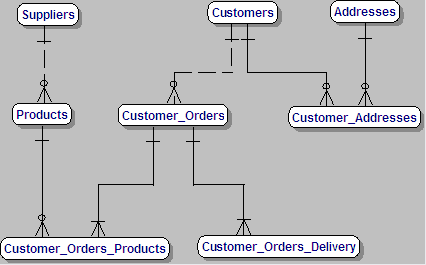
SAMPLE BASIC MODEL

* Customers and Orders at Starbucks and at Amazon.com.

**Overview of Buying from Amazon.com and Starbucks**

* Order
* Pay
* Take Delivery



**Introduction**

 This diagram is an overview of our Database Schema which we will be going through in detail before we review it at the end of this Tutorial.

 We will be talking about a Database Schema covering Customers, Deliveries, Orders, Products and Suppliers.

 We will start by looking at a Schema for Starbucks and then moving on to Amazon.

 Bear in mind that the more complex solution includes the simpler one.

 Before we begin, let me say that I have adopted the plural for Tables (such as 'Customers') because it is more natural than the singular form.

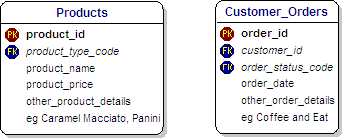
 Strictly speaking, Data Modellers would be using the singular for a Data Model of this type, but I prefer a more User-Friendly approach.

 Data Modellers would also refer to Entities, rather than Tables, for the 'Things of Interest'.

 You will notice that I also refer to Tables for the same reason of being User-Friendly.

**Enter Starbucks**

* Starbucks offers Products
* Customers can Order, for example, Coffee and Something to Eat



**Our motto - Keep It Simple and Straightforward (KISS)**   
We are starting in a simple way with this diagram which shows two Entities without a Relationship.   
The Relationship between them is established in the next diagram, when the meaning of the 'PK' and 'FK' will be explained.   
  
**Products**   
I want to convey to you the value of seeing the real world as a Database Schema.   
It's always very interesting and adds a challenge as you walk or drive around and   
think about how you would create a Database Schema for the real world that you can see around you.

 Let's say we go into Starbucks to get a coffee and something to eat.

 At this point, Starbucks doesn't know us as a Customer.

 Also we might be doing the office coffee run, amd maybe we have with us our Order on a piece of paper.

 These two 'Things of Interest', Products and Customer\_Orders, are not related until we give our Order.

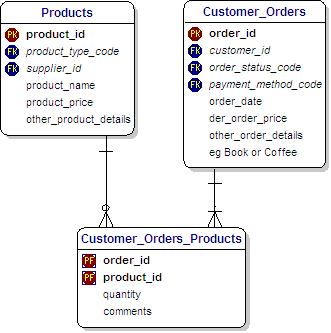
 Data Modellers commonly describe a Database Schema as an Entity-Relationship-Diagram, or ERD.

 The word 'Relationship' is therefore very important.

 However, at this point, we haven't established a Relationship between the Products and the Customer\_Orders.

 We will do that in the next slide.

1. **Keys**
   * Primary and Foreign Keys
2. **Relationships**
   * One-to-Many and Many-to-Many Relationships



**Primary Keys.**

 These are indicated by **'PK'** alongside the field in a Table.

 They show the field that identifies the record uniquely.

 For example, the 'product\_id' for a Product and an 'order\_id' for an Order.

 The values are usually generated by an 'Auto-Increment' field.

 They don't have any intrinsic meaning and are simply integers generated automatically one after another.   
  
**Primary/Foreign Keys.**

 These are indicated by **'PF'** alongside the field in a Table.

 In this case, the field also exists as a Primary Key in another Table.

 In our example, the Primary Key in the 'Customer\_Orders\_Products' Table is a Composite Key, made up of two Fields.

 The 'product\_id' is the Primary Key in the 'Products' Table, and the 'order\_id' is the Primary Key in the 'Customer\_Orders' Table.   
  
