Entity Relationship Modelling

A short guide to designing Entity Relationship Models using Barker's notation.

By Frankie Inguanez

Table of Contents

Part 01 - Introduction

Disclaimer

The contents of this document is completely my work and no profit has been generated during the making or through this document.

The Barker's notation is not my work but rather the work of Richard Barker.

This document has been created for research and educational purposes and in no way is it intended for commercial use.

Use of this document in commercial projects is at your own risk.

The colours used in this document are purely for educational purposes and should not be considered as part of the notation being documented.

The name of items, products and persons listed in examples are purely fictional and intended for educational purposes and no advertisement or reference to actual persons is intended!

Scope of this document

This document is intended for database designers, educators and learners, This document is intended as supporting documentation to the topic of Entity Relationship Modelling in particular, using the Barker Notation.

Purpose of this document

My intention for this document is to portray my method and approach of explaining topics and to partially showcase my understanding of such topics.

Part 02 - Modelling

Data Modelling

Definition: A data model is a graphical representation of the solution to a problem as adopted by a particular community.

N.B. Different communities trying to resolve the same problem will produce different yet valid solutions.

Entity Relationship Diagrams

Definition: An Entity Relationship Diagram is a logical diagram representing the database structure using the relational model.

N.B. There are different notations or ERDs, for the following notes we shall be adopting Barker's notation.

General Rules of ERDs

1. Capture all required information
2. Information appears only once
3. Model no information that is derivable from other information already modelled
4. Information is in a predictable, logical place

ERD Concepts

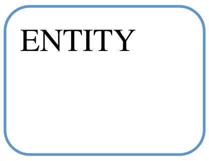
ERDs make use of three concepts:

1. **Entity** – An entity is an object which the community trying to solve a problem, feel that it is important to include. Other communities trying to resolve the same problem might depreciate the entity to an attribute or else might completely omit it from their solution.

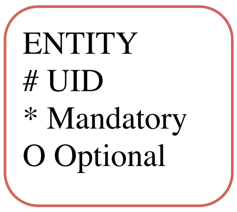
E.g.: In a database collecting information of CDs one would expect to find entities such as CD

N.B. When implementing a database an entity would transmit to a table.

N.B. An entity instance is a record within a table.

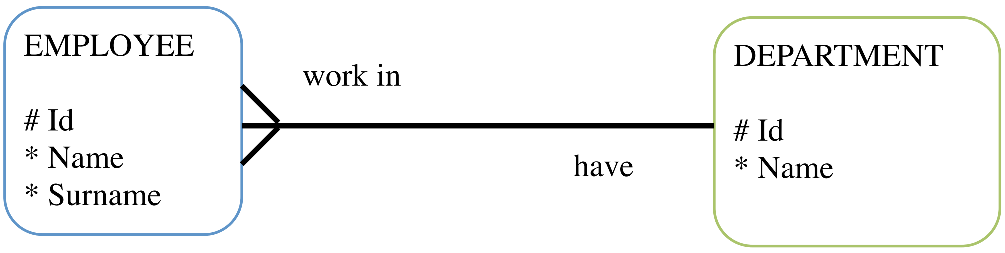


2. **Attribute** – An attribute is a property of an entity which describes the characteristics of a particular entity instance. An attribute can be of three types:
   a. Unique Identifier: A UID is an attribute whose value uniquely identifies an entity instance. A UID is implemented as a Primary Key.
   b. Mandatory Attribute: A mandatory attribute is one whose value cannot be null.
   c. Optional Attribute: An optional attribute is one whose value can be null.



3. **Relationship** – A relationship links two or more entity instances together.

N.B. A relationship is implemented as a Foreign Key and therefore the FK attribute is not listed as an attribute in the target entity.

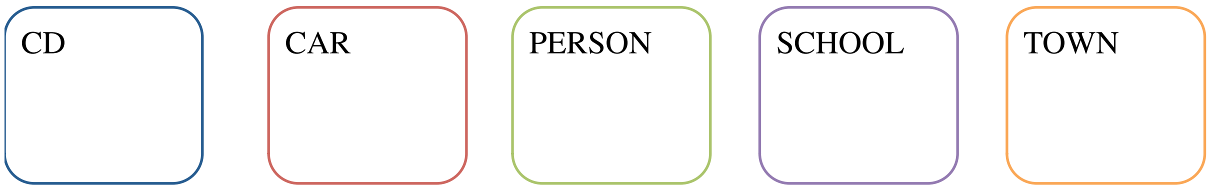


Drawing an Entity

When drawing an Entity using Barker's notation the following rules need to be respected:

1. An entity is a rounded corner rectangle.
2. An entity must be named and the name must be placed inside of the entity, upper most part.
3. The entity name should be in upper case form.
4. The entity name should be in singular form.

E.g.:

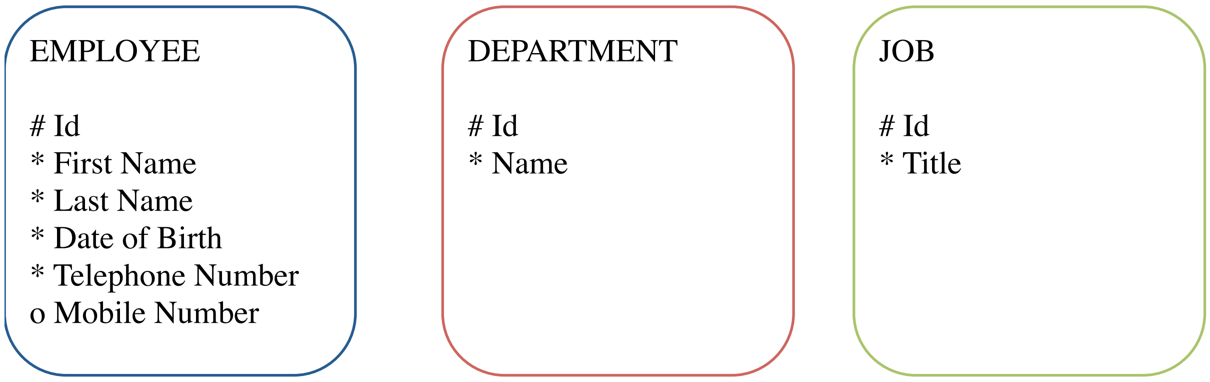


Drawing attributes

When drawing an Attribute using Barker's notation the following rules need to be respected:

1. The attributes must be written such that all persons can understand and not just developers (meaningful terms)
2. Attributes must be written with the first letter of each word in upper case and the rest in lower case.
3. Next to each attribute a symbol should be placed representing the type of attribute.

E.g.:



N.B. The colours used in this document are solely to assist in the explanation of the topics.

Drawing Relationships

When drawing a Relationship using the Barker's notation the following rules need to be respected:

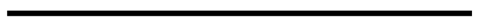
1. A relationship can exist between a maximum of two entities.
2. A relationship can exist on the same entity.
3. A relationship has two perspectives.
4. Both perspectives of a relationship must be labelled.

When drawing a Relationship using the Barker's notation the following steps need to be taken:

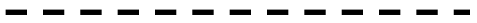
1. Determine the entities affected by the relationship.
2. Determine the optionality of the relationship.
3. Determine the degree of the relationship.
4. Label the perspectives of the relationship.

Optionality of a Relationship

1. **Mandatory Relationship:** A mandatory relationship specifies that each instance from an entity must be related to another instance. This is represented by a straight line.

