Database Answers

Learn Data Modelling by Example

Barry Williams

Part 1 - Table of Contents

First Edition: London, 2011

ISBN-13: 978-1466237414

Welcome

This is Part 1 of our book has been produced in response to a number of requests from visitors to our Database Answers Web Site.

It is intended for beginners to Data Modeling

It incorporates a selection from our Library of about 950 data models that are featured on our Web site:

- http://www.databaseanswers.org/data_models/index.htm

I hope you enjoy this Book and would be very pleased to have your comments at barryw@databaseanswers.org.

Barry Williams
Principal Consultant
Database Answers Ltd.
London, England

1. Data Modeling at Windsor Castle in England

1.1 Introduction

This first Chapter is a tutorial on data modeling for young people. It provides an introduction to data modeling that we hope you find interesting and easy to read.

It covers the basic concepts and has a very user-friendly approach, featuring a teddy bear and kitten creating a data model on a trip as tourists to Windsor Castle, which is just outside London, England.

You can find this chapter as a tutorial on the Database Answers Web site:

http://www.databaseanswers.org/tutorial4_data_modeling_dimple_and_toby_visit_windsor_castle/index.htm

In this tutorial, we will follow two young tourists as they visit Windsor Castle and create a data model.

Our tourists are Dimple, a 10-year-old girl, who likes sightseeing and ice cream and Toby, Dimple's 12-year-old brother, who likes sightseeing and designing data models.

1.1.1 What is this?

This is a tutorial on data modeling for young people that represents a typical data modeling project and illustrates the basic principles involved.

1.1.2 Why is it important?

Data modeling is important because it is the foundation for so many activities:

- It provides a vehicle for communication among a wide variety of interested parties, including management, developers, data analysts, DBAs and more.
- A physical database can easily be generated from a data model using a commercial data modeling tool.

1.1.3 What Will I Learn?

You will learn:

- How to create a data model, starting from scratch.
- What a typical data model looks like.

1.2 Topics

In this chapter, we will cover some basic concepts in data modeling:

- Primary and Foreign Keys
- One-to-Many and Many-to-Many Relationships
- Hierarchies and Inheritance
- Reference Data

1.3 Let's go to Windsor

[Dimple]: Toby, it's great being in London, which is so exciting and buzzing.

[Toby]: I'm glad you like it, Dimple. What would you like to do today?
[Dimple]: Toby, we have seen Buckingham Palace, where the Queen of the United Kingdom lives, and now I'd like to visit Windsor Castle, because it's one of the most popular tourist attractions in the UK, and it's just a short trip from London.

[Toby]: OK. Let's go...

We are starting from Buckingham Palace, where the Queen of the United Kingdom lives …

Toby and Dimple leave London and arrive at Windsor…

1.4 Arriving at Windsor

[Dimple] Wow, Toby, Windsor has a beautiful castle and here is a royal park with lots of deer.

[Toby] Yes, Dimple, and when we look around there are so many banks, cafes, pubs, restaurants, shops, wine bars and hospitals!

The other thing that we see when we look around is people - lots of people.

So we can start thinking about our data model.

1.5 Starting our Data Model

[Dimple]: How do we get started?

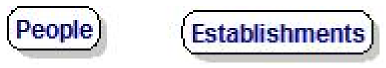
[Toby]: Well, we know that we have people and places.
The simplest start is to call all these places **establishments**.
Then we simply have different kinds of establishments.

And we have people - local people, tourists, students, people passing through, people working here, people here on business and so on.

[Dimple]: Hmmm - so how do we translate what we know to help us get started with our data model?

[Toby]: Let's start a diagram with people and establishments.

This simple diagram is going to grow into a data model.



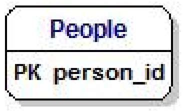
1.6 Identifiers and Primary Keys

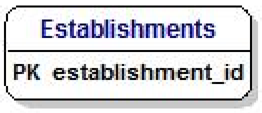
[Dimple]: Toby, I am one of these people so how do I create a unique identity for myself to make me different from everybody else?

[Toby]: We will give every person a **unique identifier** and every establishment its own unique Identifier.
When we use these we call them **Primary Keys**, and show them in the diagram with a **PK** on the left-hand side.

[Dimple]: That sounds good, Toby, but I don't know what it means.

[Toby]: Well, Dimple, let's look at how we use these identifiers...





Lots of people visit establishments like Starbucks in Windsor ;0)

1.7 Relationships and Foreign Keys

[Toby]: Dimple, now we can add some interesting details because we know that one person can visit many establishments.
We also know that one establishment is visited by many tourists.
Then we call this a **many-to-many relationship** between people and establishments.

To make it easier for you to understand I have expanded the **many-to-many relationship** into two different things, which are called **one-to-many relationships**.

[Dimple]: So Toby, is that like saying that one person can make many visits to many establishments?

[Toby]: Yes, Dimple - that's great - and we can also say that one establishment can have visits from many people.

At this point, we can show how all these boxes are related, and that is a very big step, because it takes us to the idea of 'relationships'.

We can call these boxes **tables** - or *entities* if we want to speak to professional data modelers.

A table simply stores data about one particular kind of 'Thing of Interest'.

For example, people or establishments.

Each record in a table will be identified by its own unique identifier, which we call the *primary key*.

It is not usually easy to find a specific item of data already in the table that will always be unique.

For example, in the United States, Social Security Numbers (SSNs) are supposed to be unique, but (for various legitimate reasons) that is not always the case.

Also, foreign visitors and tourists will not have SSNs.

Therefore, it is best practice to create a new field just for this purpose.

This will be what is called an **auto-increment** data type, which will be generated automatically by the Database Management System (DBMS) at run-time.

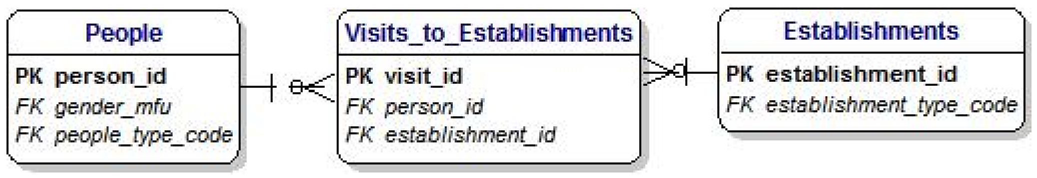
This is called a **surrogate key** and it does not have any other purpose.

It is simply a key that stands for something else.

It is a meaningless integer that is generated automatically by the database management software, such as Oracle or SQL Server. The values are usually consecutive integers, starting with 1,2,3,4 and so on.

Now we can see how useful our identifiers can be because we can include the person and establishment identifiers in our Visits Table.

Then the Person_ID field becomes a link to a record for a person in the Person Table.
This link is what is called a **Foreign Key** and we can see it's shown with **'FK'** on the left-hand side.



1.8 Products and Product Types

[Dimple]: Toby, when we go into a shop we want to buy something.
And there are thousands and thousands of possibilities.
How do we deal with all that in our little data model?

[Toby]: Well Dimple, it's really quite easy. It's like all our modeling where we look for simple patterns that cover many situations.

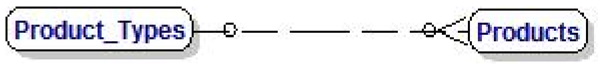
[Dimple]: Hmm - I don't know what that means. Maybe if you showed me I might understand it.

[Toby]: OK.
Everything that we buy is called a **product**, and all we have to do is simply define the type of each product - such as a coffee, muffin or a newspaper.

Then we draw a little box called *Products* and say that every product has a type.
In other words, there is a relationship between the *Products* and *Product_Types* boxes.

The lines are called **relationships** and they are very important in data modeling.
We are now creating an Entity-Relationship Diagram or "ERD".

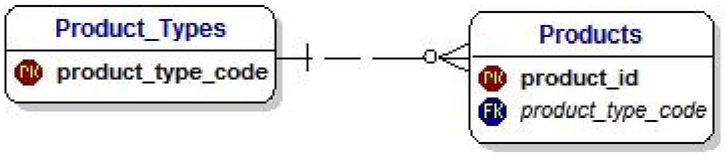
This diagram shows only a line for the relationship:



The symbol at the products end is called *crow's feet* and it shows the *many* end.

The short straight line at the Product_Types end shows the *one* end.

In other words, this line shows a one-to-many relationship.



Dimple, let me explain about the dotted line. It means that the relationship results in a 'Foreign Key' in the products table. This is shown by the 'FK' symbol next to the ***product_type_code*** field and it means that there is a link back to the Product_Types.

However, the primary key is only the Product_ID, and of course, this is shown by the 'PK' symbol next to the **Product_ID** field.

Later, when we talk about inheritance, we will use a straight line, in contrast to this dotted line here. This is to show that the foreign key field is also a primary key.

I have to say something a bit difficult about primary keys right now.

In the Products Table, we have to allow for a very large number of products being stored.

Therefore we use an ID field for the primary key.

We then create this ID field automatically as a number (called an auto-increment integer).

This number has no meaning and is simply used to identify each record uniquely among possibly millions or hundreds of millions.

However, things are different for 'type' fields.

These are what we call enumerated data and are typically **reference data**.

They are always relatively small in number and we choose a code for the primary key because we can create them and review them manually.

It also helps us to create a code that we can use and refer to, in contrast to the ID fields that have no meaning.

Typical examples would be:

- Sizes – Small, Medium and Large where we are accustomed to seeing S,M and L.
- Gender – Male and Female, where we use M and F.

- This menu board at Starbucks shows lots of products.

We know that they are organized in groups, like food and drink, and each of these has more groups and so on, right down to the particular product, like caramel macchiato or a panini.

This top-down organization is called a **hierarchy** and appears all over the place.

Luckily we can show this very easily and neatly in our data model.

1.9 Products, Types and Product Hierarchies

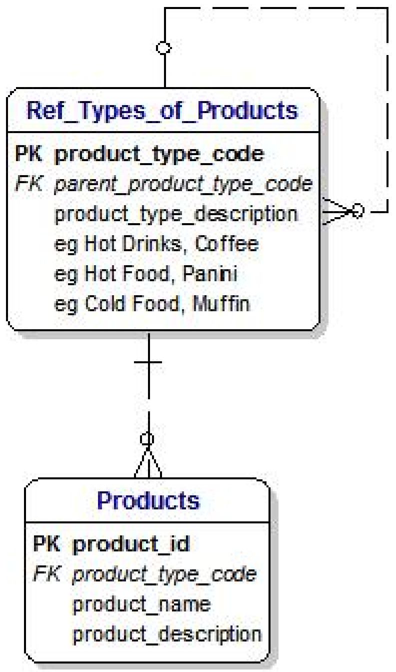
[Dimple]: Toby, when we look closely at the menu board to try to decide what to order we can see lots of possibilities. But after a while we can see a pattern that helps us decide. How do we deal with all that in our little data model?

[Toby]: Well Dimple, it's really quite easy.
We define something called a *hierarchy*.
Hierarchies are very common and simply mean any situation where there are parents, children, grandchildren and so on.
If we look at the Starbucks menu board on the right-hand side we can see a simple example of 'espresso' and under it a number of different drinks.
My favorite is caramel macchiato.
So in this case, the top-level of our hierarchy is a product category called espresso, and the next level down is a product called caramel macchiato.

[Dimple]: OK. That sounds OK.

[Toby]: Finally, we show this hierarchy by a dotted line in the top-right hand corner in the entity called 'Ref_Types_of_Products'.

This is formally called a *recursive* or *reflexive* relationship and is informally called **rabbit ears**.



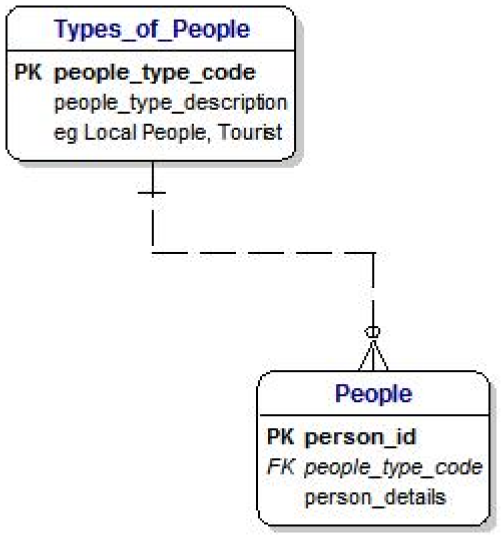
1.10 Types of People

[Dimple]: Toby, that looks OK.
I guess we can deal with types of people the same way, can we?

[Toby]: Yes, Dimple, and types of establishments as well.

[Dimple]: OK, that sounds sensible. And do they use these identifiers in a database?

[Toby]: Yes, and what is even better is that the database will automatically generate a new unique Identifier for you and your visits and purchases if you want to get a refund later.



1.11 Types of People and Establishments

[Dimple]: I see, Toby.
I guess we can deal with types of establishments the same way, can we?

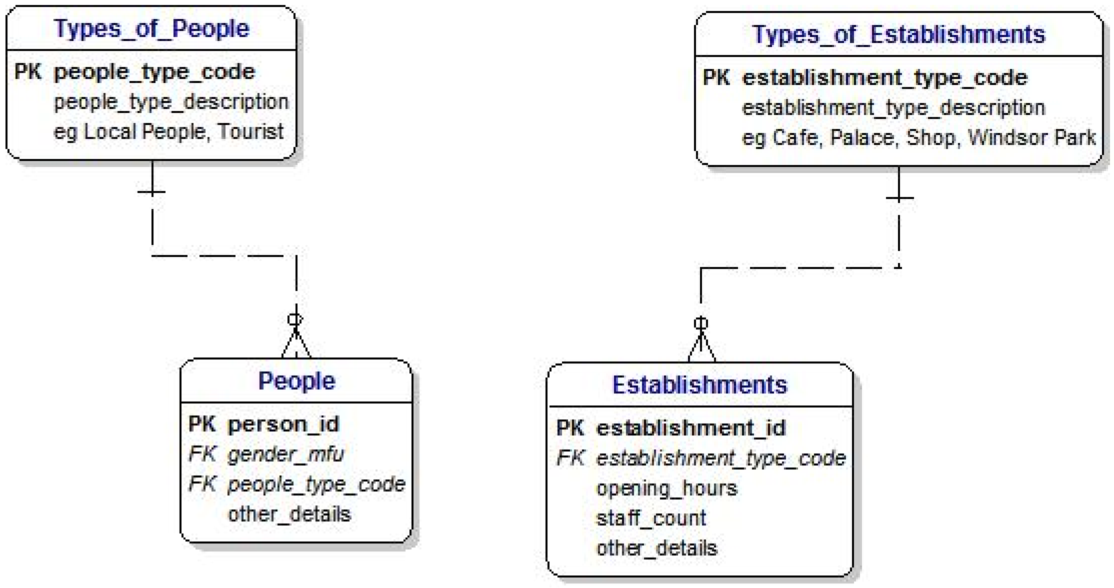
[Toby]: Yes, Dimple.

[Dimple]: OK, that sounds sensible. And do they use these identifiers in a database?

[Toby]: Yes, and we can use our new unique identifier for you and your visits and purchases in case we want to keep track of things.

Like maybe you want to get a refund later so we need to get your details from the database.

[Toby]: Before we move on, let's talk about establishments.
One special thing about Windsor is that it has a castle where the Queen lives and a very large royal park, where she keeps deer.
But when we think about these things, we find that we can simply fit them into our definition of establishments.



1.12 Visits and Purchases:

Here we can see many visitors to Windsor’s Royal Shopping Arcade.

[Dimple]: Toby, with so many people, establishments and purchases how do they keep track of everything?

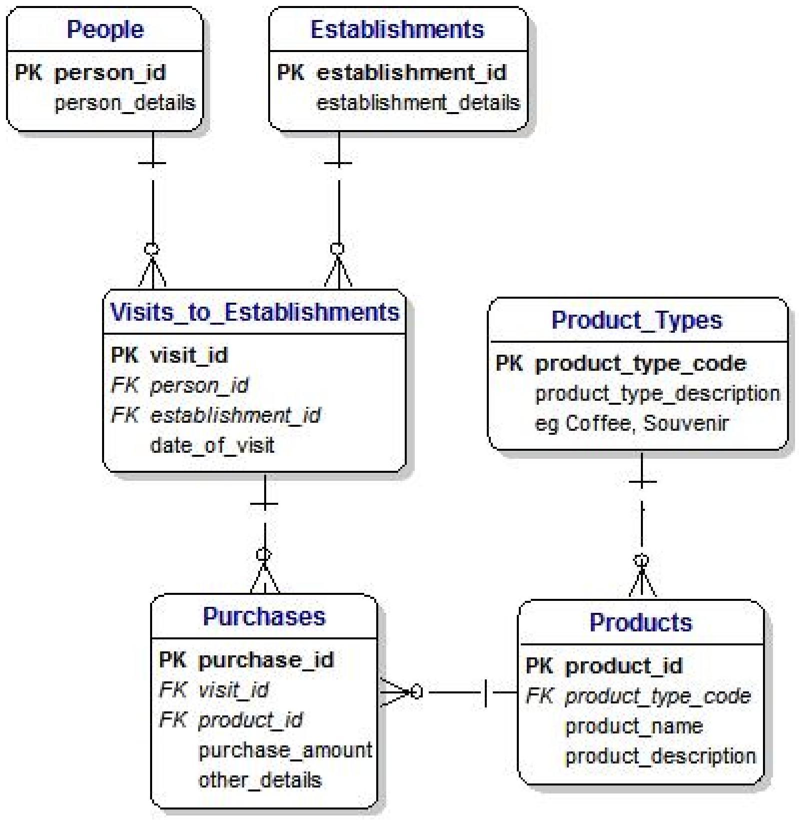
[Toby]: Well, Dimple, by this time, everything has its own identifier that is used wherever they need to keep track.

[Dimple]: OK, that sounds sensible. And do they use these identifiers in a database?

[Toby]: Yes, Dimple, and in this diagram, we can see that we can use the unique identifiers that are shown as 'PK,' for Primary Keys.

We can see that we have a PK for every entity or table so we can be pretty sure we can get from any table to any other table.

This is called *navigating* around the data model and is a good test for a well-designed data model.



1.13 People and Inheritance

[Toby]: Dimple, let's take a closer look at the different types of people we can find in Windsor.

[Dimple]: OK, Toby. I hope I don't have to think too much because I might get a headache?

[Toby]: No, Dimple, I will do the thinking and talking and all you have to do is nod your head when you understand.

[Dimple]: OK, Toby. I promise to do that.

[Toby]: We already said that we have local people and tourists.

There are always lots and lots of people visiting Windsor Castle.

When we look at this picture, we can see ceremonial guards in ceremonial red uniforms, and a big crowd, with mainly tourists but also staff in shops responsible for controlling the crowd, tourists, local people and so on.

Some of these local people are shoppers and some of them will be working in the shops. We will call the workers **staff** and we know different things about them than the things we know about the tourists.

For example, we will probably know the gender of everybody just by looking at them.
For staff, we will usually also know their date of birth and their home address.

In data modeling we have a very powerful approach that we call **Inheritance** that we can use here.

If we want to describe this in English, we would say that staff inherit the People_Type_Code and gender from the parent entity of people, and in addition, they have a date of birth and home address.

For tourists, we don’t know much, except for the date of their visit, and maybe, if they buy something in a shop using a credit card, then the shop would know the credit card details.

For the ceremonial guards in red uniforms, we can tell their rank by looking at their uniform and maybe it would also tell us which unit of the army they belong to.

Does that make sense, Dimple?

[Dimple]: I think so, Toby.

