.1** **Identifying (Primary Key)**: First add the identifying attribute to each entity. **3.2** **Linking (Foreign Key)**: Based on the various relationships, add all attributes necessary to link the entities. **3.3** **Descriptive**: Add all other attributes.
4	**Lookup Lists**	Finally check the various descriptive attributes for possible 'lookup lists'.

Taking each of these steps in turn:

1. Entities

In this step, you need to identify the entities for your client's system, or are evident in the script provided. This step can generate some uncertainty - often an entity can be mistaken for an attribute (or vice versa), one or more entities can be missed, or too many entities may be noted. These uncertainties can be overcome with experience and with careful review of the client requirements or script.

In identifying entities, you initially need to look for **groups** or **collections** of items:

1. Groups of **people** - customers, clients, employees, suppliers, members etc.
2. Groups of **other living things** - pets, wildlife, plants, etc.
3. Groups of **items** - products, stock, parts, collections (books, videos, musical items, software, etc.)
4. Groups of **documents** - orders, invoices, membership forms, legal documents, notifications, memos, etc.
5. Groups of **transactions** - customer orders, loans from a video shop or library, services provided by a gardener, tradesperson, doctor, dentist, accountant, etc.
6. **Lists attached to any of the above** - tracks on a music item, lines or details on an order, sections in a document, ingredients on a recipe.

2 Relationships

This section has been broken into four distinct steps:

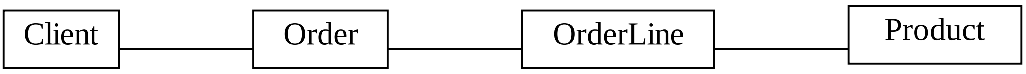
2.1 Relationships - Links

As with defining the entities, this step is made easier with a greater and clearer understanding of the client's requirements, and with experience. You need to ask relevant questions of your client, listen carefully to their needs, read and re-read the requirements, scope or script. Experience will assist you in becoming familiar with various patterns and applying these appropriately to new projects.

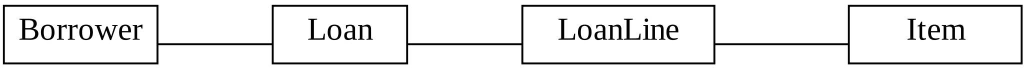
For instance:

If there is a standard business transaction that needs to be modelled, you might apply a standard 'orders' structure:

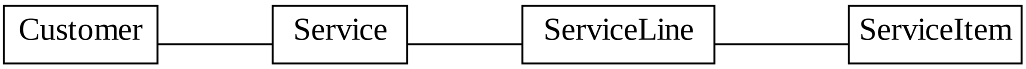
1. **Business selling goods**:



2. **Business or entity loaning items** (video shop, library):

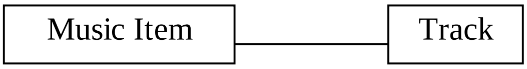


3. **Service Provider** (gardener, accountant, etc):

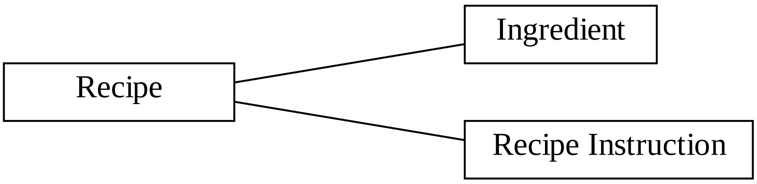


If there is a document or entity that contains a list of items that need to be tracked:

4. **Music Item**:

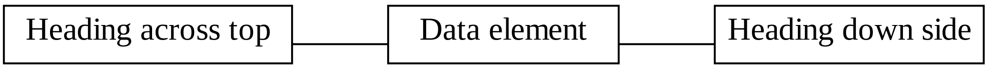


5. **Recipe**:



If there is a table with headings across the top, headings down the left hand side, and data in the middle:

6. **Data table**:



In some instances there can be uncertainty as to whether a link should be included or not. For instance, imagine we were wishing to track student assignment submissions for a given subject, and for a given teacher. It is possible for us to add links between the student and their assignment, the student and the subject, the student and the class, the teacher and the subject, the teacher and the assignment, and the teacher and the student submission. The question would have to be asked whether all these links are necessary, given the scope of the project and database required.

