

Intro

This document is in response to forum questions about linking records in databases. I hope this is easy to understand and provides some help to everyone.

Note: I am typing this document as I am thinking, so it might get a bit 'mixed up', and I might ramble on in places ☺. I also throw in some of my own suggestions here and there, but you don't have to agree with (or use) them.

Also you might find books on the subject of Databases that can better explain some of the things I am mentioning here.

Peterpuk
29th March 2007
From the EMERGENCE BASIC Forums

Determining Database Application Requirements

Creating database applications is very involved, and requires a good understanding of how databases should be designed and accessed by the application.

In this document, I am going to use an Order Entry system as an example. To help you understand what the application and database requirements will be, you must get answers to a few questions. Before you start any coding you should research such things as:

- Who is going to use the application?

Is the application going to be used by one person or a number of people?

For example, in the order entry system, one person could be responsible for adding customers, stock and orders. (This is not a recommended approach in a typical business since it does not provide any control mechanisms, and you will not get a favourable report from an auditor. But if you're the only user, then you have no option.)

If you are designing the application for a larger business, then you might have an accounts person that can only add and maintain customer records, a stock control person who can only add and maintain stock records, and an order entry person who can only enter orders.

Are they going to have full access to the data, or restricted access?

For example, an order entry clerk might only be able to add and change a customer's order, but the **supervisor is the only person who can delete** an order.

These will determine part of the application requirement.

- What output do you need to produce?

This question is most important when it comes to database design. The output required will determine what data needs to be in the database, to produce the information for the output.

In the order entry system, you might require a picking slip to be printed for the warehouse (so the order can be picked and despatched), an invoice to be printed to send to the customer, and sales reports (customer sales summary and stock sales summary) to be printed periodically.

On the 'non-priced' picking slip you might need to have special delivery details, and the stock items may need to be printed in sorted order of warehouse location so it's easy for the pickers to pick the order. This means that we must allow for delivery instructions in the 'Order Table' and a warehouse stock location in the 'Stock Table'.

The invoice will be similar to the picking slip but priced, so you need to make sure the 'Order Table' includes a price field (column) which must be saved at the time the order is created. The price should be retrieved from the Stock Table and saved in the Orders Table to ensure the customer is charged the price that was **current at time of order**. And don't forget to keep track of VAT (or GST if your in Australia).

And so it goes...you get the idea.

- What controls should be implemented?

For example, you might need to allow flagging a customer as 'STOP CREDIT' because they haven't paid their bill (so the operator can't create an order for the customer). This means you need a StopCredit field in the Customer Table, and test this field in the order entry part of the application.

You might have a minimum stock quantity for a particular stock item that the customer must order. For example, the minimum order quantity for "Sticky Tape Rolls" might be 50, so if the customer orders 10 it will be bumped up to 50 automatically. This means that the Stock Table must have a Minimum Order Quantity field which is checked during order entry. It also means that the Orders Table must have two fields which save the 'Quantity Ordered' and 'Quantity Supplied' data.

Again, you get the idea. Research what is required.

After having a serious look at the overall requirements of the system you are going to implement, you should end up with a heap of notes and sample printouts that you will refer to (and refine) during development. In most cases you will be converting (computerising) a manual paper system, so you will probably have a lot of printed examples (eg. picking/order slips, invoices, reports on spreadsheets, etc.) to work from. Unfortunately, you will find that the final system is made up of many individual programs that perform various functions, so a good menu system will be required from the start...just something else to think about.

Now, I have included a picture of a printed invoice example for you to refer to while I explain a few things.

PeterPuk's Computer Supplies
12 Mayhem Road
HAPPY HILL, SA, 5991

Phone: 08 8266 4444
Fax: 08 8266 4445
email: sales@peterpuk.com

To: Brian's Books 1567 Main Road Jersey, SA, 5234		Delivery: Shop 11 16 Prescott Way Singleton, SA, 5775		INVOICE 1001 Date: 29/3/07		
Account #: 277		Order #: B0713				
Part Number	Description	Qty Ordered	Qty Supplied	Unit Price Ex Tax	VAT/GST	Total
1001	Windows Vista Home	2	2	350.00	35.00	770.00
2045	DVD-R 50 pack	1	3	14.95	1.50	49.35
						819.35
Total includes \$74.50 VAT/GST						