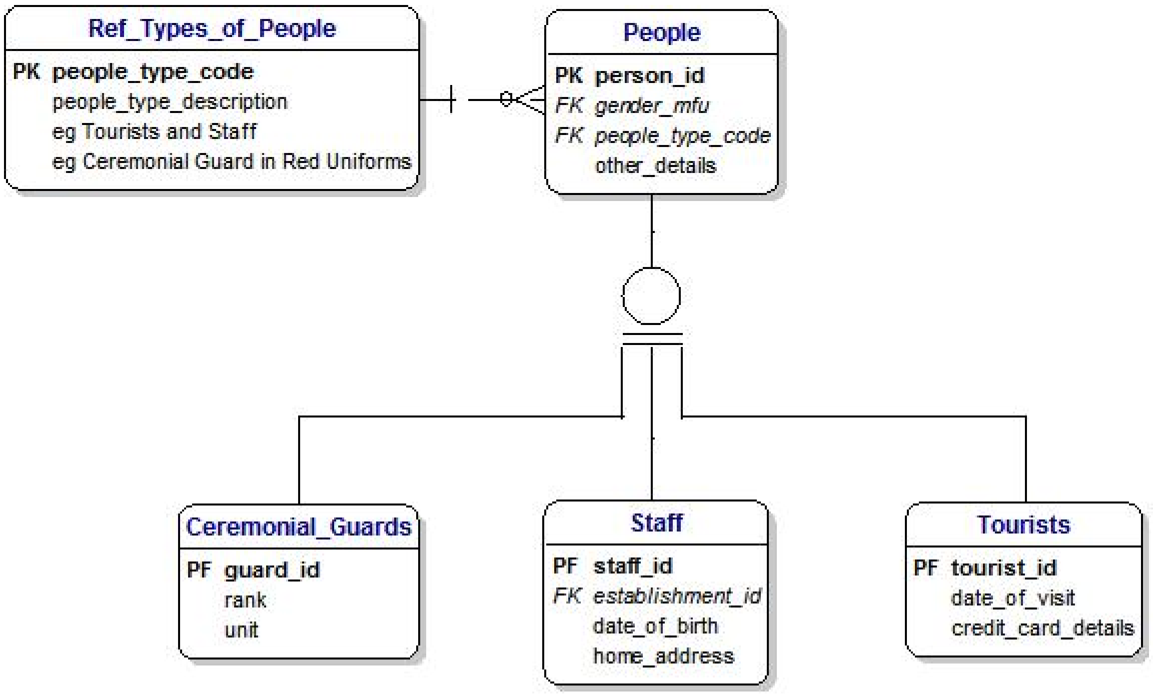
Is it like saying that we inherit having two arms and two legs from our parents because they have two arms and two legs, but that we have also have things that are just us?

[Toby]: Yes, Dimple - that's great - let's take a break and do some shopping!

[Dimple]: I like the sound of that, Toby. Can I have an ice cream?

[Toby]: Yes, of course, Dimple – this diagram shows we are doing well.

It show inheritance between people and the three different types of people:



We can see a field marked as '**PF'** in the three tables for ceremonial guards, staff and tourists.

This is unusual because it means a field that is a **P**rimary Key in the three tables and also a **F**oreign Key to the People Table.

Therefore, if your first record was a ceremonial guard, then we would have a record in the People Table with a Person_ID of 1 and a record in the ceremonial guard with a Guard_ID of 1.

Similarly, if our second record was a member of staff, we would have a record in the People Table with a Person_ID of 2 and a record in the Staff Table with a Staff_ID of 3.

1.14 Staff, Establishments and Derived Fields

[Dimple]: Toby, how do we specify that staff must work in some establishment?

[Toby]: Dimple, that's a very good question.
Fortunately, the answer is very easy.
We add a one-to-many relationship between the staff and establishment entities.
In English, we would say that every member of staff must work in one establishment and every establishment can employ many members of staff.

In the diagram, we show this with a **foreign key** by the Establishment_ID field in the staff entity.
So if we look closely at the staff entity, we will see '**FK**' by the Establishment_ID field.

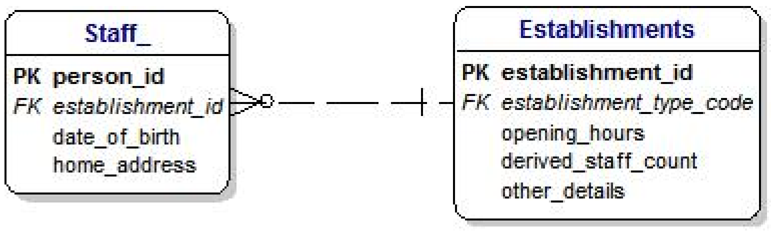
[Dimple]: OK, that sounds good, and I can see how the identifiers are very important.

[Toby]: I am glad to hear it, Dimple.
There is one more thing I have to say.
We are learning data modeling and one important thing about data modeling is that it has to follow a set of **rules**.
These rules help us to produce good data models and so they are very important.
One of the rules is that we cannot include any bits of data that can be derived from any other bits of data.
For example, we usually want to know how many people work in a shop or cafe.
Therefore we include a **staff count** field with the establishment.
But when it comes to finding the value that goes in here, we will count the records in the Staff Table for each establishment.
Therefore, it's a **derived field** and we call it a name that starts with 'derived_' to make things clear.
This is because, according to the rules, we should not include derived fields in our data model at this early stage.

I have shown it here simply as an example because it is a situation that occurs quite often so it's good to recognize it when you see it.

Does that sound sensible, Dimple?

[Dimple]: I suppose so, Toby. But I've got a headache, can we go for an ice cream now?

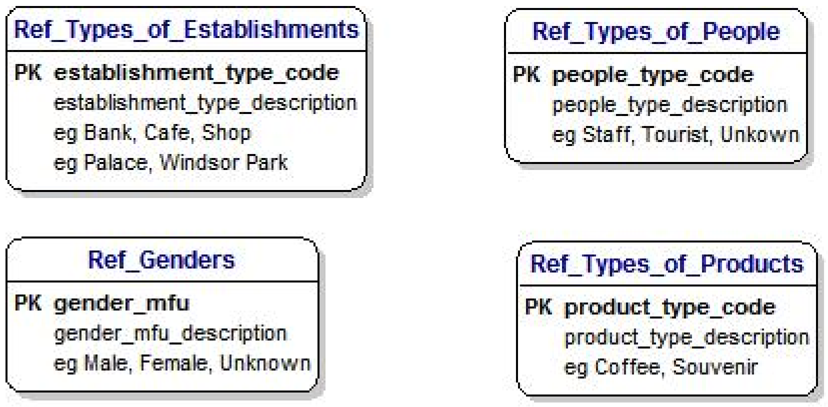


1.15 Reference Data

[Toby]: Dimple, you can see that I am using a Gender Table and People Types Table.
I have given them both names that begin with 'ref_' to make it clear that they are reference data.
This means that the values don't change much and I can use them to define what the valid values can be.
This is a technique that professional data modelers use but we don't need to worry about it today.

[Dimple]: I'm glad to hear it, Toby!
Although it isn't difficult to understand and it seems like a good idea.

[Toby]: In our small example, we have only four kinds of reference data altogether - gender, types of establishment, people and products.



1.16 Bringing it all Together

[Toby]: Dimple, if we bring together everything we have talked about, we will see that we have quite a good data model that any professional would be proud of.

[Dimple]: OK, Toby. Do you think I will understand it?

[Toby]: Let me help you by making a list of the **business rules** for our model:

- People can be either ceremonial guards, staff or tourists.
- There are a number of establishments of different types.
- Tourists can make visits to establishments and make purchases.
- Staff assist the tourists when they make a purchase.
- A purchase involves one product.

[Toby]: OK, Dimple - we have a very nice data model and now we can take the break I promised you.

[Dimple]: That's great, Toby - can I have an ice cream?

[Toby]: Sure, but before we do I should say something about **PF**, which appears in the Staff Table.

It's unusual and it's called **PF** because it means a field which is a **P**rimary Key in the Staff Table and a **F**oreign Key to the People Table.

[Dimple]: Hmmm, I've got a headache, Toby - can we please go and get an ice cream?

[Toby]: OK, Dimple. You've been a very good girl and you deserve a break.

You can admire what we have created, which is this very professional-looking data model.

1.17 Top-Level Model with Names Only

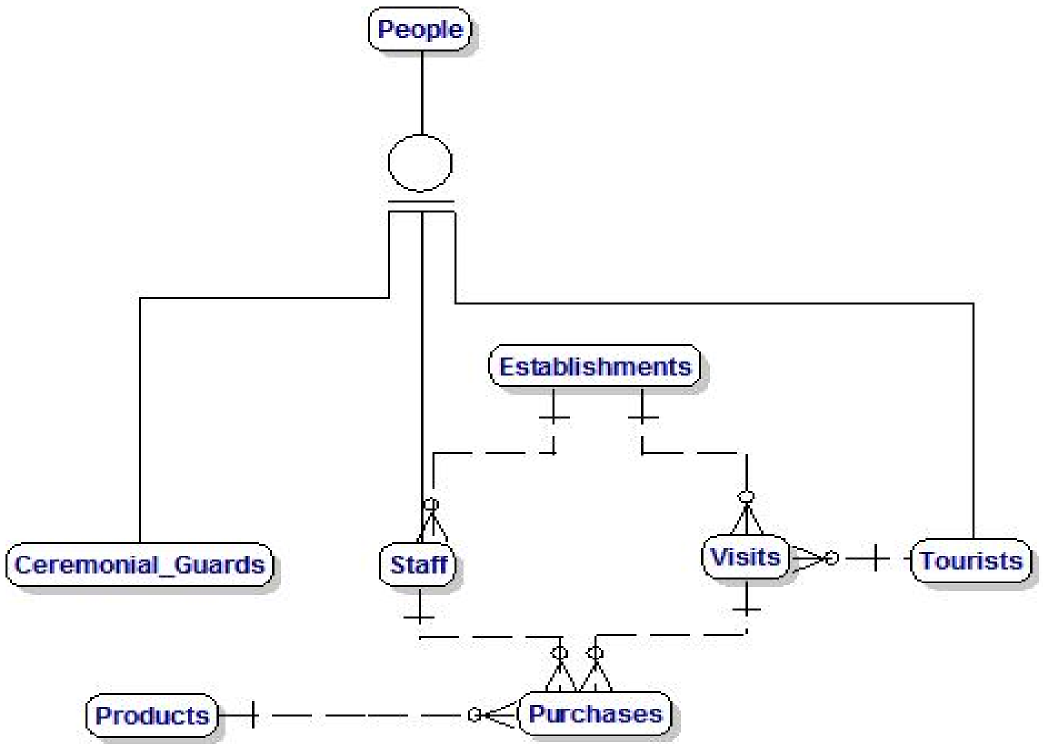
We can show our data model at the top-level, showing only the names of the 'things of interest,' which we call entities or tables if we are thinking about a database.

This is suitable for explaining what we saw in Windsor to our family or friends.

If we wanted to describe it, we could simply say:

- There are lots of people in Windsor, including ceremonial guards, staff and tourists.
- There are also lots of establishments, like shops and the Castle.

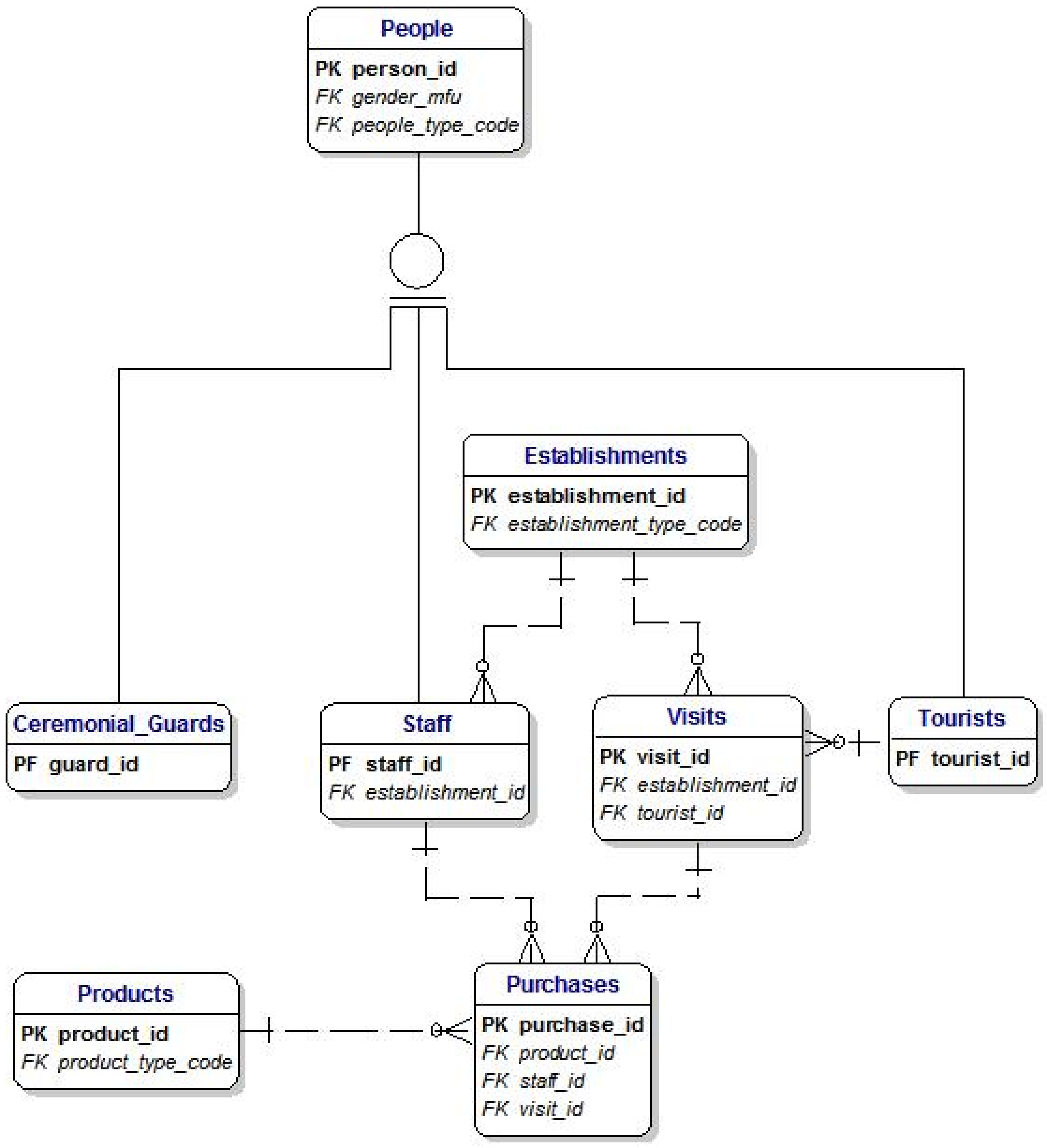
- Tourists made visits to establishments where they made purchases of products.



1.18 Top-Level Model with Key Fields

This is what our data model looks like if we show key fields only and leave out the Reference Data Tables.

This level of display is suitable if we want to confirm to each other how the tables (or entities) are related.

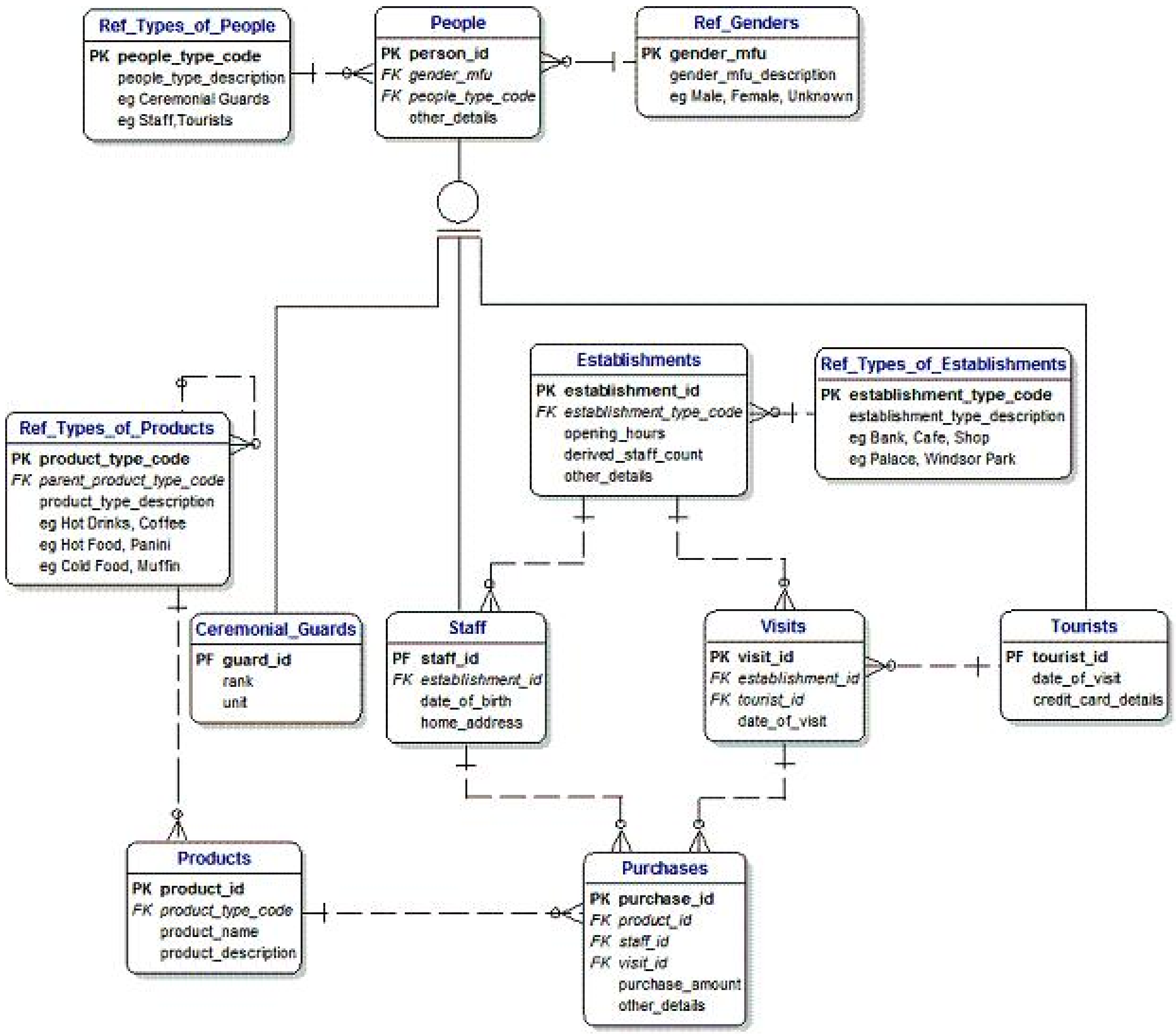


1.19 Top-Level Model with all Details

Finally, this is what our data model looks like if we show the key fields, all the data items only and the Reference Data Tables.

You can see that the amount of detail involved makes it more difficult to understand what's going on and to identify what is important.

This level of display is suitable if we want to talk about details and develop a database from our data model.



1.20 Ice Cream

[Toby]: Dimple, I've got some wonderful news for you.

[Dimple]: I'm glad to hear it, Toby - what is it?

[Toby]: I have found your favorite Baskin-Robbins ice cream here in Windsor ;)

[Dimple]: Toby, are you teasing me?

[Toby]: No, Dimple - look, there it is across the road from Windsor Castle!

[Dimple]: Wow - that's great, so I can have my favorite butter pecan ice cream.

1.21 What have we learned?

In this chapter, we have learned how to think like a data modeler and how to gradually put together a data model in our heads.

We know that if we get in the habit of doing this regularly it gets easier and more natural and soon we will be seeing the world around us as pieces of a data model that we can fit together like a jigsaw puzzle.

1.22 Please Email Me

I hope you have found this tutorial fun and useful.

I would be very pleased to have your comments – do you like this chapter or are there any changes you would recommend?