**Foreign Keys.**

 These are indicated by 'FK' alongside the field in a Table.

 They show a reference to the Primary Key for a record in another Table.

 For example, the 'customer\_id' in the Customer\_Orders Table is linked to a 'customer\_id' in the Customer Table (not shown).   
  
**One-to-Many Relationships.**

 One-to-Many Relationships are very common in the real world, and can be read that 'One' Parent can have 'Many' children.

 In this diagram, One Product can be associated with Many Customer\_Orders\_Products.

 The line between Products and Customer\_Orders\_Products is a continuous line (not a dotted line).

 This means that the 'product\_id' field is a Primary Key in the Customer\_Orders\_Products Table.

 A dotted line would indicate the existence of a Foreign Key (which is defined above).   
**'Crows-Feet'**

 At the Customer\_Orders\_Products end, there is what's called a 'Crows-Feet' symbol.

 This indicates the 'Many' aspect of the relationship.

 The little 'o' shows that this is an Optional relationship.

 In other words, not every Product has an associated record in the Customer\_Orders\_Products Table.

 This applies, of course, to Products that nobody orders.

 At the other end of the line there is a small horizontal line. This means that this is essential.

 In other words, every product\_id in the Customer\_Orders\_Products Table must match a product\_id in the products Table.

 This means that a Customer cannot order a Product that does not exist.

 In passing, let me say that this is different from the Product not being in stock.

 This aspect is called Inventory Control and is covered by a different Database Schema.

 The Relationship between Customer\_Orders and Customer\_Orders\_Products shows a short horizontal line at each end.

 This indicates that every Order must contain at least one Product.

 It also indicates that Product in every Customer\_Orders\_Products must be associated with an Order in Customer\_Orders.   
  
**Many-to-Many Relationships.**

 Many-to-Many Relationships are very important in the real world, and we see them all around us every day.

 In our example, many Customers can order the same Product.

 Many Products can be ordered by the same Customer.

 This is called a Many-to-Many Relationship. It cannot be implemented directly in a Table in a Relational Database.

 The solution is to create an intermediate table that records all combinations that exist of Customer and Product.

 For example, if a Customer ordered three Products, then there would be three records for that Customer in this intermediate table.

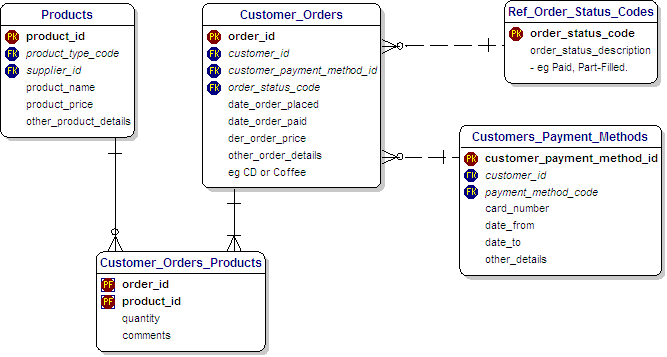
 The combination of the Primary keys in the parent Tables become the Primary key in this intermediate table.

 We also need to allow for Comments like 'Make it extra strong, please'.

 If we get the structure to the Database right, then it's straightforward to extend it.

**Pay the Bill**

* Enter Starbucks
* Order Coffee and Something to Eat
* Finally Pay for it.



**Payments**

 Of course, we have to pay for what we have ordered before we get to consume it.

 Fortunately, paying in Starbucks is quite straightforward.

 In passing, let's note that the total price we have to pay is derived from the total cost of the individual items.   
  
**Customer Payment Methods**

 These will usually be Cash or Credit Card, and are shown in more detail in the next diagram.

 Payments are simpler at Starbucks than Amazon.com, but our design is general and covers both.   
  
**Derived Data**

 The total price we have to pay is derived from the total cost of the individual items.

 In a 'Normalised' Database Schema, we do not normally show derived fields, such as total figures.