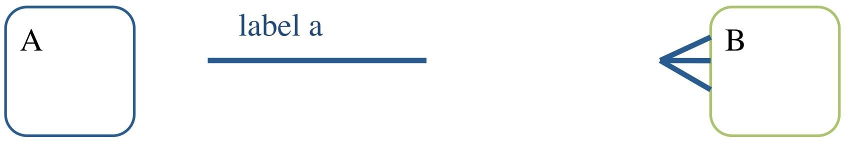


2. **Optional Relationship:** An optional relationship specifies that each instance from an entity may be related to another instance. This is represented by a dashed line.

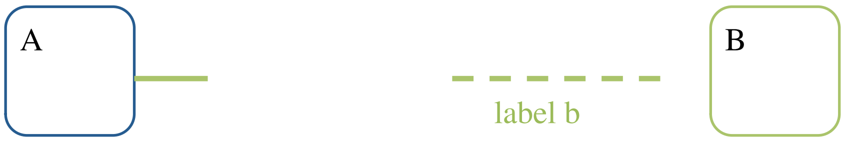


Relationship Perspectives

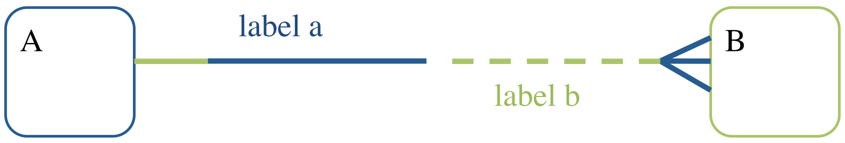
A relationship is always made up of two perspectives using the following notation:



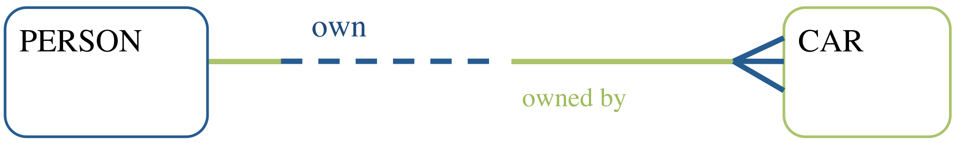
First Perspective / A Perspective: Each A must label a one or more Bs



Second Perspective / B Perspective: Each B may label b exactly one A.

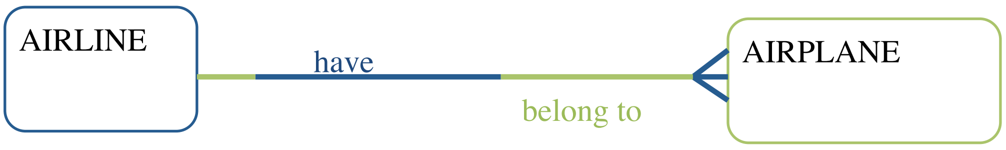


E.g.:



First Perspective / PERSON Perspective:
Each PERSON may own one or more CARs.

Second Perspective / CAR Perspective:
Each CAR must be owned by exactly one PERSON.

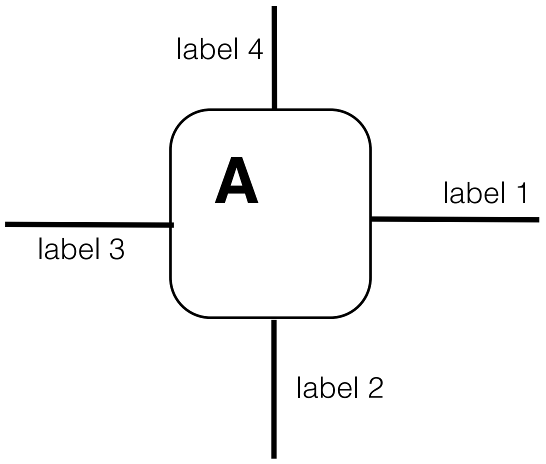


First Perspective / AIRLINE Perspective:
Each AIRLINE must have one or more AIRPLANEs.

Second Perspective / AIRPLANE Perspective:
Each AIRPLANE must belong to exactly one AIRLINE.

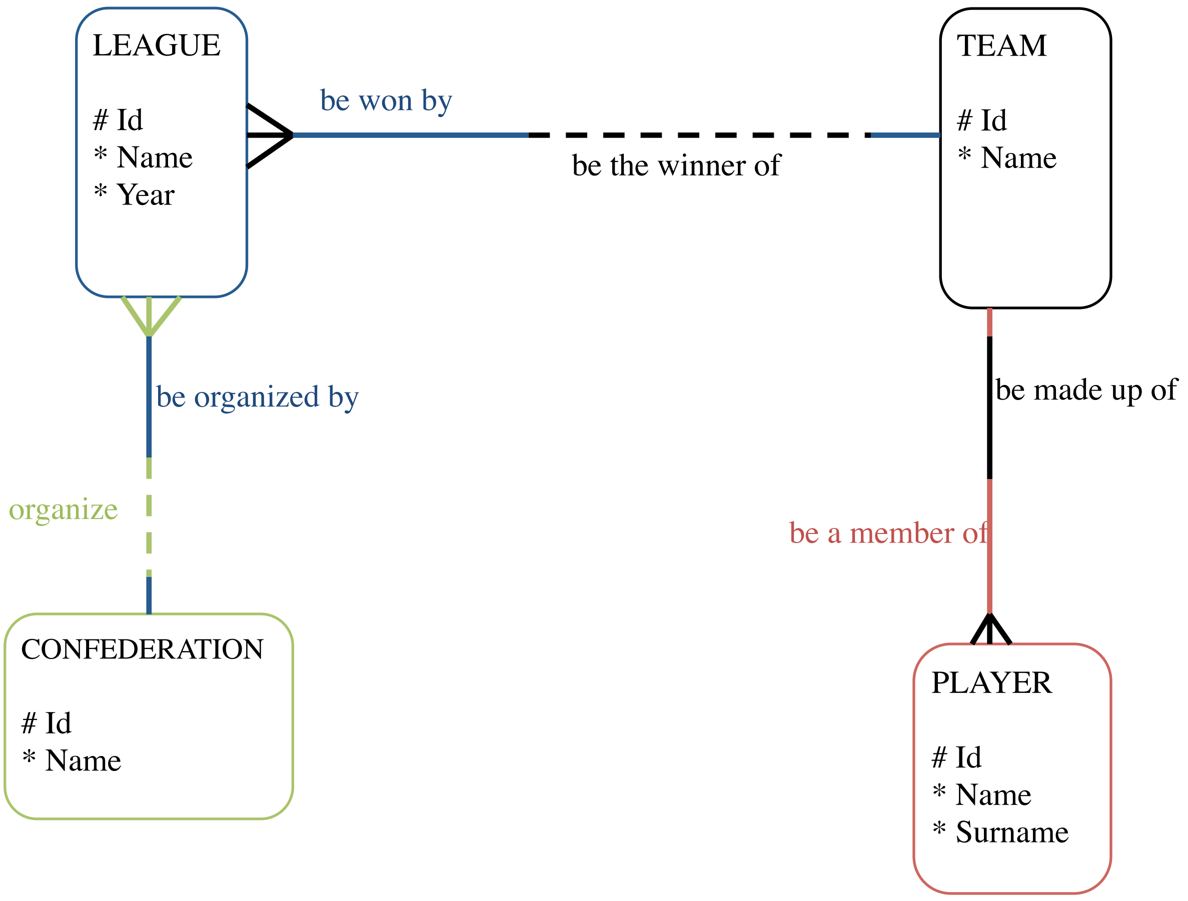
Labeling of relationships

When labelling relationships the following notations need to be followed:



1. A label should be applied for each perspective.
2. A label should be a verb.
3. A label should be written in lower case form.
4. A label should be positioned according to the diagram shown above.
5. A label should be relevant to the scenario.