In such instances, asking relevant questions, listening carefully to the client, reading the script, and logically analysing a situation can often help with your clarity.

2.2 Relationships - Cardinality

As each of the steps of this procedure is addressed, one or more of the previous steps may need to be revisited. For instance, identifying the links (as we did in the previous step) may bring an additional entity to light. This may also be the case when defining the cardinality of the various links.

Cardinality helps define the nature of the link between two entities - whether they relate one-to-one, one-to-many, or many-to-many. The cardinality between some entities can be self evident, while others may have to be considered carefully.

For instance:

Music Items: you can see by looking at a music item that there are many tracks listed on a given record or CD. Therefore we can say that one music item has many tracks.

Recipes: you can see that one recipe has many ingredients. We can say that one recipe has many ingredients.

Orders: you may also note that a given order or invoice has many products listed for purchase. If I had overlooked the OrderLine entity, I might initially consider that one order has many products listed.

However on closer inspection, we might also note that one ingredient may appear on many recipes or one product can appear on many orders.

When identifying the cardinality of a link, it is suggested that you consider an example item for one entity and ask whether it relates to one or many of the other entry. Then apply the same question in the reverse direction. For instance:

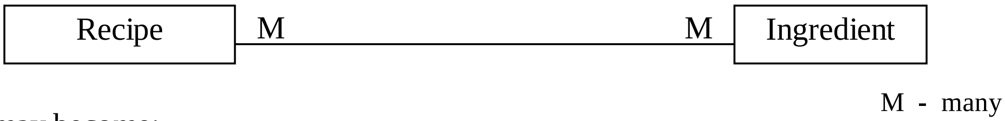
Recipes: I would picture one recipe sheet in my mind and ask how many ingredients *could* be listed on the recipe. My answer would be: *many*. I would then consider one ingredient (eg flour) and ask how many recipes this ingredient might appear on. The answer would also be: *many*. I would have realised that there is a many-to-many relationship between recipe and ingredient.

Orders: I would picture one order in my mind and ask how many products *could* be listed on the order. My answer would be: *many*. I would then consider one type of product from the business stock and ask how many orders this product might appear on. The answer would also be: *many*. I would accept that there is a many-to-many relationship between order and product.

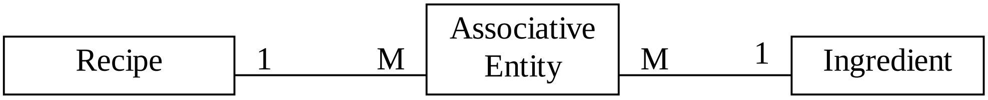
For ease of development, most many-to-many relationships can be simplified to two or more one to many relationships. A general rule of thumb can be applied in this instance, but significant care must be taken to check that the rule of thumb applies in the given instance.

Rule of Thumb 1: *If you identify a many to many relationship between two entities, place an associative entity between the two current entities, and check that the new links have become one-to-many.*

Reviewing the Recipe example:

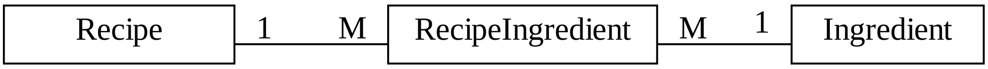


may become:



What is the associative entity? It is the ingredient listed on the recipe. The Ingredient entity represents the ingredient sitting in the kitchen cupboard - the physical ingredients themselves. The associative entity is the ingredient listed on the given recipe - the items listed on the recipe documents.