 This allows us to drill-down to the basic fundamental data.

 We want to get to the point where we can't remove anything without losing meaning or value.

 For this reason, I have shown the total price as a 'Derived' field, and called it 'der\_order\_price'.

 One of the strengths of the Relational Database is that it strips away everything which is not fundamental.   
  
**Reference Data**

 Reference data is important, because it occurs in virtually every Database Schema in the real world.   
  
**Status**

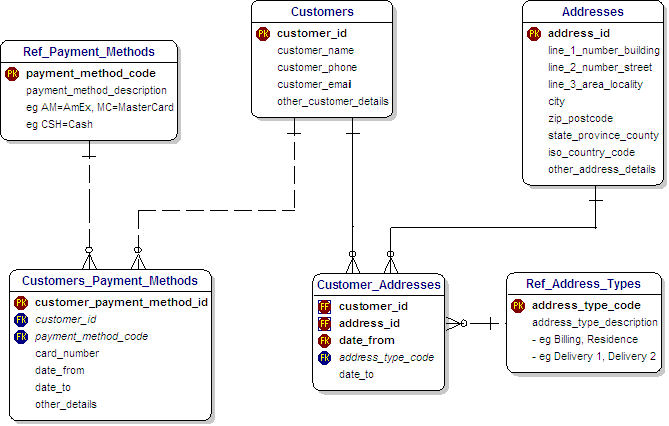
 The status of an order in Starbucks usually changes so quickly that we don't need to keep track of it.

 It is shown here to anticipate its use for Amazon.com, which is more complex, as we will see later.

 The Status can be part of the Customer\_Orders table, and take values from the 'Ref\_Order\_Status\_Codes' table.

**Customers at Amazon.com**

* Customers
* Addresses
* Payment Methods



**Addresses, Payments and Many-To-Many Relationships**

 **Customers and Addresses**

 At Starbucks, we give very little information about ourselves as Customers.

 If we pay by cash, we give no details at all.

 Amazon.com, on the other hand, needs to know a good deal more about us.

 For example, they need to store details about our addresses, and we can have many of these.

 In our Database Schema,we could have a Billing Address, a Residence Address and more than one Delivery Address.

 In passing, let me say that the existence of a separate Addresses Table is often a distinguishing characteristic between a Database Schema with   
limited scope,and one intended for use in a mission-critical capacity.

 Therefore we store Addresses in a separate Table, instead of repeating Address details many times in Customer Tables.

 This has many benefits. It allows us to validate the Address against files of recognised Addresses.

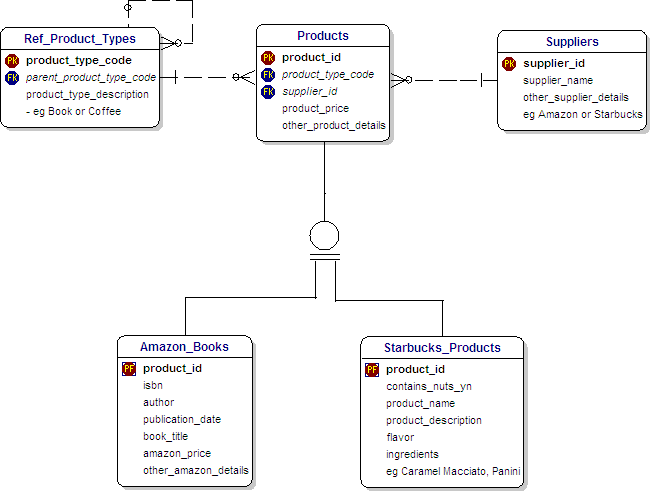
 This has become a standard approach so that many commercial products are available to help in this validation.

 In addition, maintaining a separate Address Table helps in tracking changes of Customer Addresses.