E.g.:



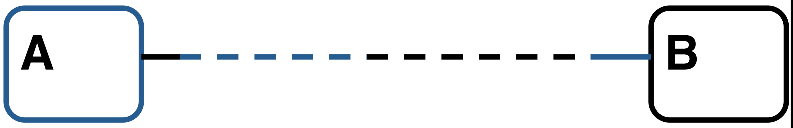
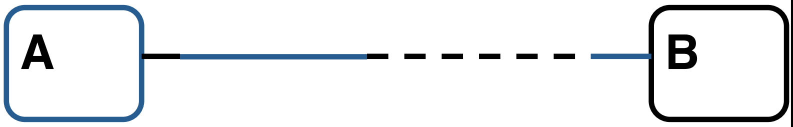
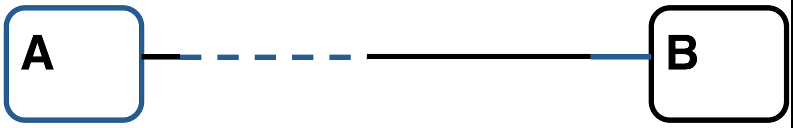
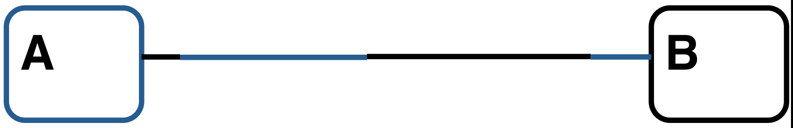
Degree of a Relationship

A relationship can be one of three types:

1. **One-to-One Relationship (1:1)**

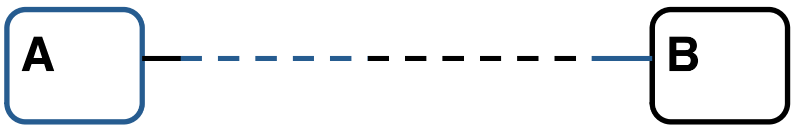
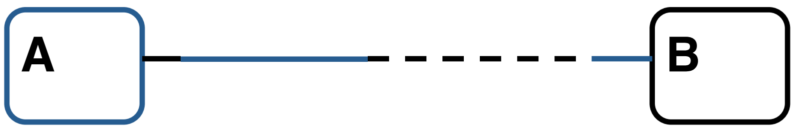
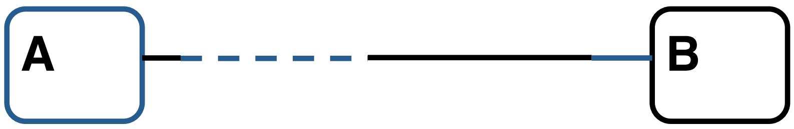
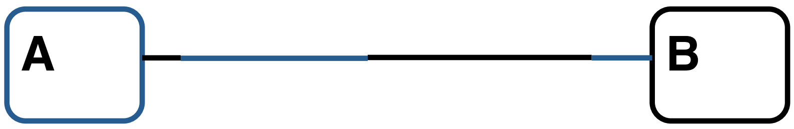
   Each entity instance is related to just one entity instance.

   In the following table we shall see the four possible combinations of a one-to-one relationship and the various optionality. In the last two columns we have an indication of the minimum and maximum number of entity instances that can be related according to perspective.

#	Scenario	Minimum	Maximum
1		**A -> B** 0 **B -> A** 0	**A -> B** 1 **B -> A** 1
2		**A -> B** 1 **B -> A** 0	**A -> B** 1 **B -> A** 1
3		**A -> B** 0 **B -> A** 1	**A -> B** 1 **B -> A** 1
4		**A -> B** 1 **B -> A** 1	**A -> B** 1 **B -> A** 1

N.B. Labels and attributes have been removed from the above scenario since they are not the focus of the topic.

The following table indicates where the foreign keys should be implemented:

#	Scenario	Foreign Key
1		FK should be placed in the entity with the least number of rows/nulls or where it is the most relevant.
2		FK should be placed in entity A since there will never be any NULL values in the FK.
3		FK should be placed in entity B since there will never be any NULL values in the FK.
4		FK should be placed where it is the most relevant according to the scenario since it will never be a NULL value in the FK.

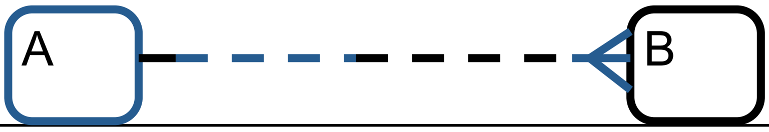
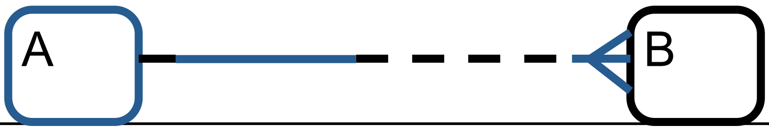
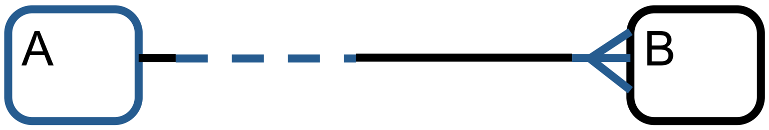
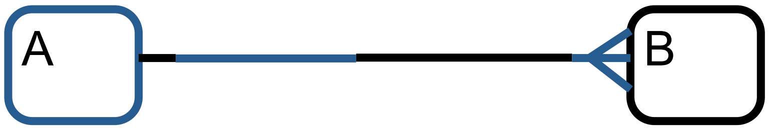
N.B. Labels and attributes have been removed from the above scenario since they are not the focus of the topic.

N.B. The FK attribute would never be listed in the ERD, the above explanation is referring to the database implementation phase.

2. One-to-Many Relationship (1:M)

Each entity instance is related to multiple entity instances.

In the following table we shall see the four possible combinations of a one-to-one relationship and the various optionality. In the last two columns we have an indication of the minimum and maximum number of entity instances that can be related according to perspective.