Which updates to:



Checking the cardinality of the new links:

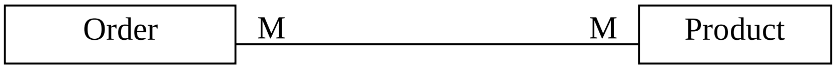
1. **One** Recipe has **many** RecipeIngredients listed on the recipe document - TRUE
2. **One** RecipeIngredient is listed on **one** Recipe - TRUE

This link is confirmed as: ONE Recipe has MANY RecipeIngredients

1. **One** Ingredient is represented as **many** RecipeIngredients on various recipes - TRUE
2. **One** RecipeIngredient (on one recipe) lists **one** Ingredient - TRUE

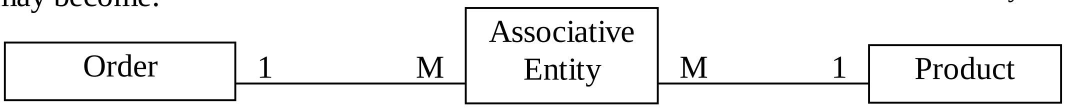
This link is also confirmed as to a one-to-many relationship: ONE Ingredient has MANY RecipeIngredients

Reviewing the Orders example:



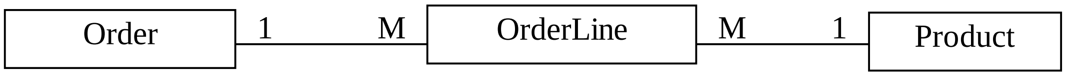
M - many

may become:



What is the associative entity? It is the product listed on the order. The Product entity represents the stock items sitting in the business store. The associative entity is the product listed on the individual orders, as purchased by a given customer.

Which updates to:



Note: *Sometimes OrderLines are called OrderDetails.*

Checking the cardinality of the new links:

1. **One** order has **many** OrderLines listed on the order document - TRUE
2. **One** OrderLine is listed on **one** Order - TRUE

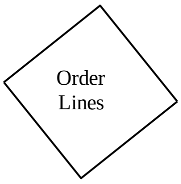
This link is confirmed as: ONE Order has MANY OrderLines

1. **One** product is represented on **many** OrderLines within various orders - TRUE
2. **One** OrderLine (on one order) lists only **one** Product - TRUE

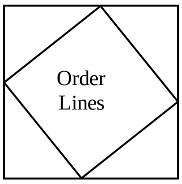
This link is confirmed as: ONE Product has MANY OrderLines

Caution: *The placement of an associative entity (as illustrated here) does not always work. It may be that the associative entity retains a many-to-many relationship with one or both of the existing entities. Apply the rule of thumb with due care and always check the cardinality of the new links. A more complex arrangement of the current entities may result or may be required.*

Note: *Associative Entities are commonly represented in diamond shaped symbols:*



Or…

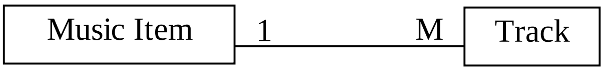


2.3 Relationships - Modality

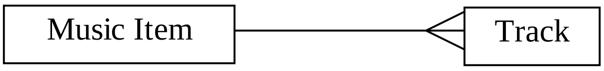
Once the cardinality of the various relationships has been established, a further characteristic can be identified - modality.

Return for a moment to the music items and tracks entities:

1. One music item has many tracks
2. One specific track is represented on only one music item *(this may not be perfectly correct, but let us assume this for the moment)*

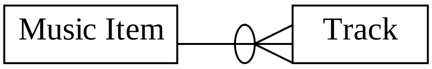


We could represent this using a slightly different notation:



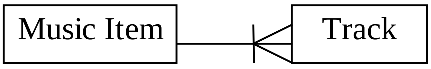
where the crows foot (—<) is used to represent the many side of the relationship.

And let us now assume we have created a database based on this Entity-Relationship diagram and we are beginning to enter data. How many tracks could we add for each music item? The answer is: many. But what would be the **minimum number of tracks** we would have to add for each music item? The answer could be zero, if we were not sure if we wanted to add tracks to the each of the music items; or one if we felt that each music item had to have at least one track. These two options might be represented as follows:



This diagram depicts that:

1. *One music item has many tracks.*
2. *Each music item can have zero or no tracks associated with it.*



This diagram depicts that:

1. *One music item has many tracks.*
2. *Each music item must have at least one track associated with it.*

A music item having a minimum of zero or one track associated with it are examples of **modality**.