You can email me at barryw@databaseanswers.org.

2. Data Modeling in Denmark

Frederiksborg Castle in Denmark

We have moved this Chapter to a separate document that you can download here :-

- http://www.databaseanswers.org/downloads/Chapter_2_Learn_Data_Modelling_Book_for_Denmark.pdf

3. Data Modeling in Turkey

3.1 Introduction

In this tutorial, we will follow two young tourists as they visit Turkey, which is a country with a tremendous history and very popular with tourists looking for something special.

Our tourists are Dimple, a 10-year old girl, who likes sightseeing and ice cream and Toby, Dimple's 12-year-old brother, who likes sightseeing and designing data models.

3.1.1 What is this?

This is a tutorial on data modeling for young people that represents a typical data modeling project and illustrates the basic principles involved.

3.1.2 Why is it important?

Data modeling is important because it is the foundation for so many activities:

- It provides a vehicle for communication among a wide variety of interested parties, including management, developers, data analysts, DBAs and more.

- A physical database can easily be generated from a data model using a commercial data modeling tool.

3.1.3 What Will I Learn?

You will learn:
- How to create a data model, starting from scratch
- The important design principles involved
- What a typical data model looks like

3.2 Topics

In this chapter, we will cover some basic concepts in data modeling:
- Primary and Foreign Keys
- One-to-Many and Many-to-Many Relationships
- Hierarchies and Inheritance
- Reference Data

3.3 Let's get started

[Toby]: We have just arrived in Turkey. What would you like to do today?

[Dimple]: Toby, It's great being in Turkey which is so exciting and has so many things to see and enjoy.

[Toby]: I'm glad you like it, Dimple. What would you like to do today?
[Dimple]: Toby, we have come to Turkey, and I would like to see Istanbul and visit the Blue Mosque, because it's one of the most popular tourist attractions here, then I would like to do some shopping, then see the sea, and I would like to finish up at Starbucks.

[Toby]: OK. Let's go.

We are starting from Istanbul, which is a beautiful place…

Toby and Dimple leave England and arrive in Turkey…

3.4 Arriving at Istanbul

[Dimple] Wow, Toby, look at all these people.

[Toby] Yes, Dimple, when we look around there are so many people, shops, banks and so on!

So we can start thinking about our data model.

3.5 Starting our Data Model

[Dimple]: How do we get started?

[Toby]: Well, we know that we have people and places.
The simplest start is to call all these places **establishments**.
Then we simply have different kinds of establishments.

And we have people - local people, tourists, students, people passing through, people working here, people here on business and so on.

[Dimple]: Hmmm - so how do we translate what we know to help us get started with our data model?

[Toby]: Let’s start a diagram with people and establishments.

This simple diagram is going to grow into a data model.

People
Establishments

3.6 Identifiers and Primary Keys

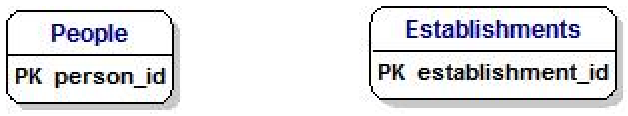
[Dimple]: Toby, I am one of these people so how do I create a unique identity for myself to make me different from everybody else?

[Toby]: We will give every person a **unique identifier** and every establishment its own unique identifier.

When we use these we call them **Primary Keys**, and show them in the diagram with a **PK** on the left-hand side.

[Dimple]: That sounds good, Toby, but I don't know what it means.

[Toby]: Well, Dimple, let's look at how we use these identifiers...



We have managed to find a quiet area where a very happy man is selling a Turkish favorite, called SIMIT ;0)

So, in other words, we have one person, who is the happy man, and one establishment, which is his simple stall.

So we can create a people record with a person ID of 1 and an establishments record for the stall, with an establishment ID of 3.

3.7 Relationships and Foreign Keys

[Toby]: Dimple, now we can add some interesting details because we know that one person can visit many establishments.
We also know that one establishment is visited by many tourists.
Then we call this a **many-to-many relationship** between people and establishments.

To make it easier for you to understand I have expanded the **many-to-many relationship** into two different things, which are called **one-to-many relationships**.

[Dimple]: So Toby, is that like saying that one person can make many visits to many establishments?

[Toby]: Yes, Dimple - that's great - and we can also say that one establishment can have visits from many people.

At this point, we can show how all these boxes are related, and that is a very big step, because it takes us to the idea of 'relationships'.

We can call these boxes **tables** - or *entities* if we want to speak to professional data modelers.

A table simply stores data about one particular kind of 'Thing of Interest'.

For example, people or establishments.

Each record in a table will be identified by its own unique identifier, which we call the *Primary Key*.

It is not usually easy to find a specific item of data already in the table that will always be unique.

For example, in the States, Social Security Numbers are supposed to be unique, but (for various legitimate reasons) that is not always the case.

Also, foreign visitors and tourists will not have SSNs.

Therefore, it is best practice to create a new field just for this purpose.

This will be what is called an **auto-increment** data type, which will be generated automatically by the Database Management System (DBMS) at run-time.

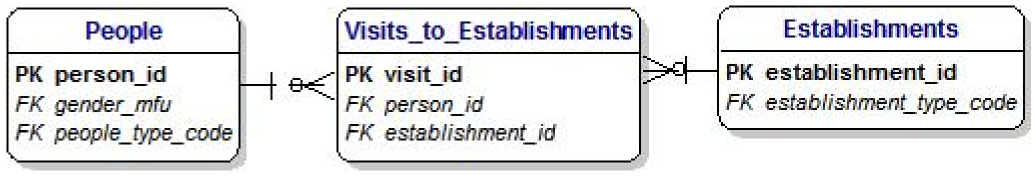
This is called a **surrogate key** and it does not have any other purpose.

It is simply a key that stands for something else.

It is a meaningless integer that is generated automatically by the database management software, such as Oracle or SQL Server, The values are usually consecutive integers, starting with 1,2,3,4 and so on.

Now we can see how useful our identifiers can be because we can include the person and establishment identifiers in our visits table.

Then the Person_ID field becomes a link to a record for a person in the Person Table. This link is what is called a **Foreign Key** and we can see it's shown with '**FK**' on the left-hand side.



3.8 Products and Product Types

[Dimple]: Toby, when we go into a shop we want to buy something.

And there are thousands and thousands of possibilities.
How do we deal with all that in our little data model?

[Toby]: Well Dimple, it's really quite easy. It's like all our modeling where we look for simple patterns that cover many situations.

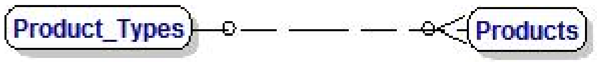
[Dimple]: Hmm - I don't know what that means. Maybe if you showed me I might understand it.

[Toby]: OK.
Everything that we buy is called a product, and all we have to do is simply define the type of each product - such as a coffee, muffin or a newspaper.

Then we draw a little box called *Products* and say that every product has a type.
In other words, there is a relationship between the *Products* and *Product_Types* boxes.

The lines are called **relationships** and they are very important in data modeling.
We are now creating an **Entity-Relationship Diagram** or '**ERD**'.

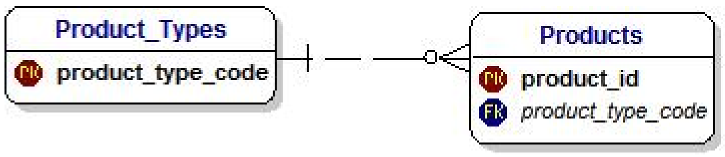
This diagram shows only a line for the relationship:



The symbol at the products end is called *crow's feet* and it shows the *many* end.

The short straight line at the Product_Types end shows the *one* end.

In other words, this line shows a one-to-many relationship.



Dimple, let me explain about the dotted line. It means that the relationship results in a foreign key in the Products Table. This is shown by the 'FK' symbol next to the ***product_type_code*** field and it means that there is a link back to the Product_Types.

However, the primary key is only the Product_ID, and of course, this is shown by the 'PK' symbol next to the **Product_ID** field.

Later, when we talk about inheritance, we will use a straight line, in contrast to this dotted line here. This is to show that the foreign key field is also a primary key.

I have to say something a bit difficult about primary keys right now.

In the Products Table, we have to allow for a very large number of products being stored.

Therefore we use an ID field for the Primary key.

We then create this ID field automatically as a number (called an auto-increment integer).

This number has no meaning and is simply used to identify each record uniquely among possibly millions or hundreds of millions.

However, things are different for **type** fields.

These are what we call enumerated data and are typically reference data.

They are always relatively small in number and we choose a code for the primary key because we can create them and review them manually.

It also helps us to create a code that we can use and refer to, in contrast to the ID fields that have no meaning.

Typical examples would be:

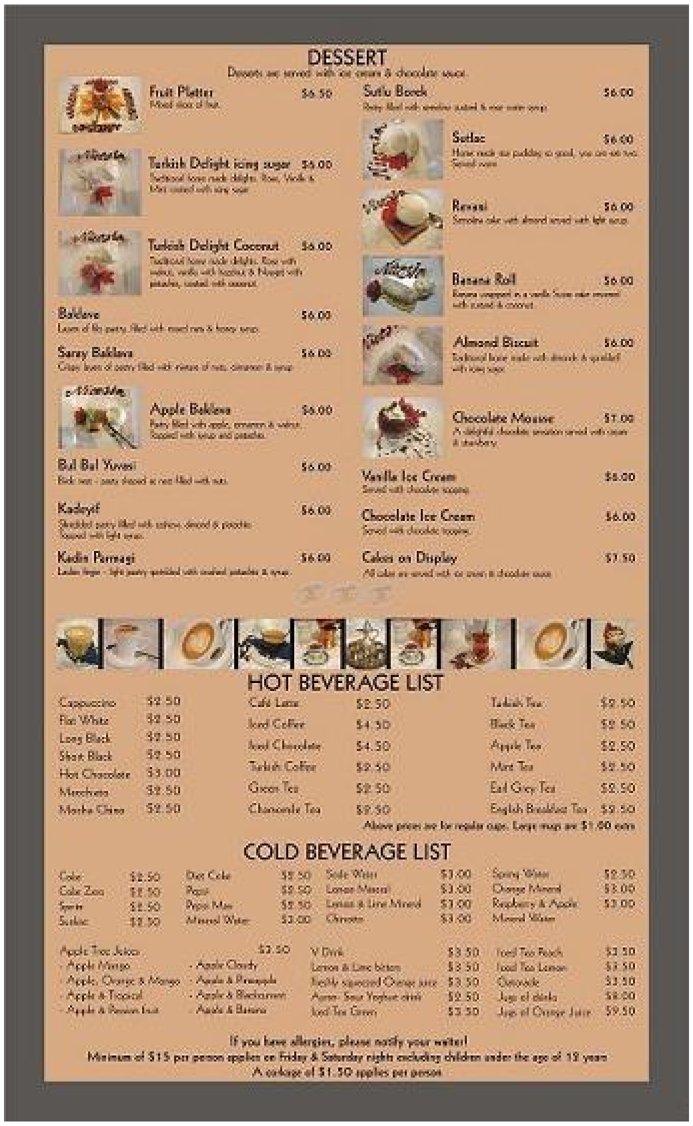
- Sizes – Small, Medium and Large where we are accustomed to seeing S,M and L.
- Gender – Male and Female, where we use M, F and U for Unknown.

- This menu board shows a typical menu in a Turkish restaurant that serves a wide range of food and drink.

We can see that they are organized in groups, like desserts and hot and cold drinks, and each of these has products, like apple baklava or turkish coffee.

This top-down organization is called a **hierarchy** and appears all over the place in our world.

Luckily we can show this very easily and neatly in our data model.



DESSERT

Desserts are served with ice cream & chocolate sauce.

Fruit Platter Mixed slices of fruit.	$6.50	**Sutlu Borek** Pastry filled with semolina custard & rose water syrup.	$6.00
Turkish Delight icing sugar Traditional home made delights. Rose, Vanilla & Mint coated with icing sugar	$6.00	**Sutlac** Home made rice pudding so good, you can eat two. Served warm	$6.00
Turkish Delight Coconut Traditional home made delights. Rose with walnut, vanilla with hazelnut & Nougat with pistachio, coated with coconut	$6.00	**Revani** Semolina cake with almond served with light syrup.	$6.00
Baklava Layers of filo pastry filled with mixed nuts & honey syrup.	$6.00	**Banana Roll** Banana wrapped in a vanilla sponge cake covered with custard & coconut	$6.00
Saray Baklava Crispy layers of pastry filled with mixture of nuts, cinnamon & syrup	$6.00	**Almond Biscuit** Traditional home made with almonds & sprinkled with icing sugar	$6.00
Apple Baklava Pastry filled with apple, cinnamon & walnut. Topped with syrup and pistachio.	$6.00	**Chocolate Mousse** A delightful chocolate sensation served with cream & strawberry.	$7.00
Bul Bul Yuvasi Birds nest - pastry shaped as nest filled with nuts.	$6.00	**Vanilla Ice Cream** Served with chocolate topping.	$6.00
Kadeyif Shredded pastry filled with cashew, almond & pistachio. Topped with light syrup.	$6.00	**Chocolate Ice Cream** Served with chocolate topping.	$6.00
Kadin Parmagi Ladies finger - light pastry sprinkled with crushed pistachio & syrup.	$6.00	**Cakes on Display** All cakes are served with ice cream & chocolate sauce	$7.50

HOT BEVERAGE LIST

Cappuccino	$2.50	Café Latte	$2.50	Turkish Tea	$2.50
Flat White	$2.50	Iced Coffee	$4.50	Black Tea	$2.50
Long Black	$2.50	Iced Chocolate	$4.50	Apple Tea	$2.50
Short Black	$2.50	Turkish Coffee	$2.50	Mint Tea	$2.50
Hot Chocolate	$3.00	Green Tea	$2.50	Earl Grey Tea	$2.50
Macchiato	$2.50	Chamomile Tea	$2.50	English Breakfast Tea	$2.50
Mocha Chino	$2.50				

Above prices are for regular cups. Large mugs are $1.00 extra

COLD BEVERAGE LIST

Coke	$2.50	Diet Coke	$2.50	Soda Water	$3.00	Spring Water	$2.50
Coke Zero	$2.50	Pepsi	$2.50	Lemon Mineral	$3.00	Orange Mineral	$3.00
Sprite	$2.50	Pepsi Max	$2.50	Lemon & Lime Mineral	$3.00	Raspberry & Apple	$3.00
Sunkist	$2.50	Mineral Water	$3.00	Chinotto	$3.00	Mineral Water	

Apple Tree Juices	$2.50	V Drink	$3.50	Iced Tea Peach	$3.50
- Apple Mango / - Apple Cloudy		Lemon & Lime bitters	$3.50	Iced Tea Lemon	$3.50
- Apple, Orange & Mango / - Apple & Pineapple		Freshly squeezed Orange juice	$3.50	Gatorade	$3.50
- Apple & Tropical / - Apple & Blackcurrant		Ayran - Sour Yoghurt drink	$2.50	Jugs of drinks	$8.00
- Apple & Passion fruit / - Apple & Banana		Iced Tea Green	$3.50	Jugs of Orange Juice	$9.50

If you have allergies, please notify your waiter!

Minimum of $15 per person applies on Friday & Saturday nights excluding children under the age of 12 years

A corkage of $1.50 applies per person

3.9 Products, Types and Product Hierarchies

[Dimple]: Toby, when we look closely at the menu to try to decide what to order we can see lots of possibilities

But after a while we can see a pattern that helps us decide.
How do we deal with all that in our little data model?

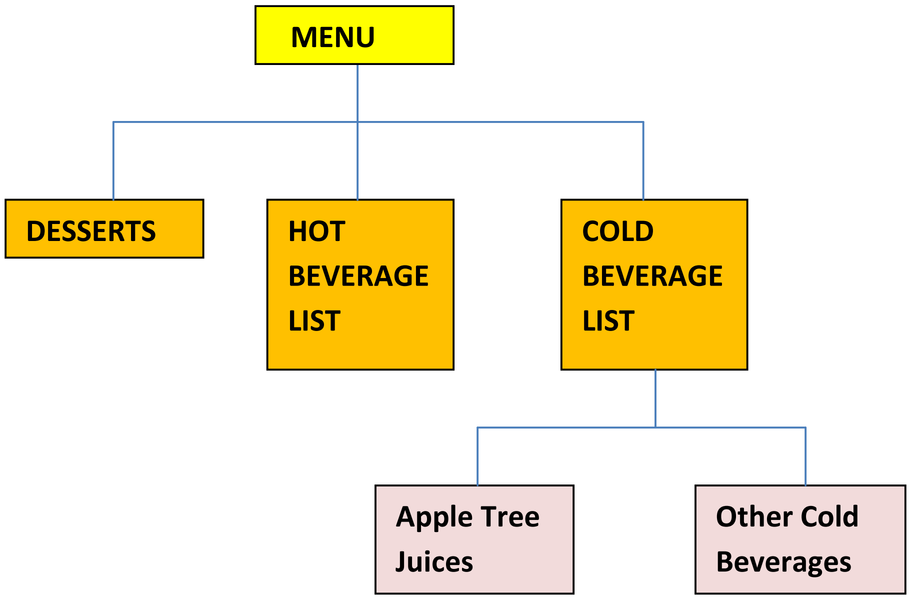
[Toby]: Well Dimple, it's really quite easy.
We define something called a **hierarchy**.
Hierarchies are very common and simply mean any situation where there are parents, children, grandchildren and so on.
If we look at this menu board at the top we can see headings saying 'Desserts' and below 'Hot and Cold Beverages'.

So in this case, the top-level of our hierarchy is 'Food and Drink'.

'Food' has just one category, which is 'Desserts'.

'Drink' has two categories, which are 'Cold Drinks' and 'Hot Drinks'.

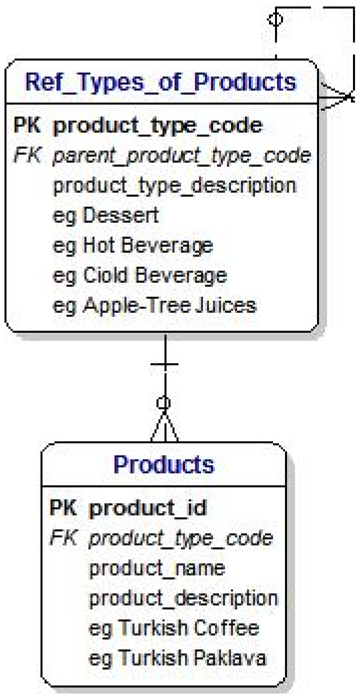
Then 'Cold Drinks' has two categories, which are 'Apple-Tree' and 'Other'.



[Dimple]: I think I understand that, it sounds OK.

[Toby]: Finally, we show this hierarchy by a dotted line in the top-right hand corner in the entity called 'Ref_Types_of_Products'.

This is formally called a *Recursive* or *Reflexive* relationship and is informally called **rabbit ears**.



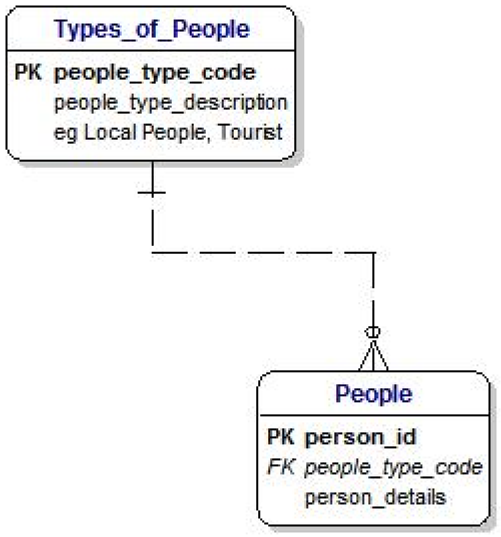
3.10 Types of People

[Dimple]: Toby, that looks OK.
I guess we can deal with types of people the same way, can we?