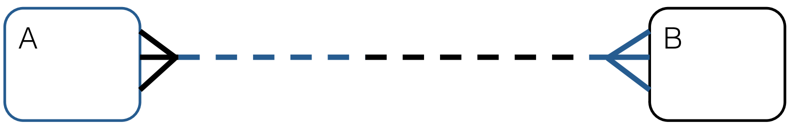
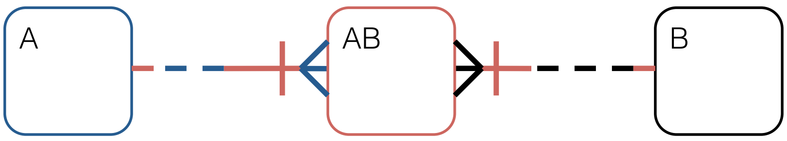
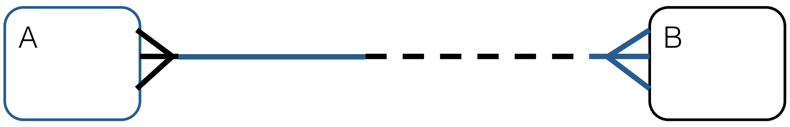
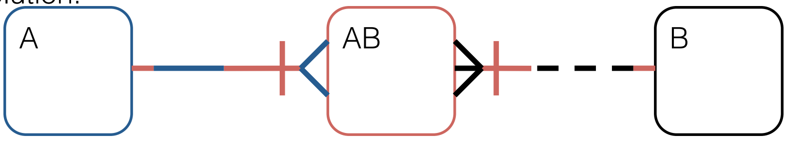
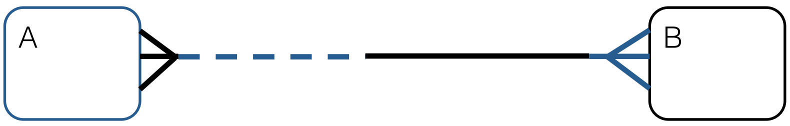
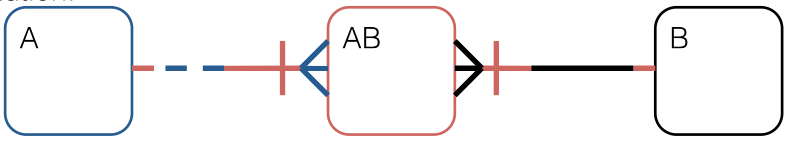
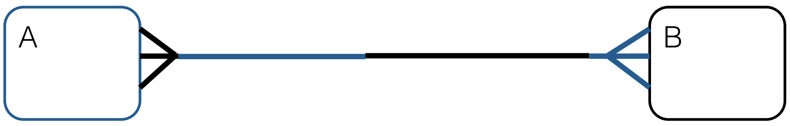
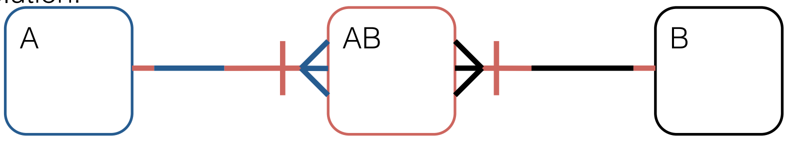
#	Scenario	Minimum	Maximum
1		**A -> B** 0 **B -> A** 0	**A -> B** n **B -> A** 1
2		**A -> B** 1 **B -> A** 0	**A -> B** n **B -> A** 1
3		**A -> B** 0 **B -> A** 1	**A -> B** n **B -> A** 1
4		**A -> B** 1 **B -> A** 1	**A -> B** n **B -> A** 1

N.B. Labels and attributes have been removed from the above scenario since they are not the focus of the topic.

N.B. When implementing this type of relationship a foreign key should be placed in entity B, meaning the entity touched with the crow's foot, many symbol, since many rows in B will be related to only one row in A.

3. Many-to-Many Relationship (M:M)

This type of relationship is never required and needs to be resolved.

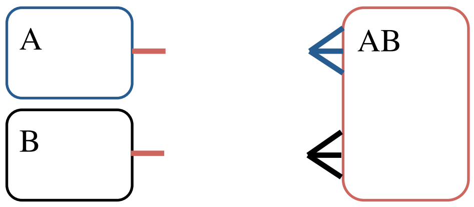
#	Scenario
1	Problem:  Solution: 
2	Problem:  Solution: 
3	Problem:  Solution: 
4	Problem:  Solution: 

N.B. Labels and attributes have been removed from the above scenario since they are not the focus of the topic.

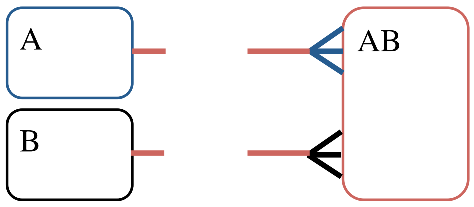
The following steps need to be followed when resolving a many-to-many relationship:

1. Create a third/intermediary/transactional entity.

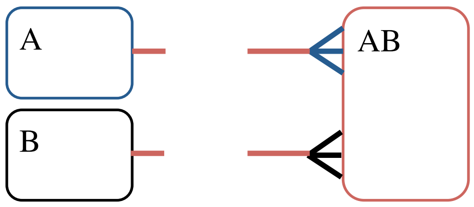
2. Add two one-to-many relationships from the original entities to the new entity.



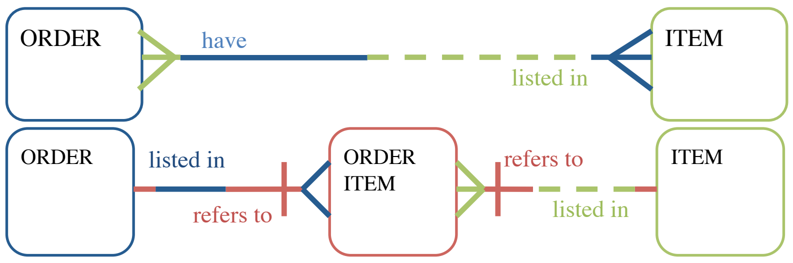
3. The optionality from the AB perspective should always be mandatory.



4. The optionality of the remaining perspectives is derived from the original many-to-many relationship.
5. UID Bars are added to the AB perspectives (These indicate that the foreign keys are also primary keys – refer to next chapter UID Bar).



E.g.:



UID Bar

It could be the case that sometimes a foreign key is also part of a primary key, generally a composite key. Consider the following scenario:

> *A database stores a list of daily orders. A table is needed for the daily orders and another for the various items available. A third table (resolving a many-to-many relationship) lists the items sold in each order*

Implementation:

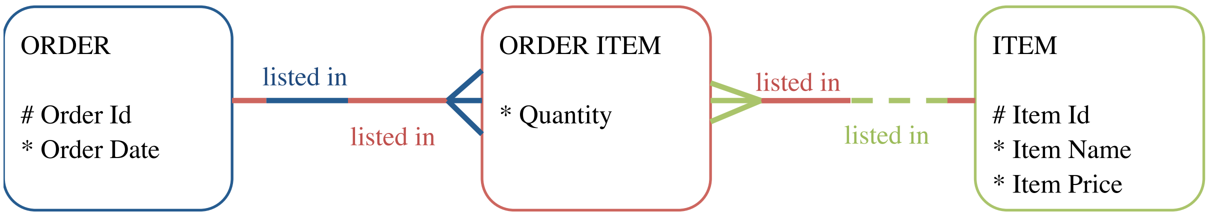
Table: Orders	
Order ID	**Order Date**
1	22/11/10
2	23/11/10

Table: Order Items		
Order ID	***Item ID***	**Quantity**
1	10	3
1	20	1
1	30	2
2	10	2
2	20	1

Table: Items		
Item ID	**Item Name**	**Item Price**
10	Coca	50
20	Fanta	40
30	Sprite	30

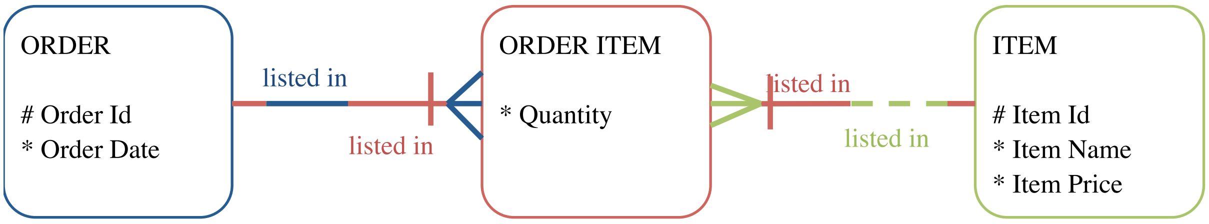
N.B. Primary Keys are underlined and foreign keys are in italics.