Modality is usually determined in both directions for each relationship. For instance:

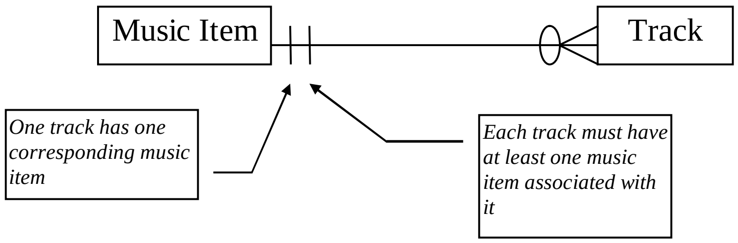
Modality of the Tracks entity based on its relationship with the Music Items entity:

1. One music item has many tracks.
2. Each music item can have no tracks associated with it.

Modality of the Music Items entity based on its relationship with the Tracks entity:

1. One track has one corresponding music item.
2. Each track must have at least one music item associated with it.

This might be depicted as follows:



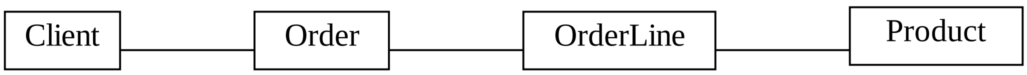
One track has one corresponding music item

Each track must have at least one music item associated with it

The (┼┼─) symbol might also be read or stated as: *one and only one*. For instance: each track has one and only one Music items associated with it.

The modality associated with a simple orders system might be derived as follows:

Note: *The following is a sample only and would not be applicable to all ordering systems*



Clients and Orders:

1. One client may have many orders.
2. Each client probably should have at least one order associated with him/her.

3. Each order has one client.
4. Each order must link to a minimum of one client.

Orders and Order Lines:

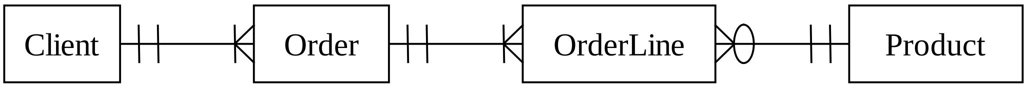
1. One order may have many orderlines.
2. Each order must have at least one orderline associated with it.

3. Each orderline exists on one order.
4. Each order line must link to a minimum of one order.

Order Lines and Products:

1. One orderline lists one product.
2. Each orderline must link to a minimum of one product.

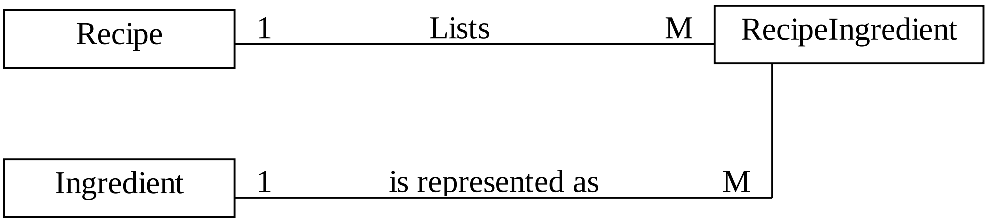
3. Products may be represented on many orderlines.
4. Some products may not be represented on any orderlines.