[Toby]: Yes, Dimple, and types of establishments as well.

[Dimple]: OK, that sounds sensible. And do they use these identifiers in a database?

[Toby]: Yes, and what is even better is that the database will automatically generate a new unique identifier for you and your visits and purchases if you want to get a refund later.



3.11 Types of People and Establishments

[Dimple]: Toby, that looks OK.
I guess we can deal with types of establishments the same way, can we?

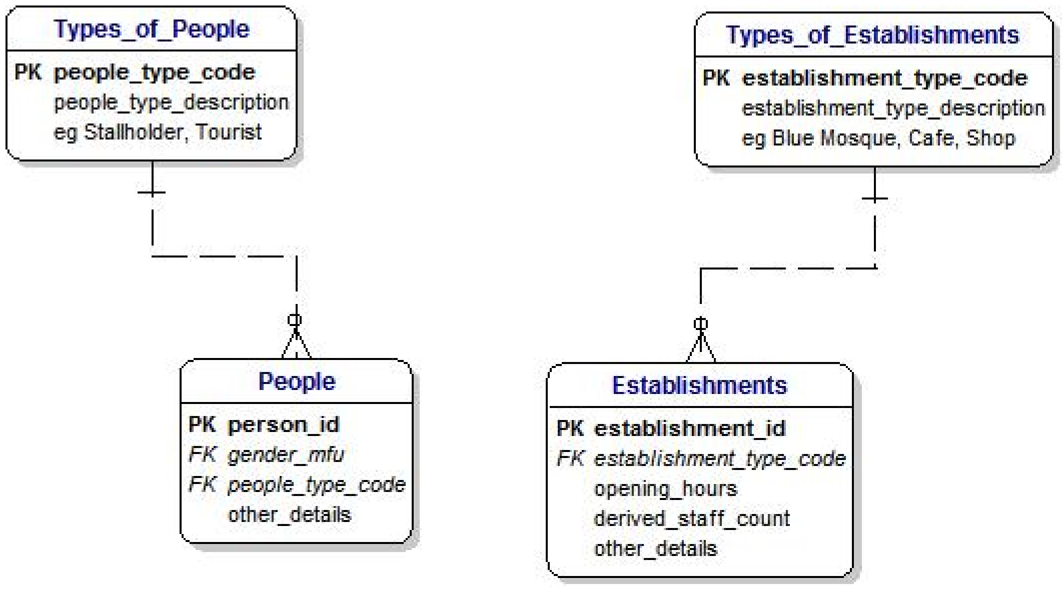
[Toby]: Yes, Dimple.

[Dimple]: OK, that sounds sensible. And do they use these identifiers in a database?

[Toby]: Yes, and we can use our new unique identifier for you and your visits and purchases in case we want to keep track of things.

Like maybe you want to get a refund later so we need to get your details from the database.

[Toby]: Before we move on, let's talk about establishments.
In Turkey, there are many different kinds of establishments, like shops, banks, cafes, restaurants, hotels, hospitals, garages and so on.
But when we think about these things, we find that we can simply fit them into our definition of establishments and identify them as different types of establishments.



3.12 Visits and Purchases:

Here we can see two visitors taking to a stallholder in a Bazaar.

[Dimple]: Toby, with so many tourists, stalls, shops and things to buy, how do we keep track of everything?

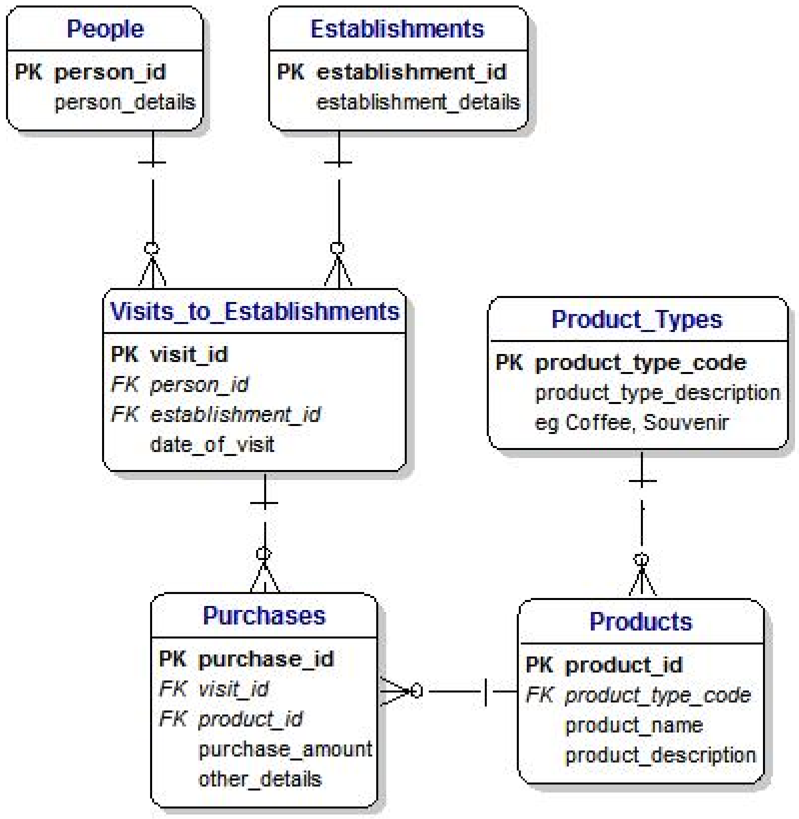
[Toby]: Well, Dimple, by this time, everything has its own identifier that is used wherever they need to keep track.

[Dimple]: OK, that sounds sensible. And do we use these identifiers in a database?

[Toby]: Yes, Dimple, and in this diagram, we can see that we can use the unique identifiers that are shown as 'PK,' for primary keys.

We can see that we have a PK for every entity or table so we can be pretty sure we can get from any table to any other table.

This is called *navigating* around the data model and is a good test for a well-designed data model.



3.13 People and Inheritance

[Toby]: Dimple, let's take a closer look at the different types of people we can find in Istanbul.

[Dimple]: OK, Toby. I hope I don't have to think too much because I might get a headache?

[Toby]: No, Dimple, I will do the thinking and talking and all you have to do is nod your head when you understand.

[Dimple]: OK, Toby. I promise to do that.

[Toby]: We already said that we have local people and tourists.

There are always lots of people visiting the Blue Mosque.

When we look at this typical street scene, we can see shoppers, stallholders, workers and local people.

We usually know different things about the stallholders and workers than the things we know about the tourists.

For example, we will probably know the gender of everybody just by looking at them.
For workers, we might also know things related to their employment, such as their date of birth and their home address.

In data modeling we have a very powerful approach that we call **Inheritance** that we can use here.

If we want to describe this in English, we would say that staff inherit the People_Type_Code and gender from the parent entity of people, and in addition, they have a date of birth and home address.

For tourists, we don't know much, except for the date of their visit, and maybe, if they buy something in a shop using a credit card, then the shop would know the credit card details.

Does that make sense, Dimple?

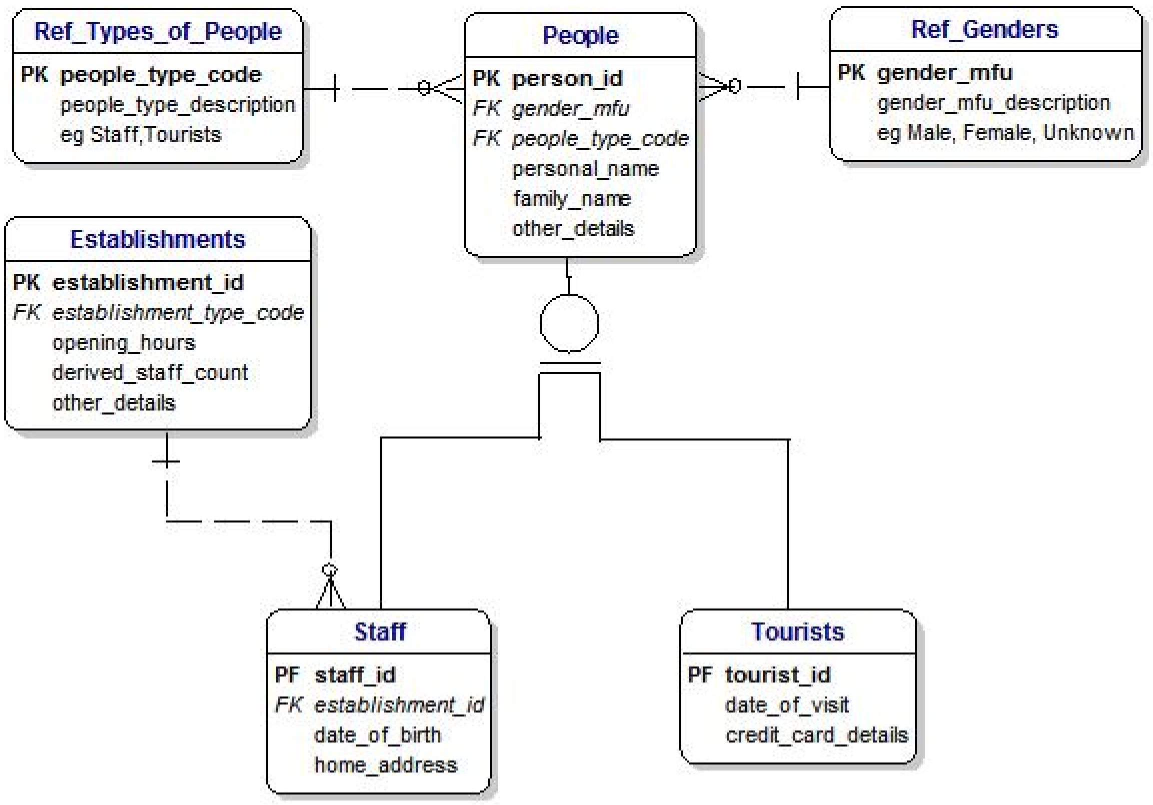
[Dimple]: I think so, Toby.
Is it like saying that we inherit having two arms and two legs from our parents because they have two arms and two legs, but that we have also have things that are just us?

[Toby]: Yes, Dimple - that's great - let's take a break and do some shopping!

[Dimple]: I like the sound of that, Toby. Can I have an ice cream?

[Toby]: Yes, of course, Dimple – this diagram shows we are doing well.

It shows inheritance between people and the two different types of people:



We can see a field marked as ‘PF’ in the tables for staff and tourists.

This is unusual because it means a field that is a **P**rimary Key in the three tables and also a **F**oreign Key to the People Table.

Therefore, if your first record was a member of staff, then we would have a record in the People Table with a Person_ID of 1 and a record in the staff table with a Staff_ID of 3.

Similarly, if our second record was a tourist, we would have a record in the Person Table with a Person_ID of 2 and a record in the tourist table with a Staff_ID of 3.

3.14 Staff, Establishments and Derived Fields

[Dimple]: Toby, how do we specify that staff must work in some establishment?

[Toby]: Dimple, that's a very good question.
Fortunately, the answer is very easy.
We add a one-to-many relationship between the staff and establishment entities
In English, we would say that every member of staff must work in one establishment and every establishment can employ many members of staff.
In the diagram, we show this with a **Foreign Key** by the Establishment_ID field in the staff entity.
So if we look closely at the staff entity, we will see **'FK'** by the Establishment_ID field.

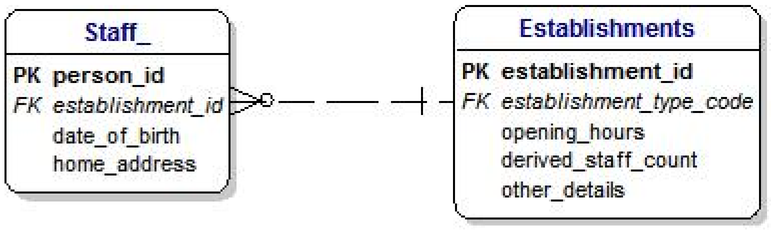
[Dimple]: OK, that sounds good, and I can see how the identifiers are very important.

[Toby]: I am glad to hear it, Dimple.
There is one more thing I have to say.
We are learning data modeling and one important thing about data modeling is that it has to follow a set of **rules**.
These rules help us to produce good data models and so they are very important.
One of the rules is that we cannot include any bits of data that can be derived from any other bits of data.
For example, we usually want to know how many people work in a shop or cafe.
Therefore we include a **staff Count** field with the establishment.
But when it comes to finding the value that goes in here, we will count the records in the Staff Table for each establishment.
Therefore, it's a **derived Field** and we call it a name that starts with 'derived_' to make things clear.
This is because, according to the rules, we should not include derived fields in our data model at this early stage.

I have shown it here simply as an example because it is a situation that occurs quite often so it’s good to recognize it when you see it.

Does that sound sensible, Dimple?

[Dimple]: I suppose so, Toby. But I've got a headache, can we go to Starbucks now?



3.15 Reservations and Generic Data Models

[Toby]: Dimple, this bit is quite hard-going so if you want to take a rest, that's OK.

[Dimple]: OK, Toby, I will just sit quietly and watch the people ;0)

[Toby]: People make reservations every day all around the world.

These reservations have a lot in common:

- Hotel bookings, airline bookings, theatres and shows, appointments to see a doctor or dentist and so on.
- The basic common things are a date and time, usually a specific facility, like a hotel, an airline seat, a theatre and so on.

This means that we can identify what they have in common and what they have that is different and specific to the type of appointment.

3.12.1 Reservations for a Hotel
Here is a very beautiful and unique hotel in the caves at Yunak Evleri (about 400 km from Istanbul), which has rooms dating back to the 5th century.

For a hotel, of course, you would book for a specific night (or night) and maybe a non-smoking room but that is about all.

Hotel in the Caves at Yunak Evleri, Turkey

3.12.2 Reservations for Whirling Dervishes

A very unusual spectacle that is unique to Turkey is the sight of Whirling Dervishes dancing.

It is part of the Muslim religious practice of the disciples

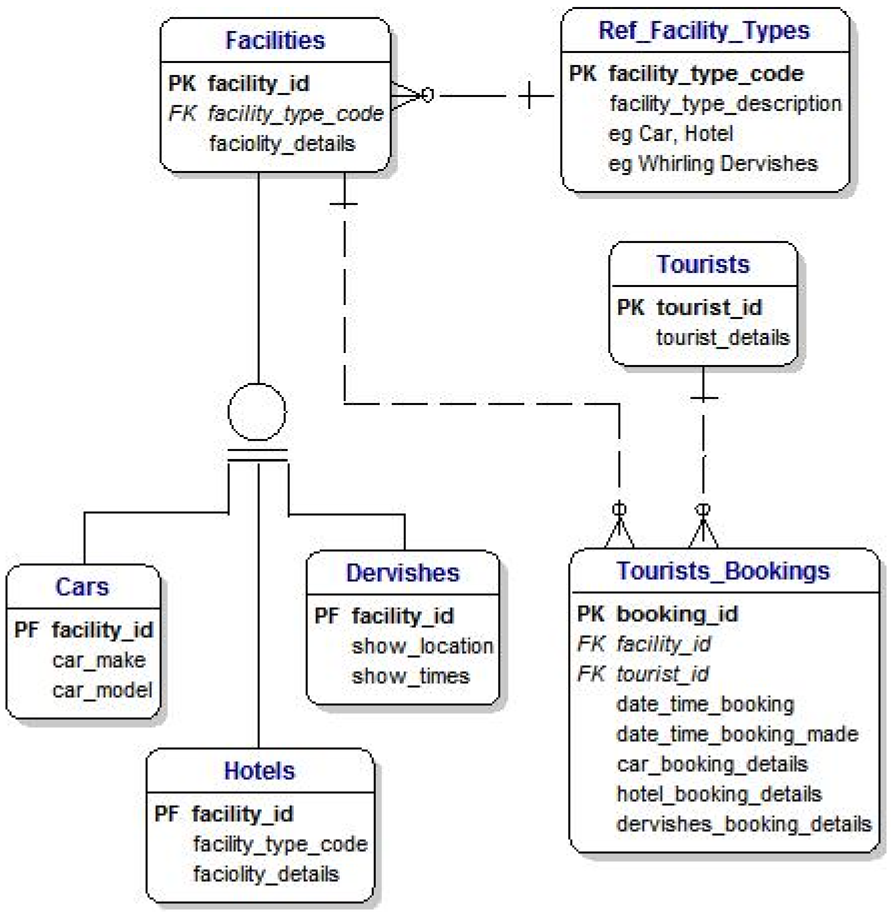
For this reservation, you would book for a specific show and a seat at the price you wanted.

3.12.3 Generic Data Model for Reservations

In this model, we define a facility to be what we are making a reservation for.

This data model is shown on this page of our Database Answers Web site:

- http://www.databaseanswers.org/data_models/generic_reservations/generic_reservations_inheritance_for_turkey.htm



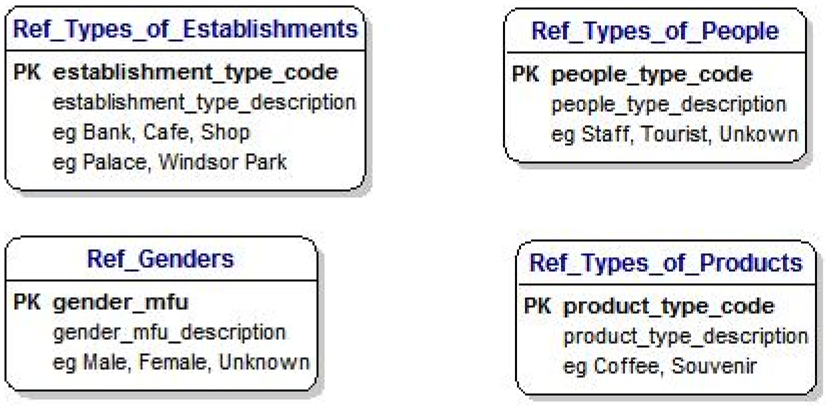
3.16 Reference Data

[Toby]: Dimple, you can see that I am using a Gender Table and People Types Table.
I have given them both names that begin with 'ref_' to make it clear that they are reference data.

This means that the values don't change much and I can use them to define what the valid values can be.
This is a technique that professional data modelers use but we don't need to worry about it today.

[Dimple]: I'm glad to hear it, Toby!
Although it isn't difficult to understand and it seems like a good idea.

[Toby]: In our small example, we have only four kinds of reference data altogether - gender, types of establishment, people and products.



3.17 Bringing it all Together

[Toby]: Dimple, if we bring together everything we have talked about, we will see that we have quite a good data model that any professional would be proud of.

[Dimple]: OK, Toby. Do you think I will understand it?

[Toby]: Let me help you by making a list of the **business rules** for our model:

- People can be either staff or tourists.
- There are a number of establishments of different types.
- Tourists can make visits to establishments and make purchases.
- Staff assist the tourists when they make a purchase.
- A purchase involves one or more products.

[Toby]: OK, Dimple - we have a very nice data model and now we can take the break I promised you.

[Dimple]: That's great, Toby - can we go to Starbucks?

[Toby]: Sure, but before we do I should say something about **PF**, which appears in the Staff Table.

It's unusual and it's called **PF** because it means a field which is a **P**rimary Key in the Staff Table and a **F**oreign Key to the People Table.

[Dimple]: Hmmm, I've got a headache, Toby - can we please go to Starbucks?

[Toby]: OK, Dimple. You've been a very good girl and you deserve a break.

You can admire what we have created, which is this very professional-looking data model.