ERD:



Problem: The problem with the above ERD is that we cannot indicate that the same foreign keys found in ORDER ITEM, represented by the relationships, are also the primary keys of the same entity. We cannot list the attributes and add a UID symbol (#) since Barker's notation requires that no data is represented twice.

Solution:



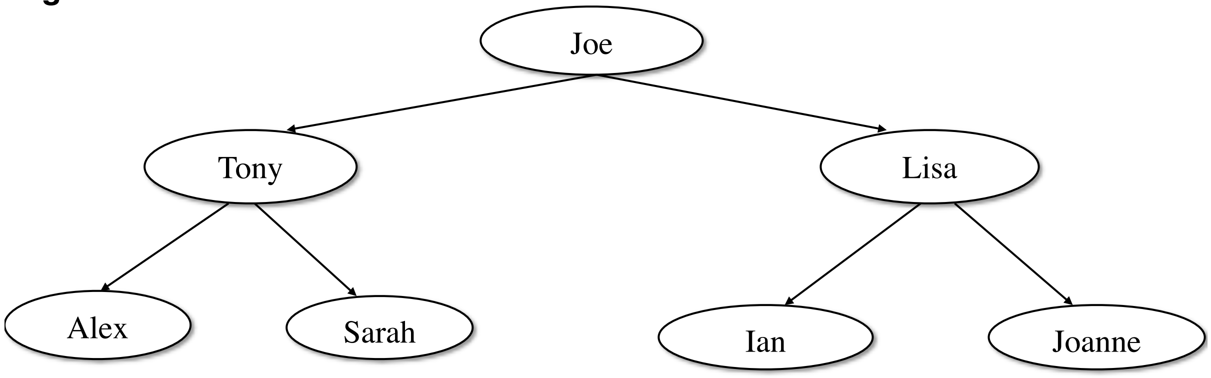
The solution is to add the UID Bar to the ORDER ITEM perspectives. Each individual UID Bar represents the fact that the foreign key represented by the relationship is also a primary key of the entity to which the perspective belongs to.

Recursive Relationships

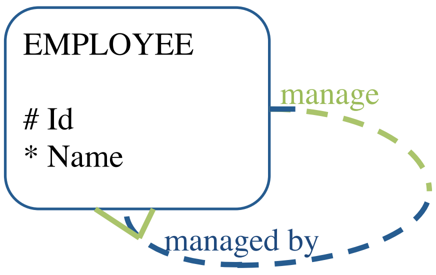
There are situations where a relationship acts upon the same entity. This type of relationship is called a recursive relationship and is generally implemented by having a foreign key referring to the primary key of the same entity. Consider the following scenario:

> *A company is made up of a number of employees. Joe is the chairman and has no manager. Joe is the manager of Tony and Lisa. Tony is the manager of Alex and Sarah. Lisa is the manager of Ian and Joanne.*

Organisational Structure Chart:



ERD:



Perspectives:

Each EMPLOYEE may manage one or more EMPLOYEEs.
Each EMPLOYEE may be managed by exactly one EMPLOYEE.

Implementation:

Table: Employees

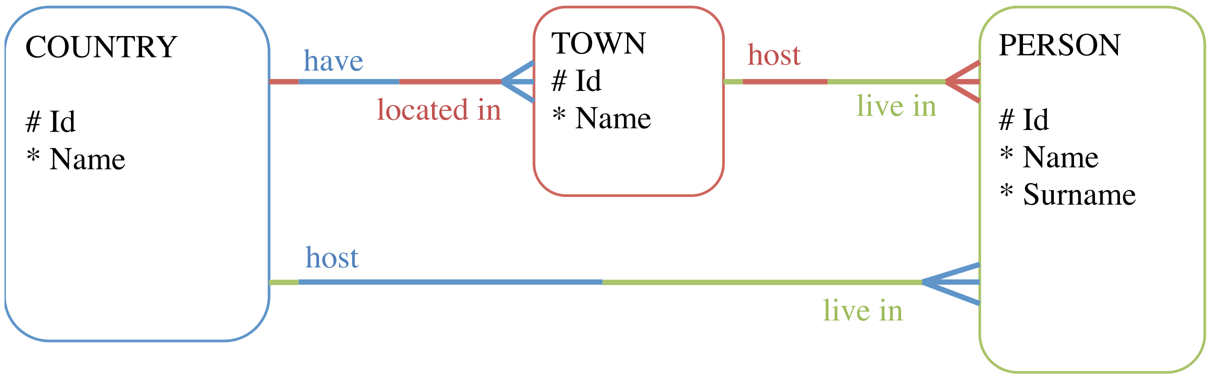
<u>**Employee Id**</u>	**Name**	***Manager ID***
1	Joe	
2	Tony	1
3	Lisa	1
4	Alex	2
5	Sarah	2
6	Ian	3
7	Joanne	3

N.B. Primary keys are underlined and foreign keys are in italics.

Redundant Relationships

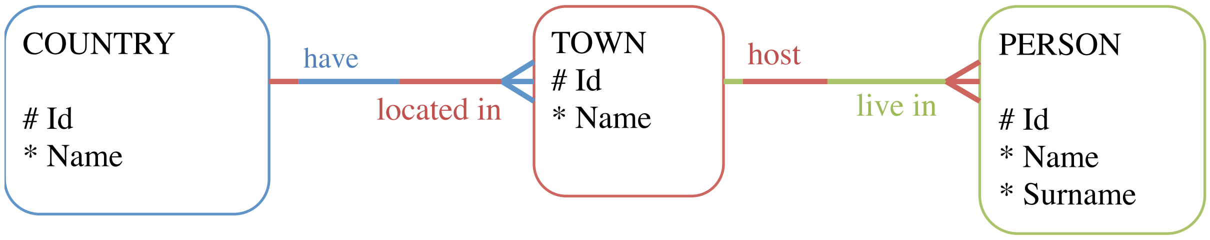
It could be the case that the information represented by a relationship can be derived from other information already being represented in an ERD. This relationship is called a redundant relationship and needs to be removed.

E.g.:



Problem: The relationship between COUNTRY and PERSON is redundant since the same information can be deduced from the other two relationships.

Solution ERD:



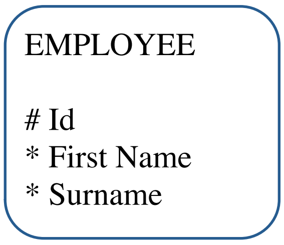
N.B. There can be other relationships between COUNTRY and PERSON as long as it represents different information.

UID Attributes

There are four types of Unique Identifier attributes which are implemented as primary keys:

1. **Single UID Attribute:** A single UID attribute is when an entity is made up of only one UID attribute which is not a foreign key.

   ERD:



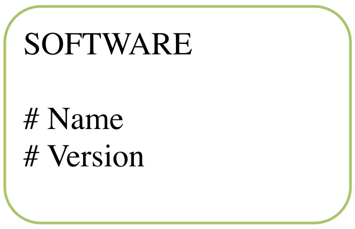
Implementation:

Table: Employees		
Id	**First Name**	**Surname**
1	Joe	Borg
2	Tony	Galea
3	Lisa	Abela

N.B. Primary keys are underlined.