2.4 Relationships - Labels

As a formal part of an Entity Relationship diagram, labels may be added to the various links. These labels describe the nature of the links and should read like a pictorial sentence

Recipe:



Sentences:

1. A Recipe **lists** many Recipe Ingredients
2. An Ingredient **is represented as** many Recipe Ingredients

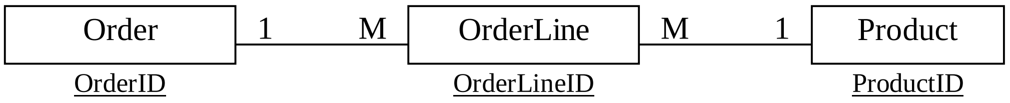
3. Attributes

This section has been broken into three distinct steps:

3.1 Attributes - Identifying (Primary Key)

In this step an identifying attribute (or the primary key of a subsequent database table) is to be established for each entity. As an initial start point, create an ID (or equivalent) identifying attribute for each entity. These ID entries would **not** contain duplicates or be left blank (often stated as: **no duplicates and no nulls)**.

For instance:



Note: *We may wish to change, replace or remove one or more of these Identifying Attributes at a later time.*

3.2 Attributes - Linking (Foreign Key)

Once all the identifying attributes have been added to the Entity-Relationship diagram, attributes that will ultimately link the database tables can be added.

In effect we need to add an attribute between each pair of entities so that we can establish the link from one to the other. For instance, if we have one entity for listing music items and another for listing the tracks on those music items, we would need some way to identify that a certain track exists on a given music item. It can be shown that in order to make such a link we must have a copy of the MusicItemID against each Track entry. In this way we could either: look at a certain track and know what Music Item it is attached to, or locate all tracks attached to a given Music Item.

Looking at sample entries in the two entities (below), we can see that with no linking attribute we have no way of identifying which music item contains the tracks 172 to 174.

MusicItems

MusicItemID	**Item**	**Artist**	**Recording Co.**	**TypeOfMedia**
31	CD 1	Artist 1	RC 1	CD
32	Record 1	Artist 2	RC 2	Rec
33	Cassette 1	Artist 2	RC 2	Cass

Tracks

TrackID	**TrackNumber**	**TrackName**	**Songwriter**	**Length**
172	1	Track 1	Song Writer 1	3 min
173	2	Track 2	Song Writer 2	4 min
174	3	Track 3	Song Writer 1	3 min

We can correct this by adding another column to the Tracks entity: MusicItemsID:

Tracks

TrackID	**MusicItemID**	**TrackNumber**	**TrackName**	**Songwriter**	**Length**
172	**31**	1	Track 1	Song Writer 1	3 min
173	**31**	2	Track 2	Song Writer 2	4 min
174	**31**	3	Track 3	Song Writer 1	3 min

We are now able to identify that tracks 172-174 are all recorded on music item 31 - CD 1.

It is important to note that we cannot add the TrackID to the MusicItems entity. Otherwise for every TrackID we would have to retype the details for the music item containing it:

MusicItems

MusicItemID	TrackID	Item	Artist	Recording Co.	TypeOfMedia
31	**172**	CD 1	Artist 1	RC 1	CD
31	**173**	CD 1	Artist 1	RC 1	CD
31	**174**	CD 1	Artist 1	RC 1	CD
32	**…**	Record 1	Artist 2	RC 2	Rec
33	**…**	Cassette 1	Artist 2	RC 2	Cass

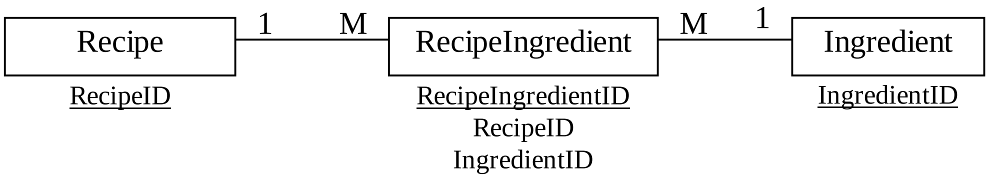
Also note that we would have to retype 31 for the MusicItemID for each track. This results in duplicates being recorded against the Identifying Attribute - MusicItemID - which is not allowed.