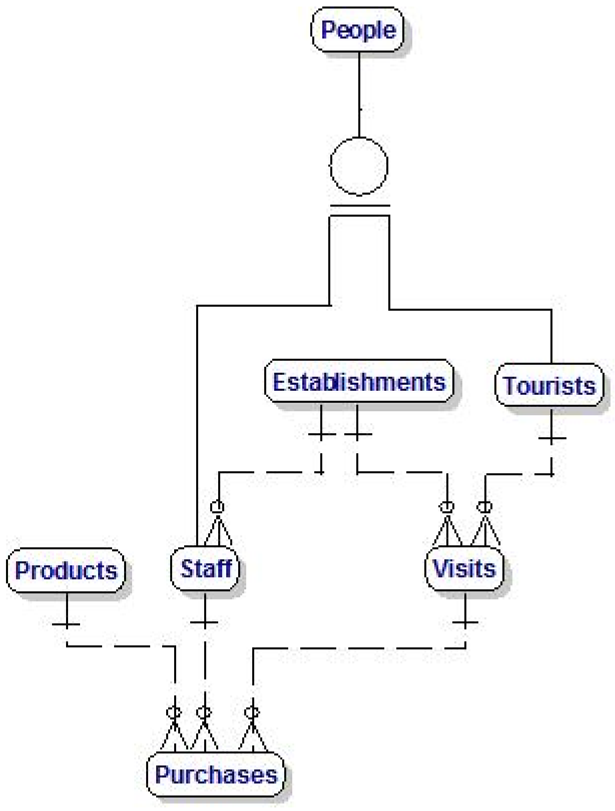
3.18 Top-Level Model with Names Only

We can show our data model at the top-level, showing only the names of the 'things of interest,' which we call entities or tables if we are thinking about a database.

This is suitable for explaining what we saw in Windsor to our family or friends.

If we wanted to describe it, we could simply say:

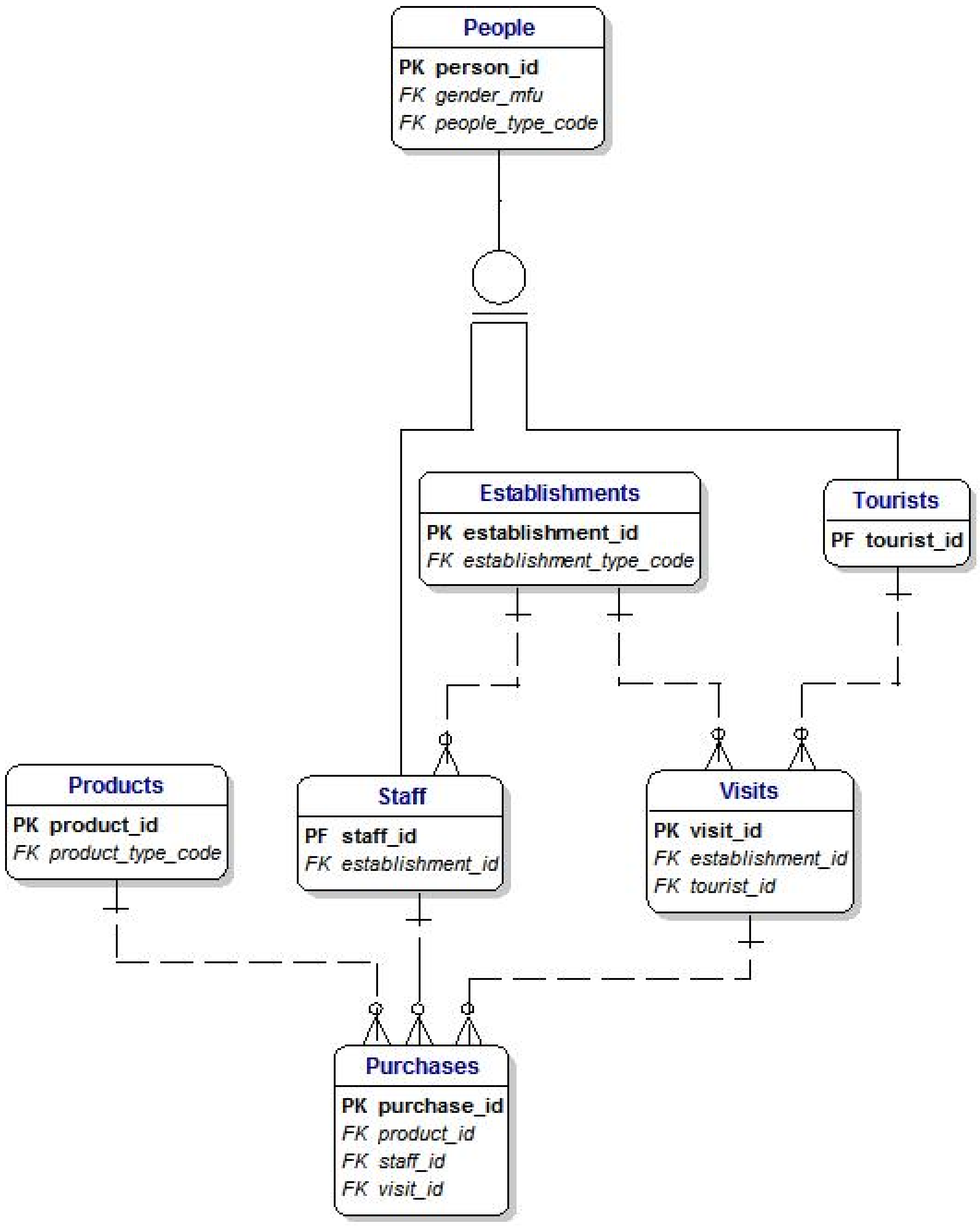
- There are lots of people in Windsor, including ceremonial guards, staff and tourists.
- There are also lots of establishments, like shops and the Castle.
- Tourists made visits to establishments where they made purchases of products.



3.19 Top-Level Model with Key Fields

This is what our data model looks like if we show Key fields only and leave out the Reference Data Tables.

This level of display is suitable if we want to confirm to each other how the tables (or entities) are related.

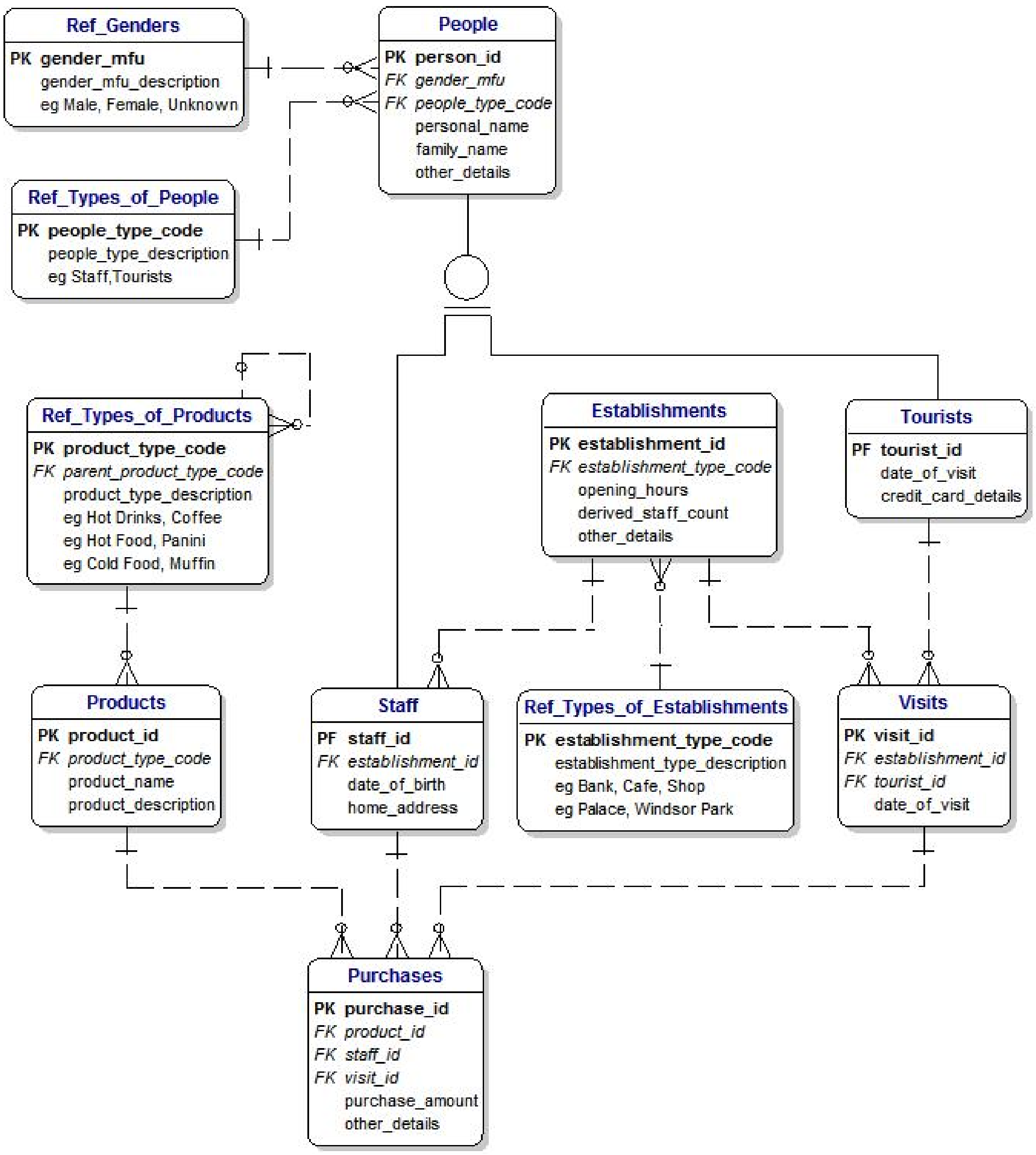


3.20 Top-Level Model with all Details

Finally, this is what our data model looks like if we show the key fields, all the data items only and most of the Reference Data Tables.

You can see that the amount of detail involved makes it more difficult to understand what's going on and to identify what is important.

This level of display is suitable if we want to talk about details and develop a database from our data model.



3.21 Starbucks

[Toby]: Dimple, I've got some wonderful news for you.

[Dimple]: I'm glad to hear it, Toby - what is it?

[Toby]: I have found Starbucks here in Turkey, so you can have your favorite things to eat or drink ;)

[Dimple]: Toby, are you teasing me?

[Toby]: No, Dimple - look, there it is across the road from the Blue Mosque!

[Dimple]: Wow - that's great, so I can have my favorite muffin.

3.22 What have we learned?

In this chapter, we have learned how to think like a data modeler and how to gradually put together a data model in our heads.

We know that if we get in the habit of doing this regularly it gets easier and more natural and soon we will be seeing the world around us as pieces of a data model that we can fit together like a jigsaw puzzle.

4. Some Basic Concepts

4.1 Introduction

This chapter discusses the basic concepts in data modeling.

It builds through a series of structured steps in the development of a data model.

This chapter covers the basic concept that provide the foundation for the data model that we designed in similar material to Chapter 1 but it is more serious and more comprehensive.

This material is also available as a tutorial for Amazon and Starbucks on the Database Answers Web site –

- http://www.databaseanswers.org/tutorial4_data_modeling/index.htm

We will cover these basic concepts:

a. Creating Entities
b. Primary and Foreign Keys
c. One-to-Many and Many-to-Many Relationships
d. Hierarchies
e. Inheritance
f. Reference Data

At the end of this tutorial, we will have produced a data model, which is commonly referred as an Entity-Relationship Diagram, or 'ERD'.

4.1.1 What is this?

This chapter is a description of the relational theory as originally established by Ted Codd, who, at the time, was a research scientist with IBM.

4.1.2 Why is it important?

The basic concepts are important because the relational theory is very powerful and provides a sound theoretical foundation for databases that have become essential since their first appearance in the early 1970s.

They were the creation of a brilliant research scientist called Ted Codd, who was working for an IBM Research Lab at the time. It is reported that he faced internal criticism initially because it was considered that his new idea would affect sales of established IBM database products.

It is the foundation for so many activities:

- It provides a vehicle for communication among a wide variety of interested parties, including management, developers, data analysts, DBAs and more.
- A physical database can easily be generated from a data model using a commercial data modeling tool.

4.1.3 What Will I Learn?

You will learn:

- How to create a data model, starting from scratch.
- What a typical data model looks like.

4.2 What is the Scope?

Our photo shows a typical Starbucks. If we look closely, we can see people eating, drinking and placing orders. What Starbucks sees are customers, products and orders being met.

During the course of this book we will see how data models can help to bridge this gap in perception and communication.

Getting Started:
The area we have chosen for this tutorial is a data model for a simple **Order Processing System** for Starbucks.

We have done it this way because many people are familiar with Starbucks and it provides an application that is easy to relate to.

We think about the area we are going to model.

We can see customers ordering products (food, drinks and so on).

Our approach has three steps:

1. Establish the scope of the data model.
2. Identify the 'things of interest' that are within the scope, These will be called entities.
3. Determine the **relationships** between them.

Deciding the Scope of Our Data Model
When we step inside, we see that Starbucks sells a wide range of products, so our first task is to decide which of them should be included in our data model.

Right now, we are interested only in something to eat and something to drink. Therefore, all the mugs and other items shown in this picture on the left, are outside the **scope** of our data model, and are not 'Things of Interest'.

4.3 What are the 'Things of Interest'?

Our first step is to decide what things are we interested in.

In other words, what is the scope of our data model?

Customers
Orders
Products

These things will be called 'Entities in a Data Model' and 'Tables in a Database'.

4.4 Creating Entities

Dezign is a data modeling tool that I use extensively because it is very good and very affordable.

You can download a free trial from this Web site:

http://www.datanamic.com

Here is a list of modeling tools on the Database Answers Web site:

http://www.databaseanswers.org/modelling_tools.htm

This is how you create an entity in the Dezign data modeling tool:
1. Right-click on a blank area in the diagram

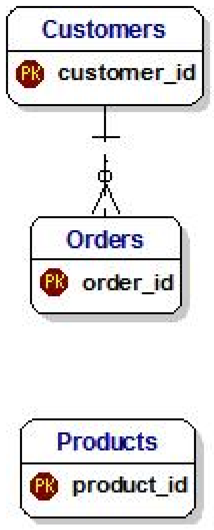
4. From the drop-down list, choose *Insert* and *Entity*
3. Check the *PK* box for the primary key attribute, which will usually be the first one on the entity.
7. Click on *Close* to save the results.

4.5 Primary Keys

We decide that the things we are interested in are customers, orders and products. You can buy a range of products in Starbucks, including souvenir mugs, coffee and newspapers.

For the purpose of our first model, we restrict our products to food and drink.

This diagram shows the corresponding entities with primary keys.



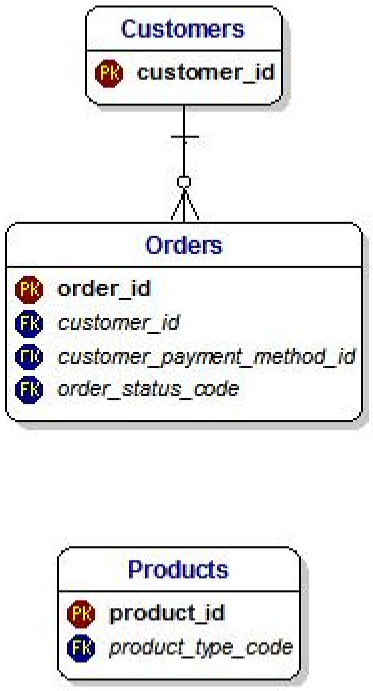
1. At this stage, we show only the entities with no relationships and minimum attributes and specify only the primary key and one *details* field that will be replaced later on.
2. The *Primary Key* field(s) should always be first.
3. You will notice that the first field in the Customers_version2 Table is the Customer_ID.
4. It has a *PK* symbol beside it, which indicates that it is the primary key for the table.
5. The primary key is very important and is the way that we can recognize each individual record in the table.

Creating a primary key in the Dezign tool:
1. Right-click on the *Entity*
3. Choose *Attributes*

6. Check the *PK* box for the primary key attribute, which will usually be the first one on the entity.
7. Click on *Close* to save the results.

4.6 Foreign Keys

This diagram shows entities with foreign keys.
Customer_ID is a foreign key that links orders to customers.



Here we have added the **relationships** between the entities.

When this primary key is used in another table, it is referred to as a *foreign key*.

We can see a good example in this diagram, where the Customer_ID appears in the Orders Table as a foreign key.

This is shown with an 'FK' symbol beside it.

Mandatory Key Fields

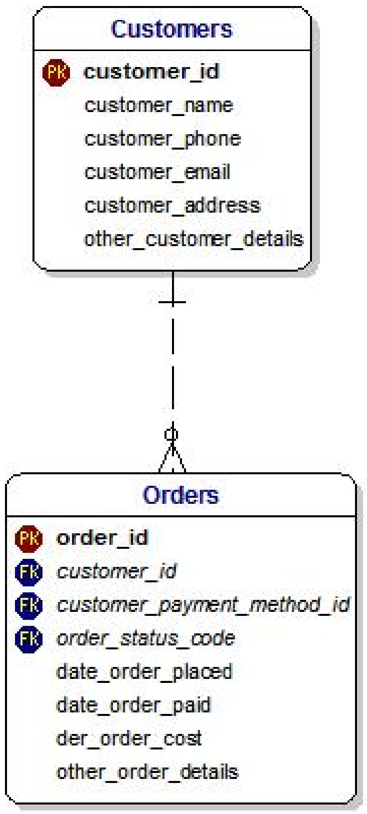
A foreign key is usually **mandatory.** For example, a value for a Customer_ID in the Customers_Payment_Methods Table must correspond to an actual value of the Customer_ID in the Customers_Version_1 Table. This is shown in the diagram by the short straight line at the end of the dotted line close to the Customers Table.

Foreign Keys in the Dezign Tool

Foreign keys are created automatically when you make a relationship between two entities. We recommend that you move the field up in the entity so that it takes it place alphabetically among the key fields.

To do this, right-click on the entity, choose the *Attributes* option, then click on the up or down arrow on the right-hand side.

4.7 One-to-Many Relationships



In this diagram, a customer can place zero, one or many orders.
This defines a one-to-many relationship.

This is shown by the symbol that has three small lines at that end of the relationship dotted line, which is referred to as *crow's feet*.

Optional Key Fields

Strictly speaking, a customer does not have to place an order. He or she could change their mind and walk out without ordering anything. In other words, we would say that the relationship is **optional** at the orders end. This is shown by the little *O* at that end of the relationship dotted line.

A data modeler would say "For every customer, there can be zero, one or many orders".

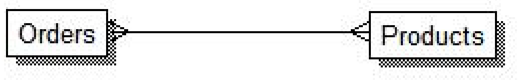
TERM	**DEFINITION**
Customer	Any unit that can raise a demand.
Demand	A request for assets to be supplied. The format of a request can be an electronic message, a paper form and so on.

Business Rules: A customer can raise zero, one or many demands.
: A demand must be associated with a valid customer.

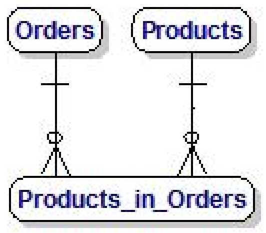
4.8 Many-to-Many Relationships

This diagram shows a many-to-many relationship between orders and products.
An order can include many products and a product can appear on many orders.