2. **Multiple UID Attribute:** A multiple UID attribute is when a primary key is made up of multiple attributes (composite primary key) all of which are non-foreign keys.

ERD:



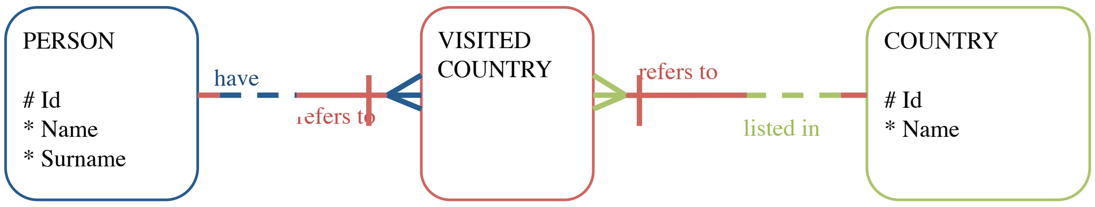
Implementation:

Table: Software	
Name	**Version**
Oracle DBMS	11g
MySQL	5.1
SQL Server	2008

N.B. Primary keys are underlined.

3. **Composed UID Attribute:** A composed UID attribute is when an entity has a primary key which is also a foreign key. These are marked with a UID Bar.

ERD:



Implementation:

Table: Persons		
Id	**Name**	**Surname**
1	Joe	Borg
2	Tony	Galea
3	Lisa	Abela

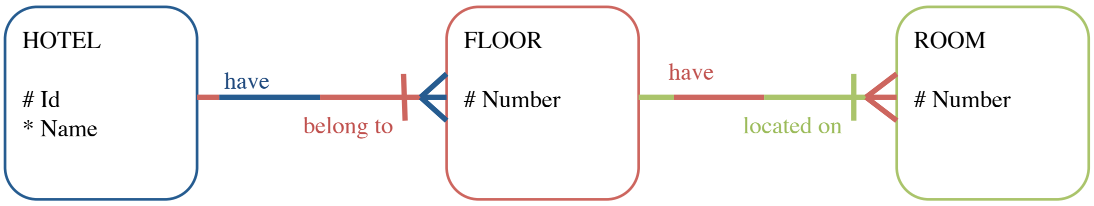
Table: Visited Countries	
Person ID	***Country ID***
1	1
1	2
1	3
2	4
3	4

Table: Countries	
Id	**Name**
1	Italy
2	England
3	France
4	Switzerland

N.B. The attributes Person Id and Country Id in the relation Visited Countries are the composed UID attributes since they are both foreign keys as well as primary keys in the VISITED COUNTRY entity. This is represented by the UID Bars on both relationships.

4. **Composed Cascade UID Attribute:** A composed cascade UID attribute is when an entity uses its foreign keys as primary keys from an entity with composed UID attributes.

ERD:



Implementation:

Table: Hotels	
Id	**Name**
1	Hilton
2	Westin

Table: Floors	
Hotel ID	**Floor Number**
1	1
2	1
2	2

Table: Rooms		
Hotel Id	***Floor Number***	**Room Number**
1	1	1
1	1	2
2	1	1
2	1	2
2	2	1

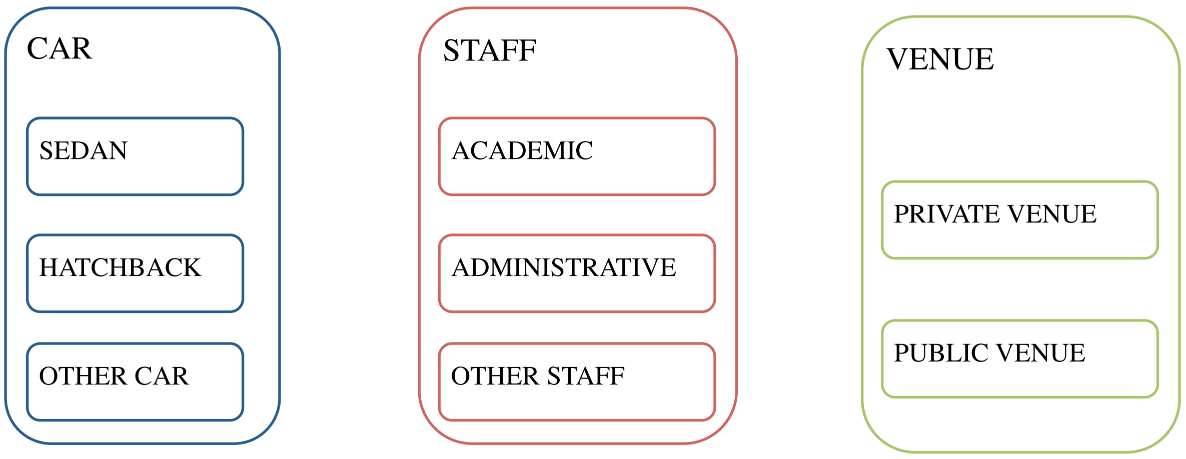
N.B. Primary keys are underlined and foreign keys are written in italics.

N.B. Hotel Id and Floor Number in ROOM are composed cascade attribute UIDs.

Entity Subtypes

There are situations where different entities can be grouped together under one common entity called a super type entity whilst the nested entities would be the subtype entities.

E.g.:



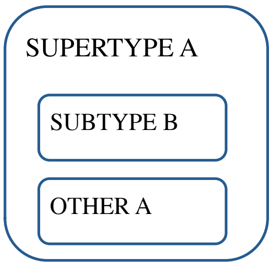
Entity Subtypes Rules

1. **Exhaustive Rule:** This rule states that every entity instance of the super type must be an instance of one of the subtypes.

2. **Mutually Exclusive Rule:** This rule states that every entity instance of the super type can be an instance of only one entity subtype and not the other.