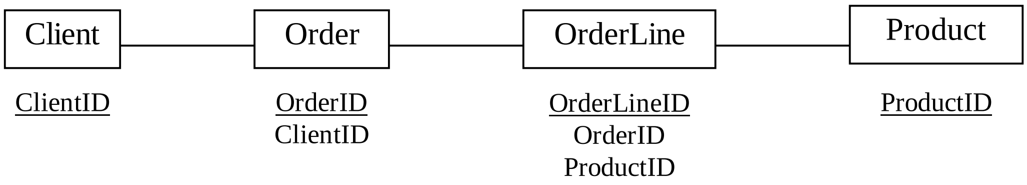
It would be extensively time consuming if we needed to create this type of sample data for every pair of entities in a system. Therefore in order to simplify this process, a second rule of thumb can be utilised:

Rule of Thumb 2: *If two entities are joined in a one-to-many relationship, copy the Identifying Attribute (Primary Key) from the entity on the '1' side and place it in the entity (table) on the many side.*

Rule of Thumb 3: *If two entities are joined in a one-to-one relationship, you may wish to use the same Identifying Attribute in both, and/or link the respective Identifying Attributes.*

Further examples:

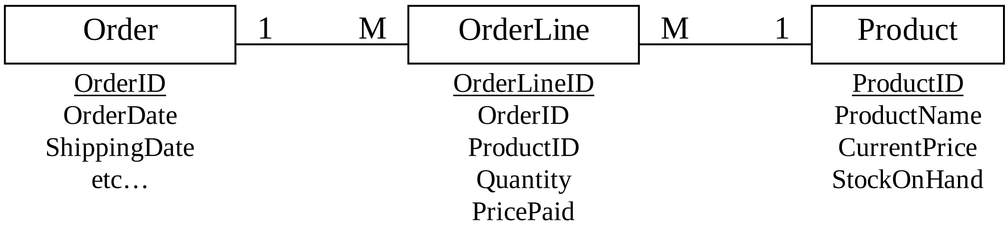




3.3 Attributes - Descriptive

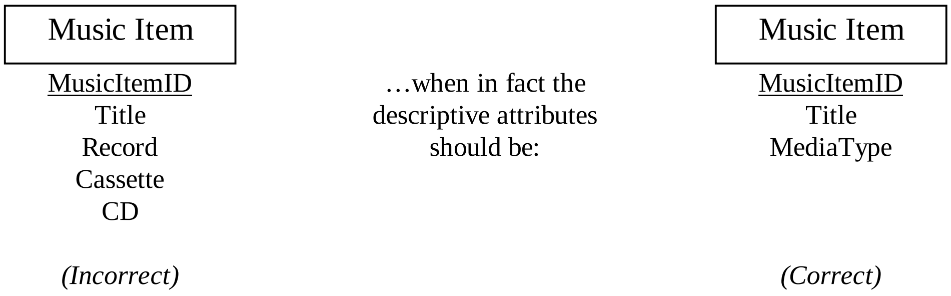
Once the Identifying Attributes and the linking attributes have been added, other descriptive attributes can be listed.

For instance:



Note that the OrderID and ProductID have both been added as linking attributes to the Orderline entity. It is possible for us to remove the OrderLineID Identifying Attribute from the OrderLine entity and use a combination of OrderID and ProductID.

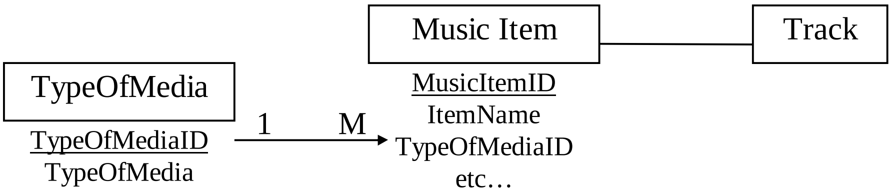
One point of possible confusion that may occur at this point is the distinction between a descriptive attribute and a record that would be added to the corresponding database table. For instance, we may be considering the descriptive attributes that need to be added to the Music Items table. The client or the script has indicated that the music items could be cassettes, records or CDs. There may be a temptation to add cassette, record and CD as descriptive attributes, rather than providing a MediaType attribute and allowing the user to type cassette, record and CD as entries within the associated database table:



4. Lookup Lists

A final step in the process might be to check for various ‘lookup’ lists. These lists would be synonymous with combo boxes on database screens. For instance we might wish to create a ‘lookup’ list of suburbs and post/zip codes. The inclusion of lookup lists in a database is to either: improve data entry efficiency by reducing typing, or to improve data integrity by ensuring that the spelling of various items (such as suburb names) is consistent.