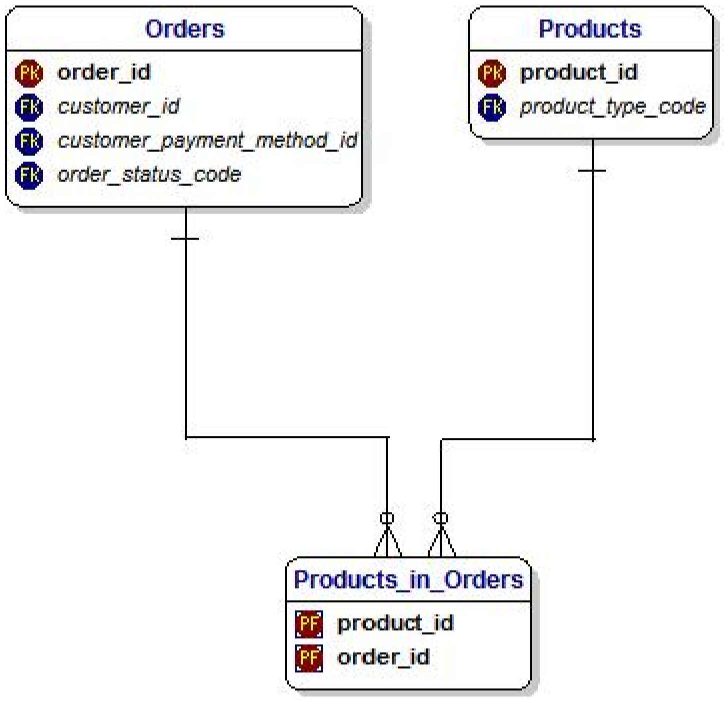
This defines a many-to-many relationship and is shown in a data model as follows:



A many-to-many relationship cannot be implemented in relational databases.
Therefore we resolve this many-to-many into two one-to-many relationships, which we show in a data model as follows:



Sometimes it is useful to see the key fields to ensure that everything looks alright.



When we look closely at this data model, we can see that the primary key is composed of the Order_ID and Product_ID fields.

This reflects the underlying logic, which states that every combination of order and product is unique.

In the database, this will define a new record.

When we see this situation in a database, we can say that this reflects a many-to-many relationship.

However, we can also show the same situation in a slightly different way, which reflects the standard design approach of using a surrogate key as the primary key and showing the demand and product IDs simply as foreign keys.

A surrogate key is simply a key that stands for something else.

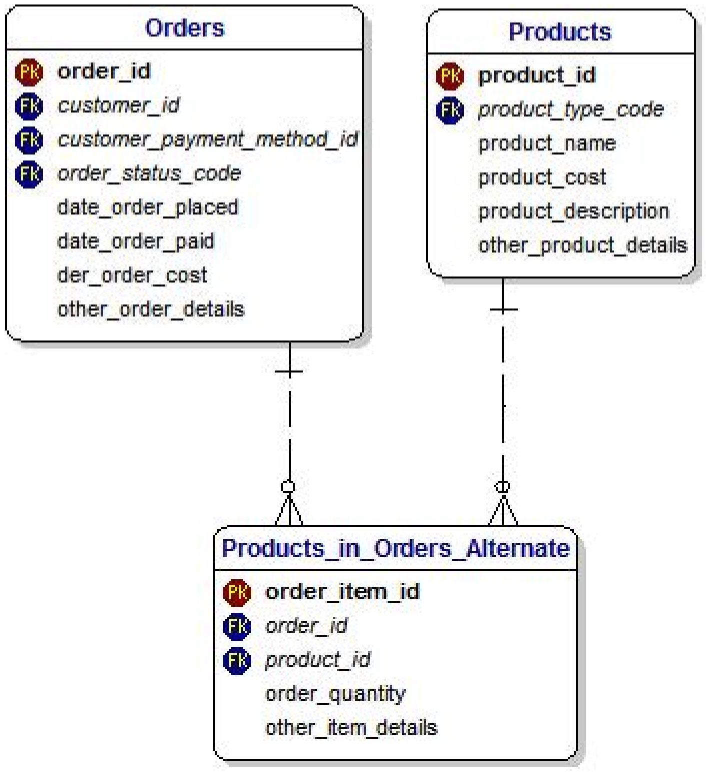
We use one when it is a better design or is simply more convenient.

It is a meaningless integer that is generated automatically by the database management software, such as Oracle or SQL Server, The values are usually consecutive integers, starting with 1,2,3,4 and so on.

The benefit of this approach is that it avoids the occurrence of primary keys with too many fields if more dependent tables occur where they cascade downwards.

The benefit of the previous approach is that it avoids the possibility of *orphan* records in the Products in a Demand Table.

In other words, invalid records that have invalid demand ID and/or product ID values.



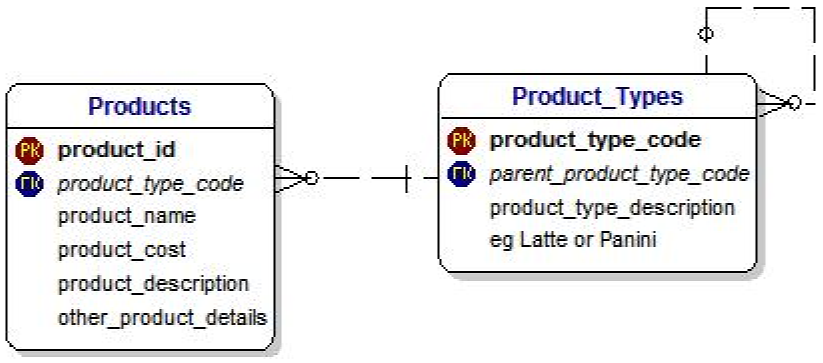
TERM	**DEFINITION**
Order	A request for products to be supplied. The format of a request can be verbal, an electronic message, a paper Form, etc.

Product	An Item that can be supplied on request. It can be something small, like a muffin, or something that contains other products, like a sandwich with multiple fillings.

Business Rules: An order can refer to zero or many products.
: A product can appear in zero, one or many orders.
: We can also say "An order can refer to many products and a product can appear in many orders".
: In other words, there is a many-to-many relationship between orders and products.

4.9 Hierarchies and Rabbit Ears

Hierarchies are very common and we can see them all around us.
Fortunately, we can handle them every easily in data models.



This diagram shows how the hierarchies of products and product types that we have just discussed are shown in our **Entity-Relationship Diagram**.

You will notice that the table called 'Product_Types' has a dotted line coming out on the right-hand side and going back in again on the top-right corner.

Data analysts call this a *recursive* or *reflexive* relationship, or informally, simply *rabbit ears*.

In plain English, we would say that the table is joined to itself and it means that a record in this table can be related to another record in the table. This approach is how we handle the situation where each product can be in a hierarchy and related to another Product.

For example, a product called Panini could be in a product sub-category called 'Miscellaneous Sandwiches' which could be a higher product category called 'Cold Food,' which itself could be in a higher product super-category called simply 'Food'.
Next time you go into Starbucks, take a look at the board behind the counter and try to decide how you would design the products area of the data model.

You should **pay special attention** to the little 'zeros' at each end of the dotted line. These are how we implement the fact that the Parent Product Type Code is optional, because the highest level will not have a parent.

This tutorial is also available on the Database Answers Web site:

- http://www.databaseanswers.org/tutorial4_data_modeling/index.htm

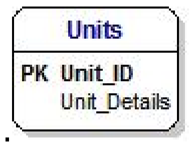
A number of data models show examples of inheritance, including:

- Charities
- City Tourist Guide
- CMDB - Configuration Mgt DB
- Customers Commercial and Personal
- Event Registrations
- Games Store
- Insurance Brokers
- Libraries for Lawyers
- National Trust (UK)
- New Egg
- Photo Catalogs
- School Management Systems
- Shrek 2 Movie
- Tracking Manufactured Items
- Travel & Tourism Worldwide
- Vehicle Imports

An Example in the Military

We start with the definition of a Unit, which at its simplest, looks like this:

In this case, we use a meaningless ID for the unit ID which is simply a unique number.



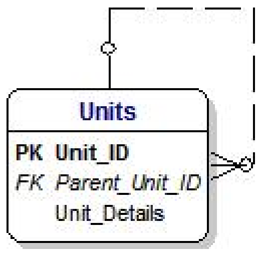
Then we think about the fact that every unit is part of a larger organization.

In other words, every unit reports to a higher level within the overall organization.

Fortunately, we can show this in a very simple and economical fashion by creating a relationship that adds a parent ID to every unit.

This is accomplished by adding a relationship that joins the table to itself.

This is formally called a *reflexive* or *recursive* relationship, and informally called *rabbit ears*, and looks like this:

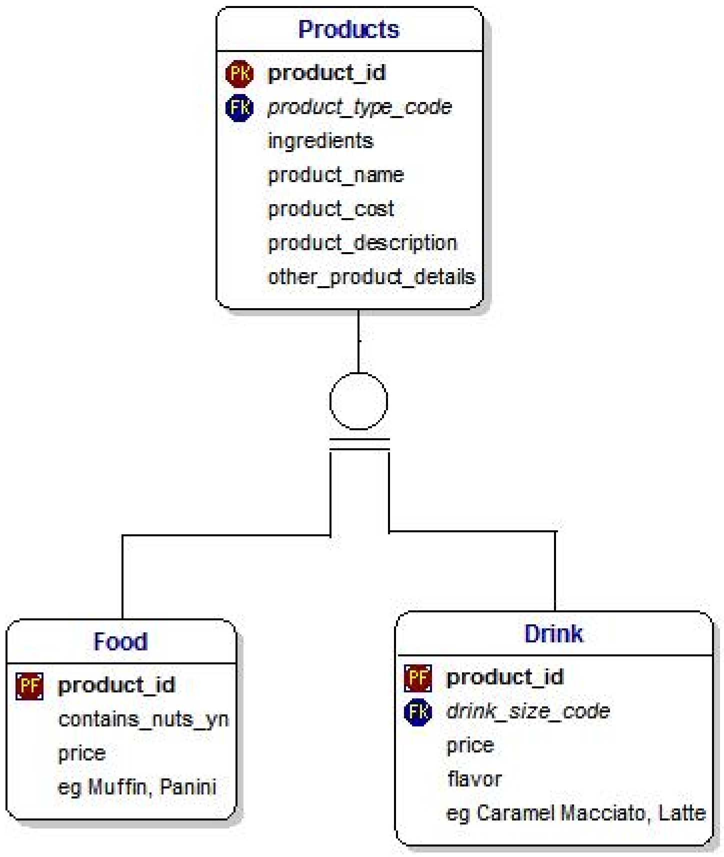


The unit at the very top of organization has no one to report to, and a unit at the lowest level does not have any other unit reporting to it.

In other words, this relationship is **optional** at the top and bottom levels.
We show this by the small letter *O* at each end of the line that marks the relationship.

4.10 Inheritance

Inheritance is a very powerful technique. It allows us to model complex situations in a manner and style that is very simple.



In this situation, we are thinking about 'Food and Drink'.

'Food and Drink' are specific examples of the more general thing called a *product*.
They inherit common attributes from the product, and also have some of their own.
For example, 'Food' can contain 'Nuts' but 'Drink' may not contain 'Nuts,' but both have a product name.

The unusual symbol in the middle of the diagram, composed of a circle with two small lines underneath it is how **inheritance** is shown using the Dezign data modeling tool.

Inheritance is a very important topic when you are creating a data model. In plain English, we would say that inheritance occurs where a Parent-Child relationship exists between things of interest (or entities).

You can ask the simple **'Is-a'** question - in this case, if we ask 'Is a muffin a product' then clearly the answer is 'yes' so we have established that there is an inheritance relationship between them.

In the example of inheritance shown in this diagram, we can see that all products have names and descriptions.

Therefore, 'Food and Drink' will inherit these characteristics from the parent product.

We call the product the **Super-Type** and 'Food and Drink' are **Sub-Types**.

However, each sub-type of product will have specific characteristics that it does not share with other sub-types. For example, a 'Drink' has a flavor but 'Food' does not.

One of the important things in your data model is to be sure you have identified all the inheritance relationships. However, an inheritance relationship is often blurred in a real physical database because it can be clumsy to implement and has to be resolved with the addition of a table that is often called an associative table. This associative table has a one-to-many relationship with each of the original tables that were in the many-to-many.

There are broadly two types of data model:

- Conceptual or Logical
    - This focuses on a business-oriented specific of a situation that identifies the 'things of interest' and how they are related.
- Physical
    - This introduces aspects that relate to implementation in a specific database

Inheritance can appear in a logical data model but it disappears in the physical database, which is what ultimately becomes the database.

Relational databases do not support inheritance. Therefore our thinking must include the question of when we stop showing the inheritance relationship and replace it with two one-to-many relationships. Business users tend to be comfortable with many-to-many but for data modelers, DBAs and developers it is usually better to replace them.

Inheritance is a very simple and very powerful concept. We can see examples of inheritance in practice when we look around us every day. For example, when we think about 'Houses,' we implicitly include bungalows and ski lodges, and maybe even apartments, beach huts and house boats.

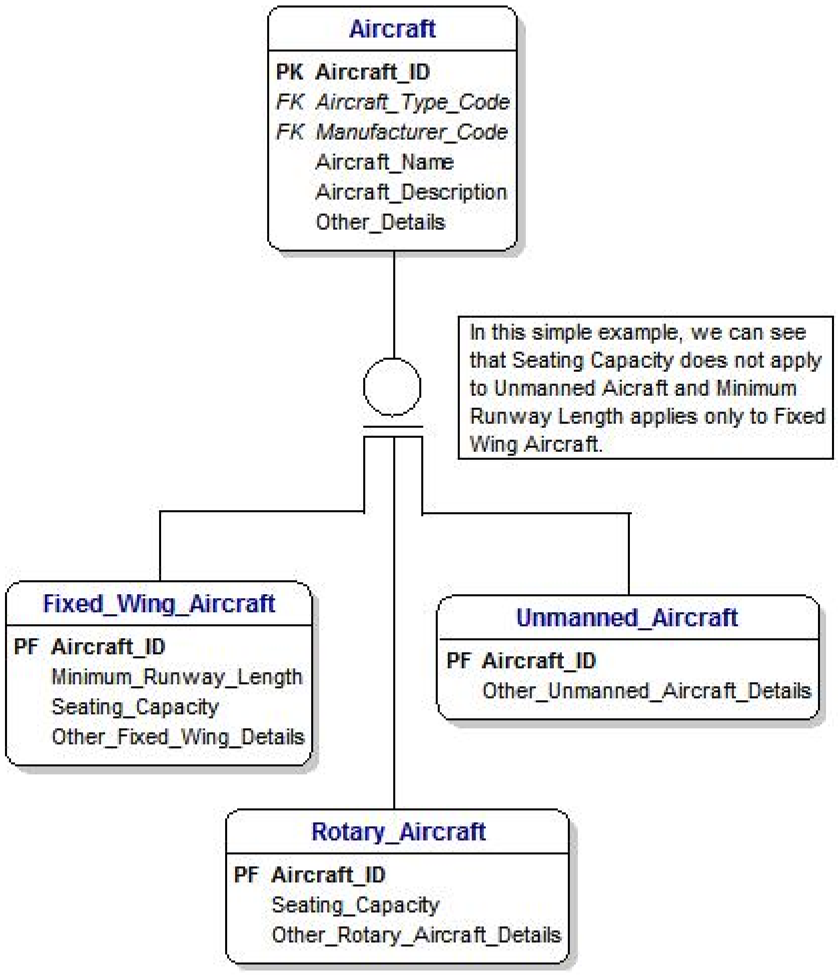
In a similar way, when we discuss aircraft we might be talking about rotary aircraft, fixed wing aircraft and unmanned aircraft.

However, when we want to design or review a data model that includes aircraft, then we need to analyze how different kinds of aircraft are shown in the design of the data model.

We use the concept of 'Inheritance' to achieve this. Inheritance in data modeling is just the same as the general meaning of the word. It means that at a high level, we identify the general name of the 'Thing of Interest' and the characteristics that all of these things share. For example, an aircraft will have a name for the type of aircraft, such as *Tornado* and it will be of a certain type, such as fixed-wing or rotary.

At the lower level of fixed-wing aircraft, an aircraft will have a minimum length for the runway that the aircraft needs in order to take off.

This situation is shown in the following diagram:



4.11 Reference Data

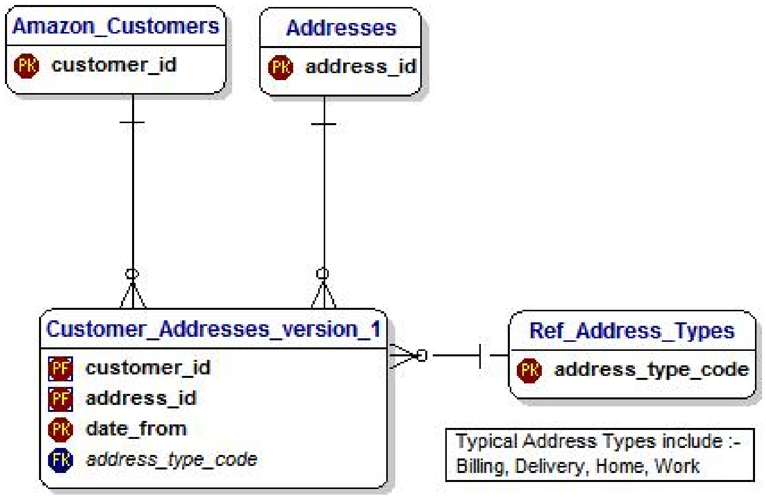
Reference data is very important. Wherever possible, it should conform to appropriate external standards, particularly national or international standards. For example, the International Standards Organization (ISO) publishes standards for country code, currency codes, languages codes and so on.

For addresses, the UK Post Office Address File (PAF file), is the standard used to validate addresses within the UK.

4.11.1 Address Types example

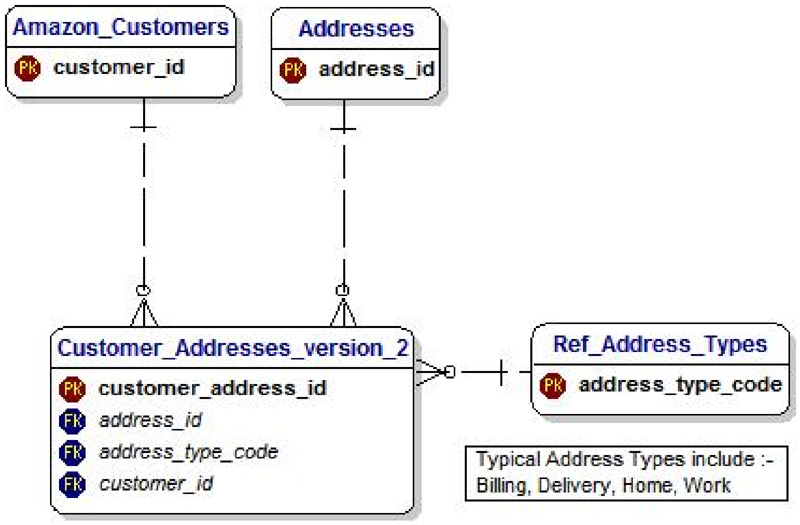
Address types are another example of reference data.

There are two design possibilities.
The first is good because it shows clearly the logical relationship where a customer address can be identified uniquely by a combination of the customer ID, the address ID and the date from when the address was valid for the customer.



Of course, it is not always possible to determine the 'Date From' value, and it is not always something that it is appropriate to ask every customer.

Therefore, a better and more general approach is to use a key (that we discussed in Section 3.3) for a record and leave the 'Date From' field optional.



4.11.2 Customer Addresses

This is a general and flexible approach to handling addresses in our data model.

We have a separate Address Table, which allows us to have more than one address for any customer very easily.

This design also has other benefits:

- We can accommodate more than one person at the same address. We need to do this because different members of a family may sign up separately with Amazon.
- With a separate table of addresses, we can easily use commercial software to validate our addresses.
- To find this kind of software, simply Google 'Address Validation Software'.
- We have used QAS with great success in the past.
- With this approach, we can always be sure that we have 100% good address data in our database.