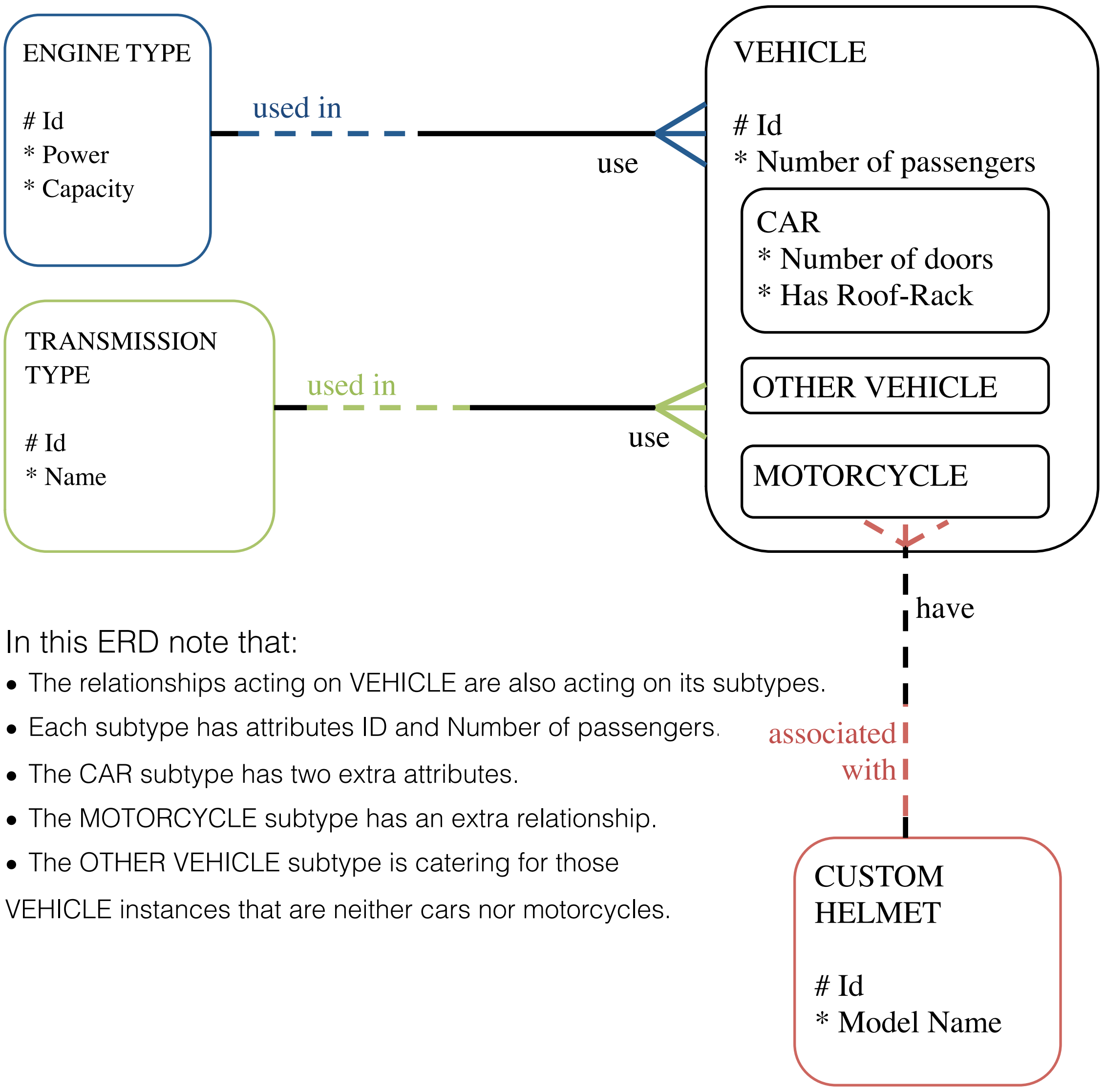
Drawing subtypes

When drawing entity subtypes the following notes need to be kept in mind:
1. Each subtype is a specialisation of a super type and therefore must be enclosed within an entity.
2. The common attributes and relationships for all subtypes must be listed in the super type only but are inherited in every subtype.
3. A subtype can and would generally have attributes and relationships of its own.
4. There can never be just one subtype; another subtype should be created to cater for the rest.



N.B. If none of the subtypes have any unique attributes or relationships then it is not recommended to make use of subtypes but rather create a separate entity containing a list of the various types required.

E.g.:



In this ERD note that:

- The relationships acting on VEHICLE are also acting on its subtypes.
- Each subtype has attributes ID and Number of passengers.
- The CAR subtype has two extra attributes.
- The MOTORCYCLE subtype has an extra relationship.
- The OTHER VEHICLE subtype is catering for those

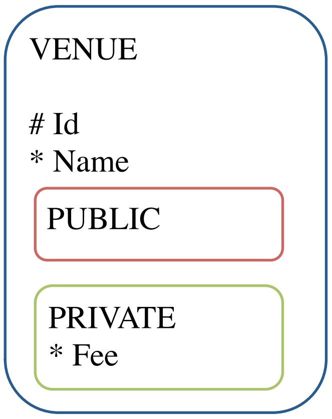
VEHICLE instances that are neither cars nor motorcycles.

Implementing a subtype

There are various methods of implementing a subtype but the preferred method is as follows:

1. Create a table for the super type with all common attributes and relationships.
2. Create a separate table for each subtype.
3. In the subtypes include the specific attributes and relationships.
4. Create a foreign key in the super type for each subtype.
5. Impose a constraint that only one foreign key may not be null.
6. Impose a constraint that at least one foreign key must not be null.

ERD:



Implementation:

Table: Venues			
Vid	**Name**	***Pub_id***	***Prv_id***
1	Floriana Granaries	1	
2	Villa Arrigo		1
3	MFCC		2

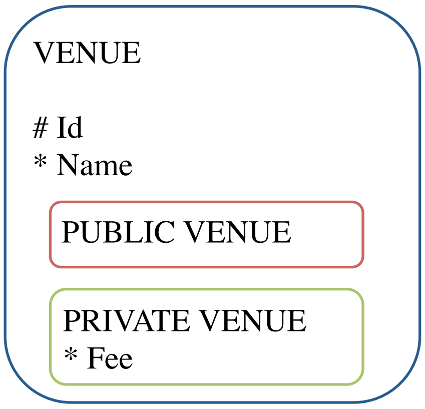
Table: Public Venues
Pub_id
1

Table: Private Venues	
Prv_id	**Fee**
1	500
2	600

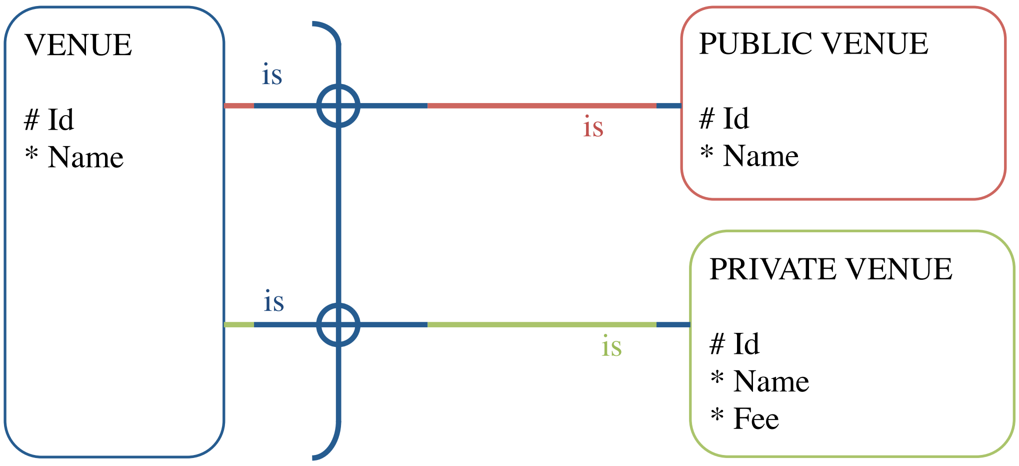
Exclusive Relationship Arc

There can be situations where an entity is either related to one entity or to another but not both. Whenever a selection needs to be done between relationships, an exclusive relationship arc is created.

Recall the subtypes scenario:



The above entity with respective subtypes can, in principal, be represented as follows:



Each VENUE must either be exactly one PUBLIC VENUE or PRIVATE VENUE.
Each PUBLIC VENUE must be exactly one VENUE.
Each PRIVATE VENUE must be exactly one VENUE.

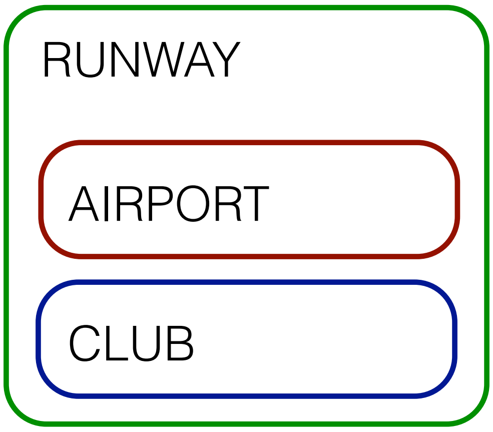
N.B. Even though both rules of subtypes can be modelled as shown above they should always be represented as subtypes rather than using the arc.