 The Primary Key in the Customer\_Addresses table is the combination of the Customer ID, the Address ID,   
and the Date when the Customer moved in.   
This works because the Customer cannot move in to the same Address more than once on the same Date.   
Of course, they could move out and back in on another date, and our design covers that situation.

**Products at Amazon.com or Starbucks**

* Coffee
* Books
* Product Types (Inheritance and Recursion).



**Products**   
Let's move on to the topic of Products. As you might imagine, this can be a complex area.   
Two important aspects are Inheritance and Product Hierarchies.   
  
**Inheritance**

 This diagram shows a 'Products' table that holds details that are common to all Products.

 It shows an 'Amazon\_Books' table that holds details that are specific to Books from Amazon.

 It also shows a 'Starbucks\_Products' table that holds details that are specific to Products from Starbucks.

 These two are Children of the Parent 'Products' Table.

 Therefore, this concept is referred to as Inheritance and is very powerful in the real world.

 On the diagram, it is shown by the circle with two short horizontal lines underneath it.

 Data Modellers define these as 'Sub-Types' of a 'Super-Type'.

 This diagram shows the Logical level of the Schema.

 At the physical level, this can be implemented in one of three ways :-

1. Three tables - exactly like the Logical level.
2. Two tables - one for Amazon and one for Starbucks, with the common data items copied down from the Parent Table.
3. One table - at the higher level, with common data items and those for Amazon and Starbucks all in one large table.

**Recursive Product Hierarchy**

 The line that comes out of the Ref\_Product\_Types Table and then goes back in again is informally called a 'Rabbits Ear',   
or more formally a 'Recursive Relationship'. It defines a Product Hierarchy.

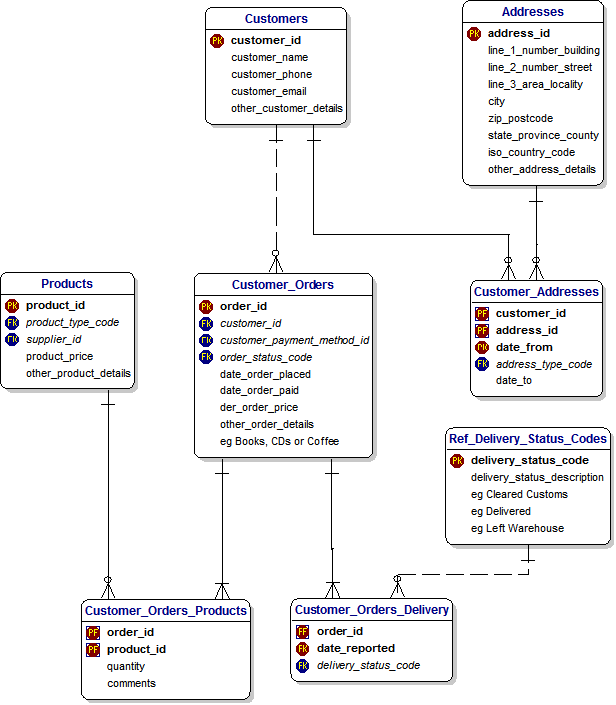
 It means that any Product can be part of a higher grouping of Products, such as Categories, Types and so on.

 This is a very neat and powerful way to implement this concept.

 In our example, the 'parent\_product\_type\_code' is a reference to the next highest level in the hierarchy.

**Deliveries from Amazon.com**   
Deliveries go thru a series of progressive stages, such as (alphabetically) :-

* Cleared Customs
* Delivered
* Left Warehouse



**Deliveries**

 Deliveries is a straightforword area to model.

 We have added just two Tables.

 The 'Customers\_Orders\_Delivery' Table keeps a history of the Stages in the delivery of the Order.

 The 'Ref\_Delivery\_Status\_Codes' tells us what those Stages can be, and they can be displayed in a Drop-down list in an Application.

 As you can see, a series of records will be stored in the 'Customers\_Orders\_Delivery' Table.

 Each record has a different date and time and a different Stage.

 This Table makes it possible to track the Order to final Delivery.