4.11.3 Reference Data

Reference data has the following characteristics:

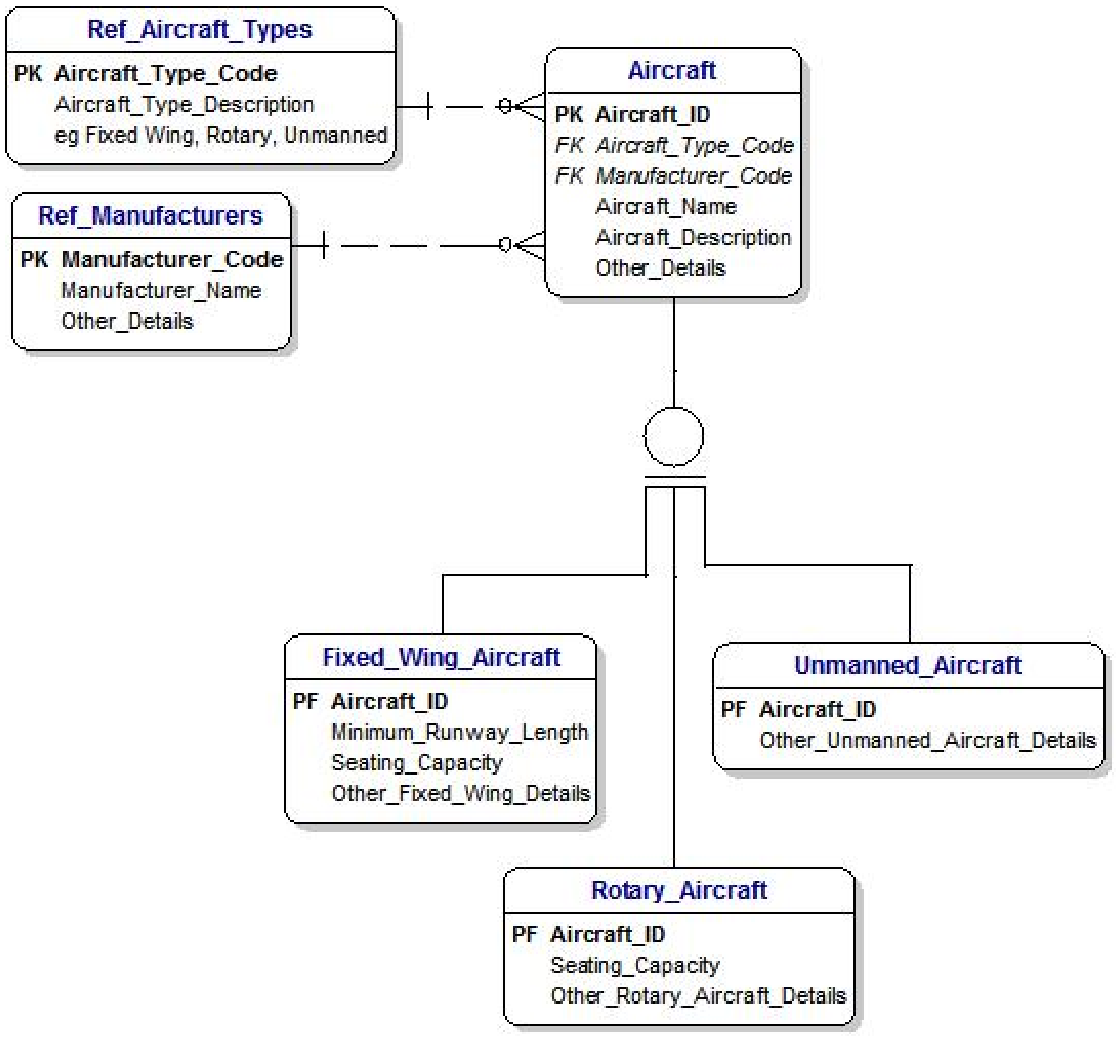
- It does not change very much.
- It has a relatively small number of values, usually less than a few dozen and never more than a few hundred.
- Therefore we can show it with a code as a primary key.
- Data in Reference Data Tables can be used to populate drop-down lists for users.
- In this way, it is used to ensure that all new data is valid.

4.11.4 Standards

- In the Address Table, you will see a field called 'ISO_Country_Codes'.
- ISO stands for the 'International Standards Organization'.
- It is always good to use national or international standards.

4.11.5 Aircraft example

This diagram shows two basic examples of Reference data that might apply to our simple aircraft data model.



4.12 What have we learned?

In this chapter, we have covered the basic concepts in data modeling, including:

- Primary and Foreign Keys
- One-to-Many and Many-to-Many Relationships
- Rabbit Ears or Reflexive Relationships
- Inheritance
- Reference Data

That will give us the basics of the language in which we can talk about and describe data models.

5. A Database for a Video Game

5.1 Approach

The first step is to decide on the theme of the Game.

For this Tutorial, we have chosen a 'Shoot 'em Up' which is based on a very popular Game for the Microsoft Xbox called Gears of War.

Here is the page on the Database Answers Web Site that shows the Data Model :-

- http://www.databaseanswers.org/data_models/gaming_gears_of_war/index.htm

This Kick-Start Data Model features :-

- the Good Guys, who are Soldiers
- the Weapons
- the Bad Guys, who are Locusts
- the Rules of Engagement between the Good Guys and the Bad Guys

5.2 The Good Guys

There are seven Soldiers in the Game, with different Ranks and different backgrounds

5.2.1 Colonel Victor Hoffman

His Profile reads :-
A military man to the core, Colonel Victor Hoffman demands discipline and sacrifice from those under his command.

5.2.2 Sergeant Marcus Fenix

His Profile reads :-
Few have given more and lost as much as Marcus Fenix.

A promising soldier during the Pendulum Wars, Marcus saw everything change on Emergence Day.

Marcus bravely fought the Locust for ten years, then, during an intense battle, he abandoned his post to rescue his father, Professor Adam Fenix.

But he arrived too late.

Marcus was tried for dereliction of duty and sentenced to 40 years in
Jacinto Maximum Security Prison.
Incarcerated for four years before being released to fight Locust again, Marcus was later promoted to sergeant.

5.2.3 Private Damian Baird

His Profile reads :-
Private Damon Baird is a dedicated tech-head and professional skeptic.
In Baird's world, if something can go wrong, it probably already has.
His sarcasm can keep people at a distance, which is why Baird prefers the company of machines.
He believes in the Coalition's cause, but he's often frustrated with command decisions, and took offense when Hoffman promoted Marcus Fenix to lead Delta Squad instead of him.

5.2.4 Private Anthony Carmine

His Profile reads :-

As the youngest member of Delta Squad during the Lightmass Offensive, what Private Anthony Carmine lacked in combat experience, he made up for in unbridled enthusiasm.

5.2.5 Private Dominic Santiago

His Profile reads :-
A seasoned fighter who's positive even in the darkest of hours, Dominic Santiago freed his best friend Marcus Fenix from Jacinto Maximum Security Prison and recruited him into Delta Squad.

His battlefield intensity is rivalled only by his loyalty to Marcus--and his wife, Maria.

Dominic's relentless search for his wife finally ended during Operation: Hollow Storm, when he and Marcus found her in a Locust processing facility, barely alive and irrevocably twisted.

Marcus left his side to allow Dom a final moment with his beloved Maria before ending her suffering.

5.2.6 Lieutenant Anya Stroud

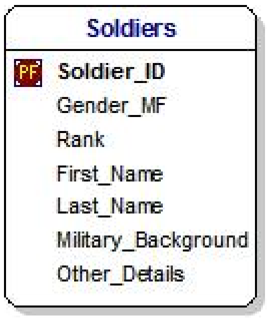
Her Profile reads :-
As Delta's Control contact, Anya Stroud guided Delta Squad on their mission to destroy the Locust, providing vital intel and strategic advice to the squad in the field.

5.2.7 Samantha 'Sam' Byrne

Her Profile reads :-
Samantha "Sam" Byrne's father, Sgt. Samuel Byrne, fell in battle at the siege of Anvil Gate in Anvegad, Kashkur before the birth of his daughter.

5.2.8 Summary

At this point, we can see that all Soldiers have Gender, Names, Ranks and a military background.
Therefore, our Soldiers Table looks like this :-



5.3 Choosing the Weapons

Now we can choose the Weapons to match the Soldier's unique qualities.
These are our options that are described here.

5.3.1 Boomshot Grenade Locust

Description of the Boomshot Grenade Locust is :-
A Boomshot Grenade Locust is a short- to mid-range grenade launcher that can easily take down a target in a single shot

5.3.2 Hammer of Dawn

Description of the Hammer of Dawn is :-
The Hammer of Dawn is An Imulsion-powered satellite that rains down a devastating particle energy stream.
It can wipe out anything from small Locust squads to entire city blocks.

5.3.3 Long Shot Sniper Rifle

Description of the Longshot is :-
The Longshot Sniper Rifle is a high-powered, bolt-action sniper rifle with a powerful zoom sight.

5.3.4 One Shot

Description of the OneShot is :-
The OneShot is an intimidating and obscenely powerful long-range sniper rifle capable of destroying most foes in a single shot

5.3.5 Scorcher Flamethrower

Description of the Flamethrower is :-
The Scorcher Flamethrower is a short- to mid-range weapon that emits a concentrated stream of fire that chars your enemies.

5.3.6 Troika

Description of the Turret is :-
The Troika Turret is a high-powered, turret-mounted Locust machine gun that fires continuous rounds across the battlefield.

5.3.7 Summary

The descriptions of all the Weapons looks like this :-

1) A Boomshot Grenade Locust is a short- to mid-range **grenade launcher** that can easily take down a target in a single shot.

2) The Hammer of Dawn is An Imulsion-powered **satellite** that rains down a devastating particle energy stream. It can wipe out anything from small Locust squads to entire city blocks.

3) The Longshot Sniper Rifle is a high-powered, bolt-action sniper **rifle** with a powerful zoom sight.

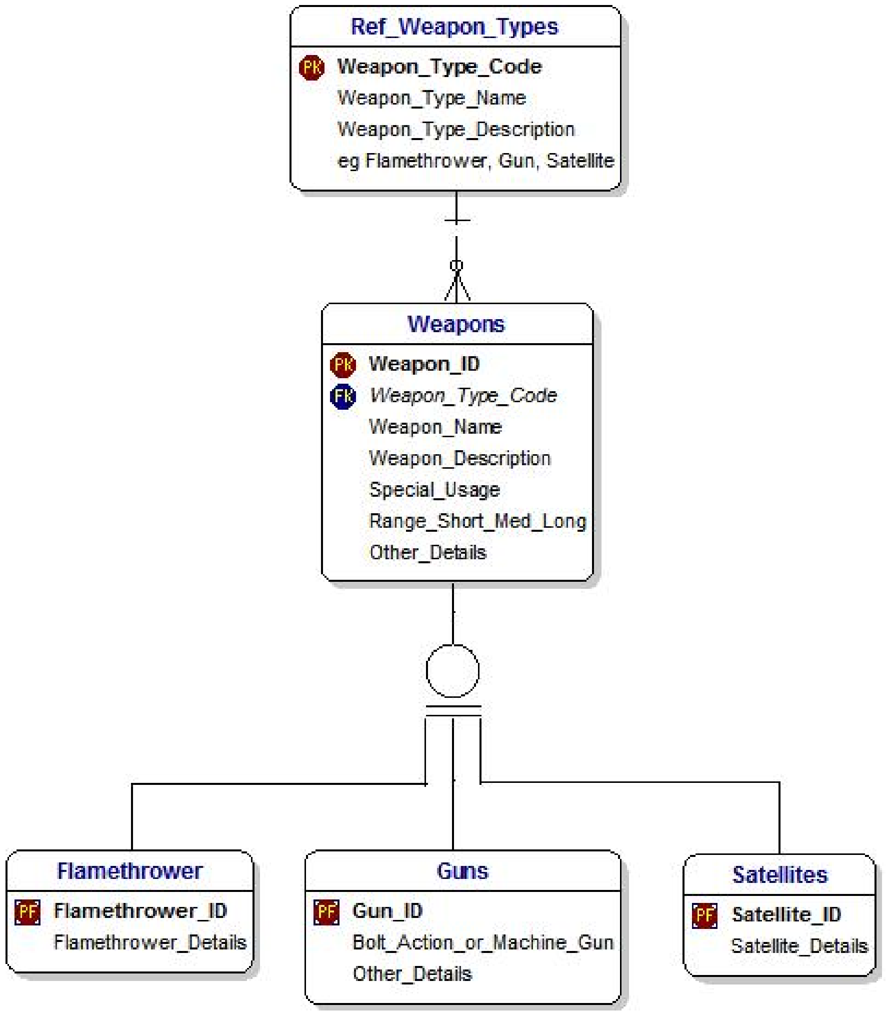
4) The OneShot is an intimidating and obscenely powerful long-range sniper **rifle** capable of destroying most foes in a single shot

5) The Scorcher **Flamethrower** is a short- to mid-range weapon that emits a concentrated stream of fire that chars your enemies.

6) The Troika Turret is a high-powered, turret-mounted Locust **machine gun** that fires continuous rounds across the battlefield.

An analysis of these weapon identifies that we have one Flamethrower, three Guns and a Satellite.

Therefore, the fragment of our Data Model for Weapons looks like this :-



5.4 The Bad Guys

In this game, the Bad Guys are all Locusts. However, they come in different shapes and sizes, and offer different threats.

5.5.1 Berserkers

The Berserkers Profile reads :-

Berserkers are female Locusts.
They use their keen hearing and sense of smell to seek out their prey and bludgeon it to death with their hammer-like fists.

5.5.2 Brumaks

The Brumaks Profile reads :-

To stand in the Brumak's shadow is to stare death in the face.
These hulking war machines possess a deadly assortment of weapons, from wrist-mounted machine guns to over-the-shoulder rocket launchers.
For any chance of survival against a Brumak, blast away bits of its armor to reveal the soft, weak spots underneath.

5.5.3 Grenadier

The Grenadiers Profile reads :-

Locust Grenadiers are never afraid to get up close and personal.
They have a hard-charging kamikaze attack and rush their enemy with little concern for their own welfare.

They specialize in both grenades and the Gnasher Shotgun, drawing their targets out of cover with one before blasting them to pieces with the other.

5.5.4 RAAM

The RAAM Profile reads :-

An imposing figure, RAAM towers over all humans, his silent demeanor concealing a violent and merciless nature.
In battle, RAAM is a formidable opponent who wields a Troika Machine Gun while controlling the Kryll that he sometimes employs as a shield. RAAM met his demise at the hands of Marcus Fenix aboard the Tyro Pillar, where his reign of terror came to an abrupt and welcome end.

5.5.5 Summary

One of the Locusts is identified as being female. Therefore, we have to assume that all Locusts have a gender, which will be Male, Female or Unknown.

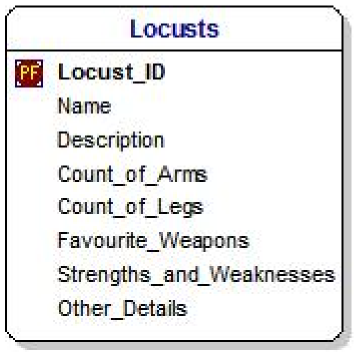
Locusts have a description of their strengths and weaknesses.

The photos of the Locusts show them having arms and legs. Therefore we include an Arm Count and Leg Count fields, which we default to two of each.

Gender is a code that has only three values – Male, Female and U for Unknown.

So we can include them in a field called 'Gender_MFU'.

Therefore, our Locusts Table looks like this :-



5.5 Thinking in General Terms

5.5.1 Soldiers, Locusts and Inheritance

In understanding the Game, we need to consider how to simplify the way we define Soldiers and Locusts.

This will help us to define the Game at a higher level and think about it in a more general way.

Our first step is to consider Soldiers and Locusts as Participants in the Game.

We could call them Beings or Actors or Roles but for simplicity at this basic level we call them simply 'Participants'.

They both have Names, and a Gender. Therefore, we move the Gender_MFU to the Participants entity.

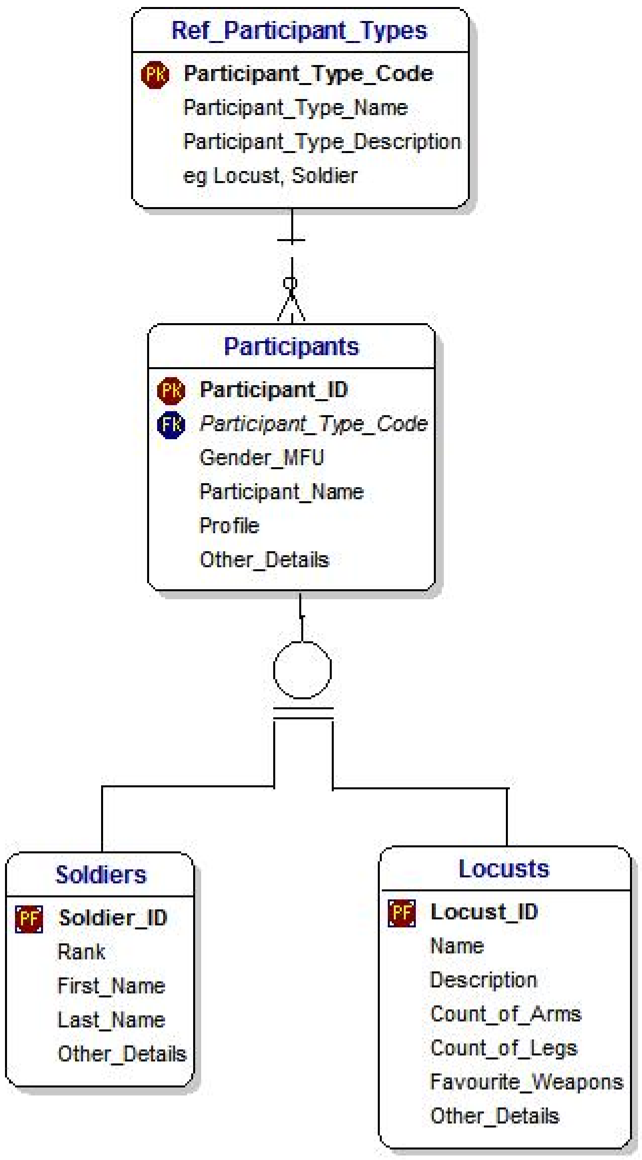
Soldiers have a Military Background and Locusts have a Strenths_and_Weaknesses field.

We can replace these two by one field in the higher 'Participants' Table.

We will call this field 'Profile'.

Therefore, we show our new 'Participants' table as the Parent or 'Super-Type. With Soldiers and Locusts as Children or Sub-Types.

The Data Model fragment will look like this :-



5.5.2 Favourite Weapons and Many-to-Many Relationships

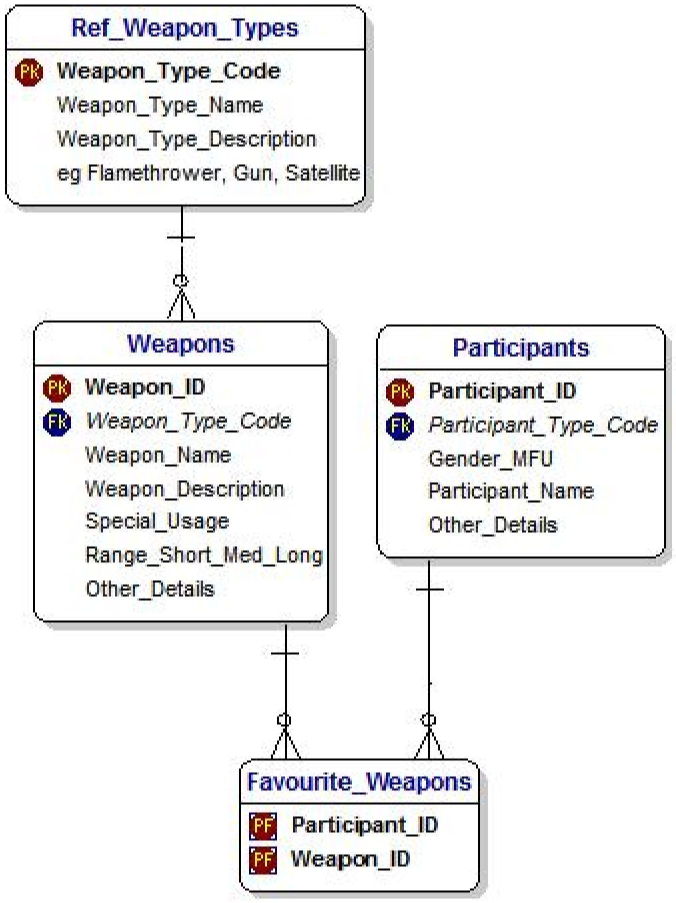
It turns out that both Soldiers and Locusts have favourite Weapons and now that we have established a Participants entity, we can favourite Weapons as an attribute of the Participants entity.