N.B. If other relationships cross the exclusive relationship arc but are not affected by the exclusivity then they will not be circled.

Consider:

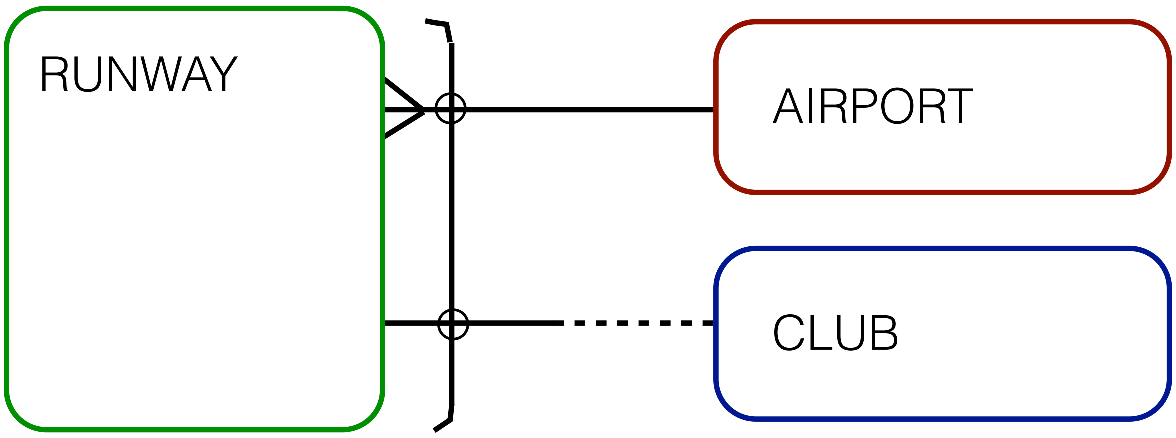
A database is required to store a list of runways. A runway might be owned by an airport, such as Malta International Airport. An airport would own at least one runway but may own more. A runway may instead be owned by a club, such as a Motor Racing Club that owns the runway in Ħal-Far, or the Remote Control Club that owns the runway in Ta' Qali. Lets say that a law in our country imposes that a club is allowed to own only one runway. Lets also say that no runway may be abandoned.

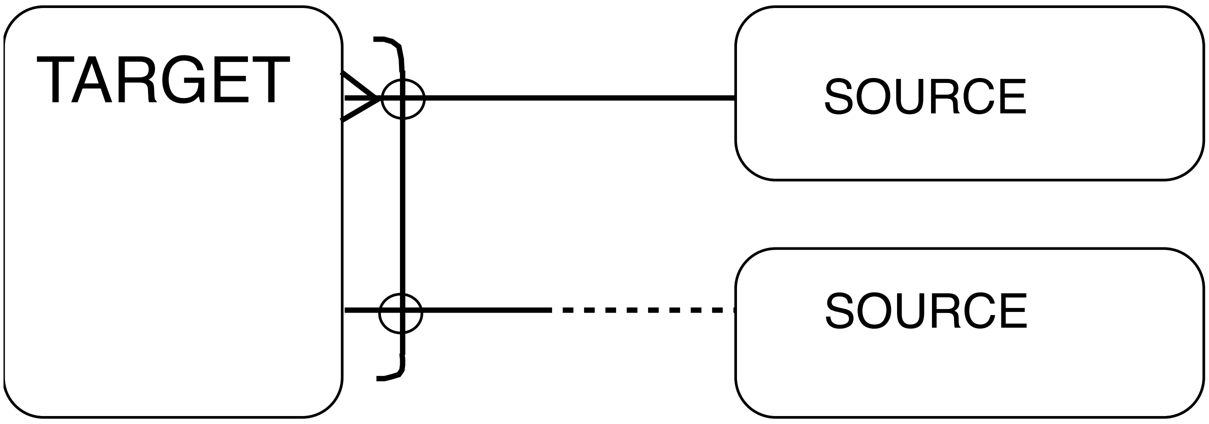
N.B. We cannot use subtypes since a RUNWAY is not an AIRPORT, nor is it a CLUB. Also an airport may have more than one runway whilst a club may have only one runway. Therefore the following is wrong.



This scenario therefore cannot be resolved by using subtypes. But we need the mutually exclusive rule. Therefore we use the exclusive relationship arc.

Solution:





Rules of the exclusive relationship arc

The following rules need to be applied when drawing an exclusive relationship arc:

1. A relationship arc may be applied to only one entity, called the target entity.
2. The relationship arc must be applied to a minimum number of two relationships.
3. The target entity will contain the foreign keys of the relationships affected by the arc.
4. The optionality of the relationships affected by the arc must be the same from the perspective of the target entity.
5. The optionality of the relationships affected by the arc may be different from the perspectives of the source entities.
6. The relationships affect by the arc may be have a different cardinality (one-to-one, one-to-many).

Drawing an exclusive relationship arc

To design an exclusive relationship arc the following steps need to be taken:

1. Determine the owner of the relationship arc.
2. Draw an arc across the relationships to be affect and around the entity that owns the arc.
3. Circle the crossed relationships that are to be affected.

Implementing an exclusive relationship arc

To implement an exclusive relationship arc the following steps need to be done:

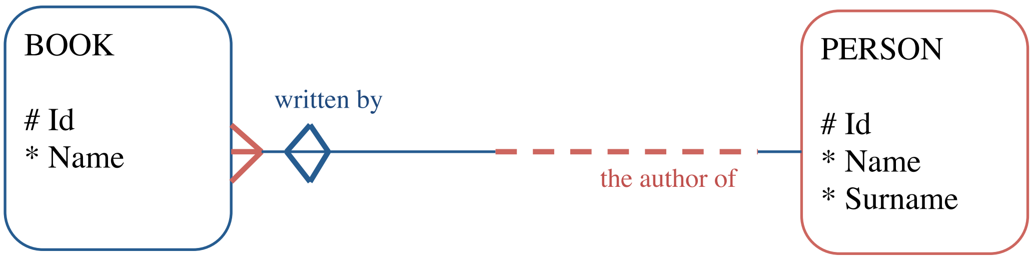
1. Create a foreign key for each relationship affected by the arc. The foreign keys must be placed in the entity to which the arc belongs to.
2. Implement a constraint that allows only one foreign key value not to be null.

Non-Transferability of Relationships

In certain situations once a relationship is set it can never change. We call this the non-transferability of a relationship.

E.g.:

ERD:



Perspectives:

Each BOOK must be written by exactly one PERSON.

Each PERSON may be the author of one or more BOOKs.

Drawing of Non-Transferability

When drawing the non-transferability of a relationship the following rules need to be respected:

1. The diamond symbol should be on the perspective where the foreign key will be implemented.
2. The non-transferability symbol should be displayed only once per relationship provided that any many-to-many relationships have been resolved.

Implementation of Non-Transferability

When implementing the non-transferability of a relationship a constraint is applied to the foreign key which does not allow the values to be updated.

Part 04 - Notes

Part 04 – Feedback and Queries

Contact Details

In case you need to query about this report or would like to provide feedback you can use the following contact details:

Name:	Frankie Inguanez
E-Mail:	frankie.inguanez@gmail.com
LinkedIn:	http://mt.linkedin.com/in/frankieinguanez