Usually such lists have much less data than the entity or table to which they are linked. A Music Items table in a database might include an attribute (field) - TypeOfMedia (record, cassette, CD etc). A TypeOfMedia lookup list could be created within its own table and be used as the data source for a combo box on the Music Items data entry form. While there might be several hundred entries in the Music Items table, there might only be 4 or 5 entries in the TypeOfMedia table.

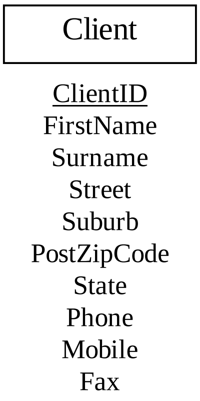


In adding a lookup list, we are adding an entity to the system and in so doing, returning us to the first step of this design process. We are able to fast-track many of the intermediate steps by utilising a forth rule of thumb:

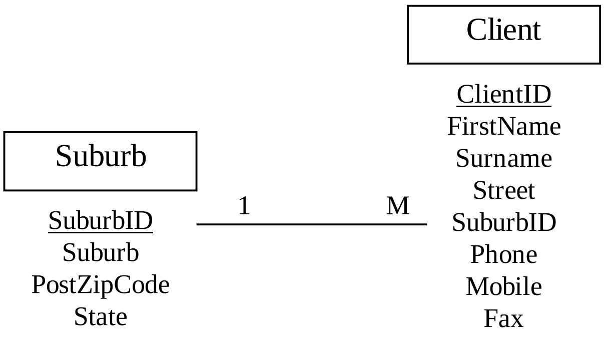
Rule of Thumb 4:	*When you have identified a lookup list, you need to create a new entity or table. The link or relationship between the existing entity and the new lookup list is established between the Identifying Attribute in the new lookup list and the descriptive attribute that requires the lookup list. The cardinality always runs one-to-many from the lookup list to the existing entity.*

A second example might be the creation of a Suburb lookup list for a Client entity.

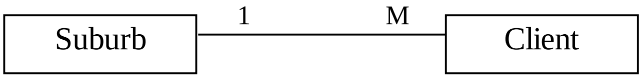
The Client entity might originally be as shown right:



We could remove the Suburb, PostZipCode and State attributes and replace them with a SuburbID attribute. We could then add another entity - Suburb - containing the original Suburb, PostZipCode and State attributes:



Note that the relationship between these two entities (tables) is one to many from the lookup list to the original Client entity. Also be aware that although the link between these two entities has been drawn between the two SuburbID's, the link in a formal ER diagram would be drawn between the two entities:



Lookup lists can be 'hard coded' to various combo boxes or lists boxes on database screens rather than being added as separate entities. That is to say, lists can be typed directly into a program's code or onto property line(s) making them less accessible to the user for updates and changes. This may be a preferable approach where a lookup list is fixed and is **never** likely to change over time.

1. References and Resources

Dennis & Wixom	Systems Analysis and Design - An Applied Approach John Wiley & Sons, Inc, New York, 2000
D’Orazio & Happel	Practical Data Modelling for Database Design Jacaranda Wiley Ltd, Brisbane, 1996
Hawryszkiewycz, Igor	Introduction to Systems Analysis and Design, Fourth Edition Prentice Hall, Sydney, 1998
Kendall & Kendall	Systems Analysis and Design, Fourth Edition Prentice Hall International, Inc, New Jersey, 1999
Shelly, Cashman, Rosenblatt	Systems Analysis and Design, Fourth Edition Course Technology, Thomson Learning, Boston, 2001