Each Participant can have many favourite weapons, and each particular type of Weapon can be the favourite of many Participant.

In Data Modelling terms, we call this as a 'Many-to-Many Relationship' between Participants and Weapons.

We have moved the Favourite_Weapons attribute from the Locust entity to the new Favourite_Weapons entity.

Therefore, the Data Model fragment Soldiers and Locusts look like this, where the 'Favourite_Weapons' entity shows that each Participant can have many favourite Weapons and vice versa. :-

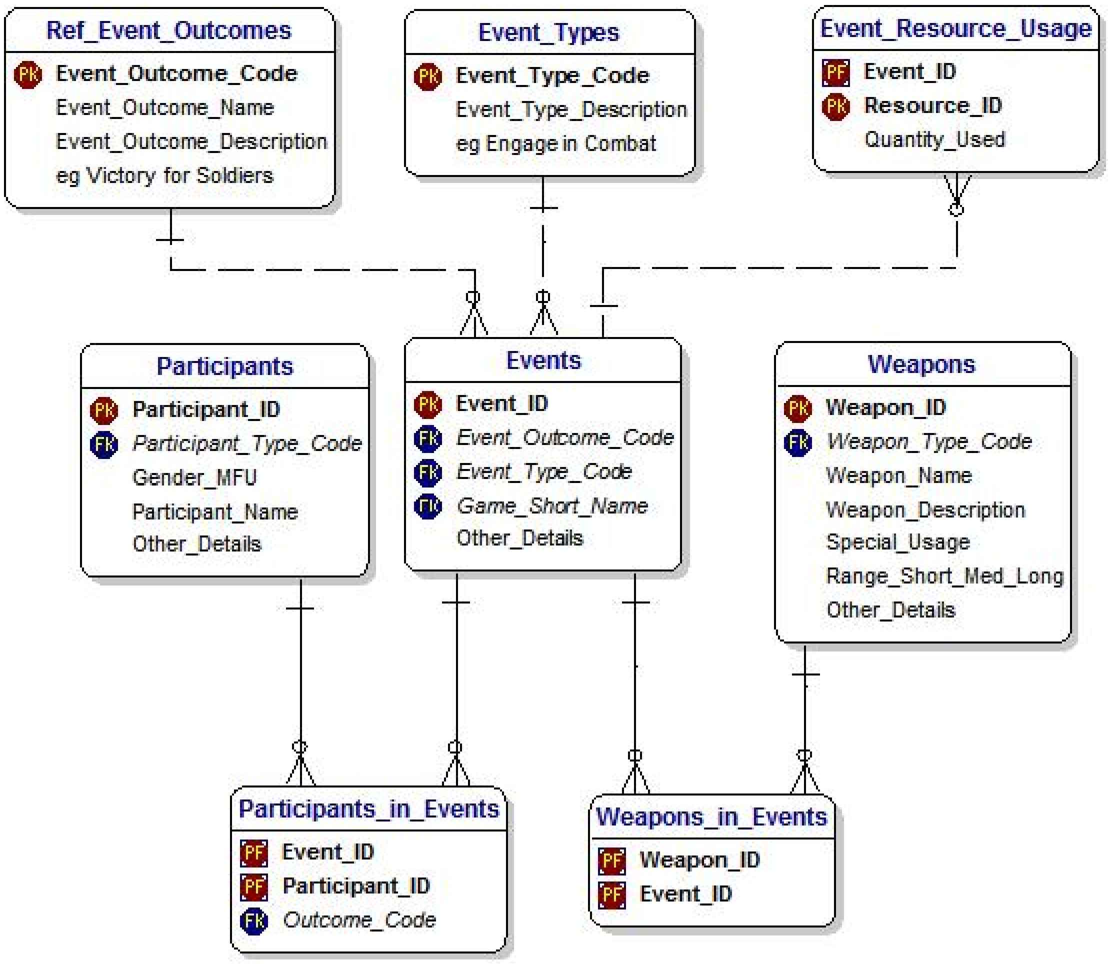


5.6 Rules of Engagement

In general terms , the Soldiers will have a number of Objectives and will have to engage with the Locusts to achieve these Objectives.

We will consider these engagement as a series of Events involving all Participants (bit not every time).

Each Event will have an outcome (like Victory or Defeat) and use Resources, such as Bullets, that might be in limited supply.



5.7 Design Patterns

5.8.1 Introduction

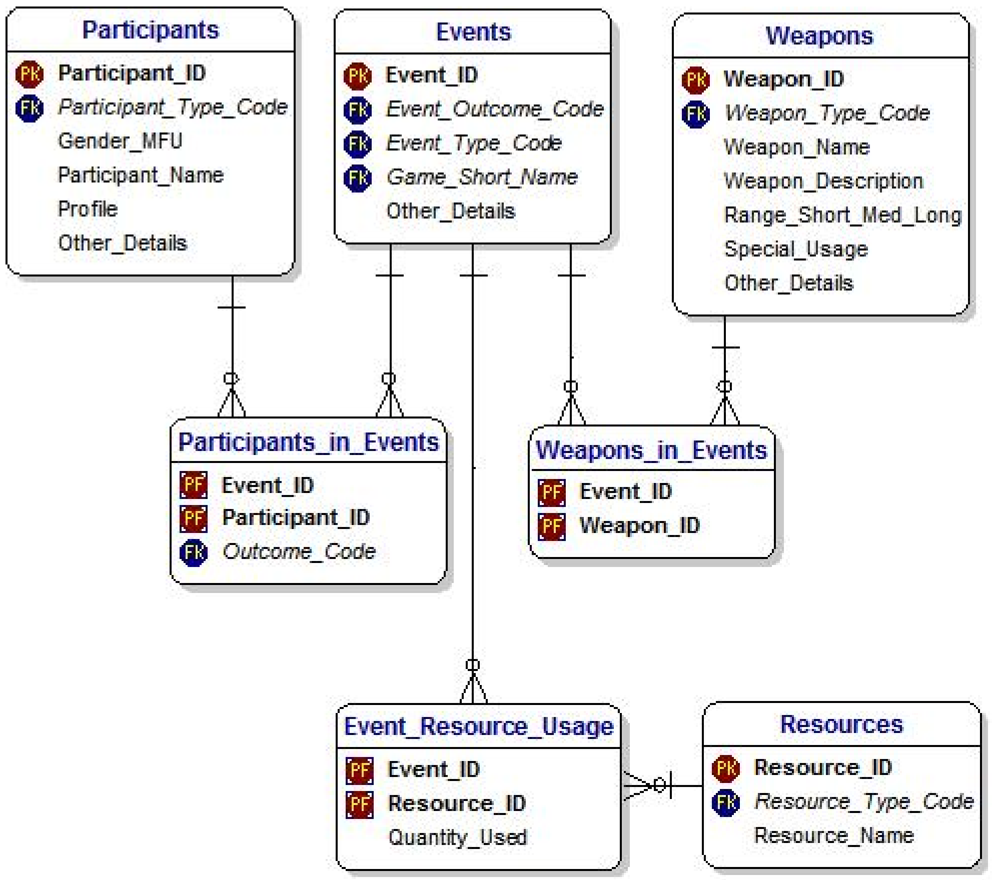
Design Patterns are very powerful because they help us to recognise similar situations that occur very frequently in real life.

Then we start thinking like a Database Designer or Data Modeller and suddenly, it becomes a lot easier.

The obvious candidate for a Design Patter in our simplified Video Game is Events.

5.8.2 Complete Design Pattern

This Model shows all the components in the Complete Design Pattern :-



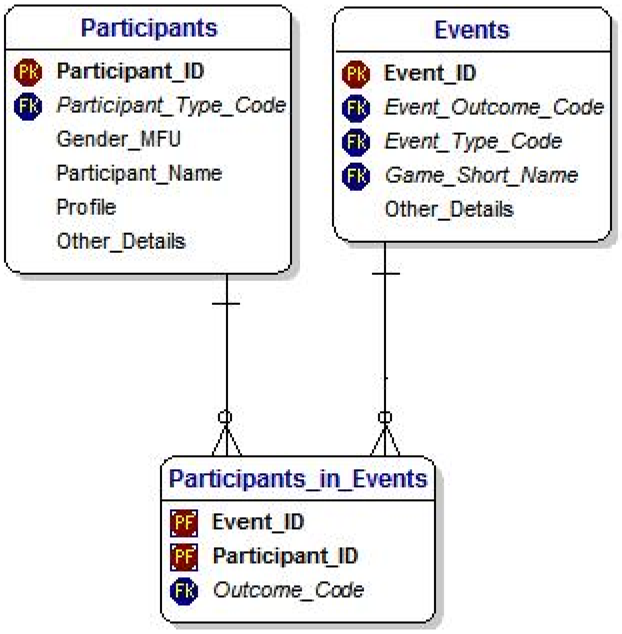
The logic behind different variations says that :-

- There will always be an Event
- There will always be at least one Participant
- Weapons will be involved for an Event that is a fight between Soldiers and Locusts.

5.8.3 Participants and Events

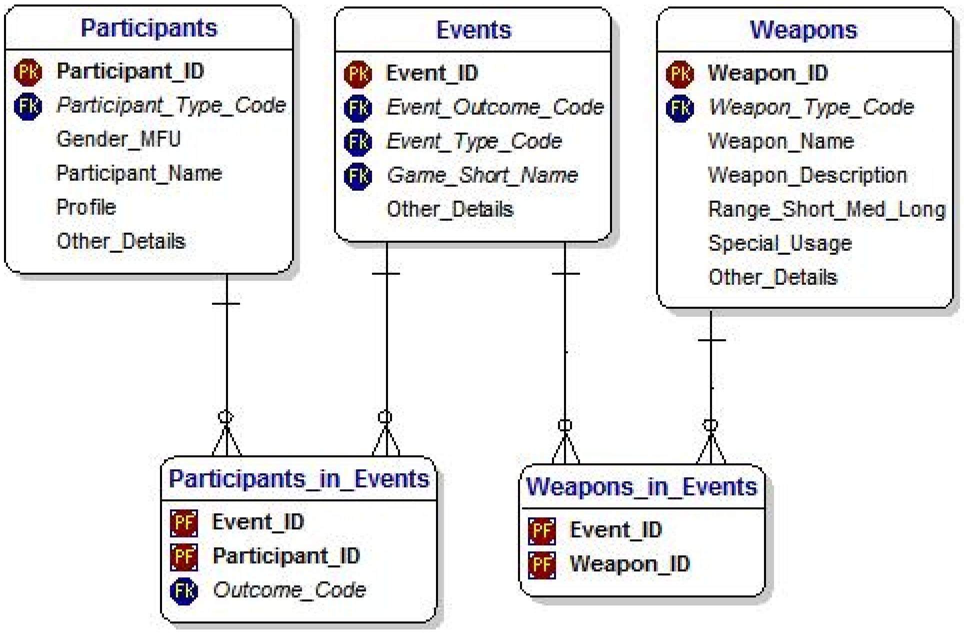
This is the smallest version of the Pattern, because an Event will always be involved and Participants will always be involved.

If we think about an Event that does not use Weapons, for example, retreating or advancing without engaging with the Locusts, then we have a simpler Data Model that does not involve Weapons and looks like this :-



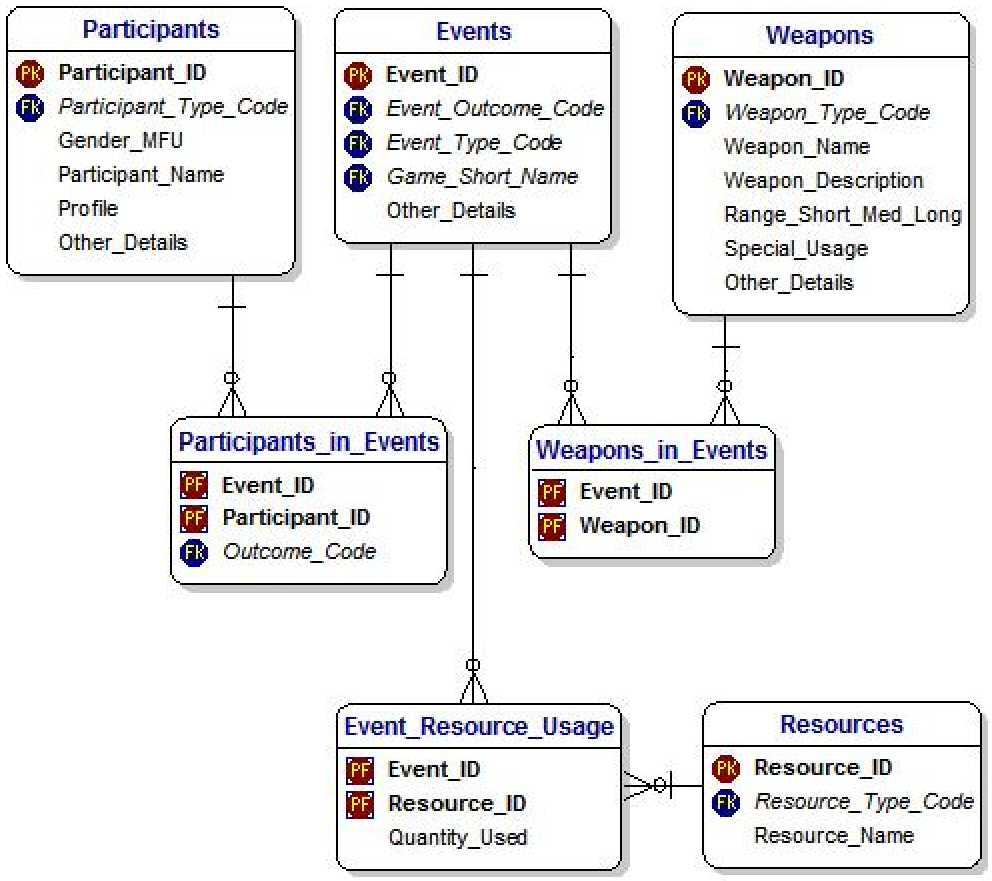
5.8.4 With Weapons

If the Soldiers meet Locusts while they are on the move, then we add Weapons and the Data Model that looks like this :-



5.8.5 With Weapons and Resources

If the Soldiers meet Locusts while they are on the move, then we add Weapons and the Data Model that looks like this, which is the full Design Pattern that we started with :-

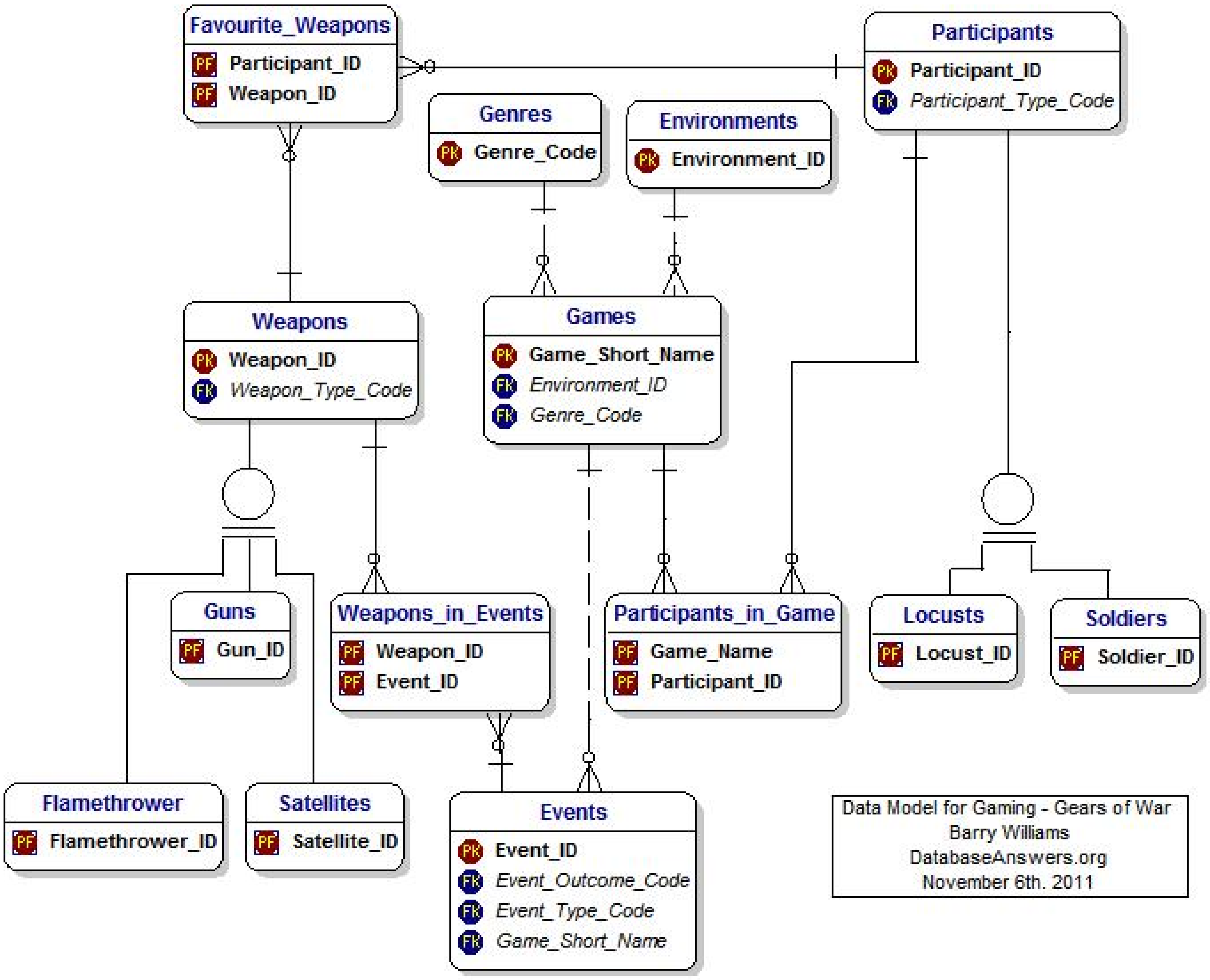


5.8 The Complete Data Model

When we combine the Soldiers, Locusts and Weapons, this is what our Complete Data Model looks like.

We have left out the Reference Data tables to keep the Model simple and easier to read.

For the same reason, we have included only the Primary and Foreign Keys and also omitted the Attributes.


Favourite_Weapons
PF Participant_ID
PF Weapon_ID
Participants
PK Participant_ID
FK Participant_Type_Code
Genres
PK Genre_Code
Environments
PK Environment_ID
Weapons
PK Weapon_ID
FK Weapon_Type_Code
Games
PK Game_Short_Name
FK Environment_ID
FK Genre_Code
Guns
PF Gun_ID
Weapons_in_Events
PF Weapon_ID
PF Event_ID
Participants_in_Game
PF Game_Name
PF Participant_ID
Locusts
PF Locust_ID
Soldiers
PF Soldier_ID
Flamethrower
PF Flamethrower_ID
Satellites
PF Satellite_ID
Events
PK Event_ID
FK Event_Outcome_Code
FK Event_Type_Code
FK Game_Short_Name
Data Model for Gaming - Gears of War
Barry Williams
DatabaseAnswers.org
November 6th. 2011