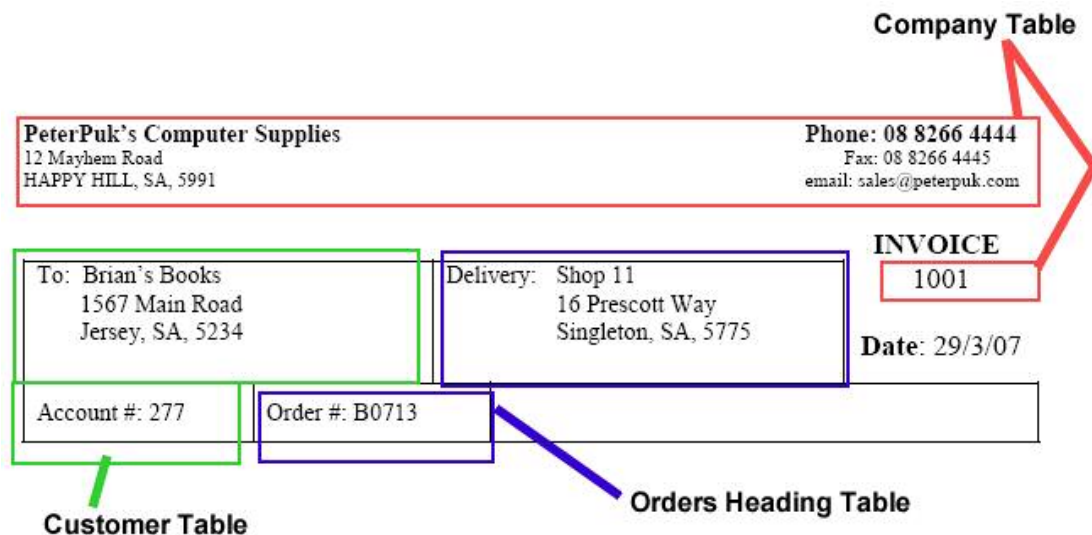
An invoice is made up of a number areas, and the information printed comes from a few Tables in the database. There are also some calculations which are performed by the program and printed (these are not usually saved in the database but can be at a risk, the risk being the calculation isn't updated correctly when making changes).

To break it down, let's start from the top.

- The customers name and address will come from the Customer Table.
- The delivery instructions can change for each order so it must be saved in the Orders Table. However a default delivery instruction can be saved in the Customer Table and displayed in the Orders Entry screen, where it can be edited and saved in the Orders Table.
- The Account # is your account number for this customer, so it comes from the Customer Table. The customer account number can be the link to the Orders Table, but this means that it definitely, positively **MUST BE UNIQUE**. However a better way of linking a customer to an order is to include an AutoIncrementing or Counter field in the Customer Table, and use this value to link a customer to any record in other Tables in the database. More on this later.
- The Order # is the customer's order number that they have given you at the time of ordering. This will be saved in the Orders Table because it relates to the order. You should always print the customer's order number on an invoice, and save it in the Orders Table. Note that this should be a STRING variable because many businesses mix characters with numbers eg B213.
- The Invoice number is a special case, but you might keep track of it in a Company Table (a table that has the details of your business). This table might have only one record that contains data such as

Business Name and address which is printed on invoices and other documents,
Business Phone and other contact details,
Invoice tracking number (which is read at the time of printing an invoice, incremented and saved),
Etc...

The following picture shows what we have covered so far. Note that this can be referred to as the invoice HEADER (or as I call it, Heading). Since an invoice is produced from an order, this leads us to another issue. An order is actually saved in a database in two tables to avoid a massive duplication of data. One table is the Order Heading, and the other is the Order Details. Therefore you have a link between these two Tables, which I will discuss later.



Next we look at the details section of the invoice where the stock items are listed.

Just a point here... This is just a guide, and I expect experienced 'inventory' database developers will probably have other ways of doing things, but the concepts are the same.

Stock Table

Part Number	Description	Qty Ordered	Qty Supplied	Unit Price Ex Tax	VAT/GST	Total
1001	Windows Vista Home	2	2	350.00	35.00	770.00
2045	DVD-R 50 pack	1	3	14.95	1.50	49.35
						819.35
Total includes \$74.50 VAT/GST						

Order Details Table

Calculated during printing

- The Part Number and Description will come from the Stock Table. The Part Number is similar to the Customer # mentioned before, whereby it can be used as the link to the items in the Order Details Table. Again, this means that it definitely, positively **MUST BE UNIQUE**. And again, this is a case where an AutoIncrementing or Counter field would be a better option for linking purposes.
- The Quantity Ordered and Quantity Supplied columns are specific to the order and must be saved in the Order Details Table.
- The Unit Price Ex Tax is a strange one, and you will notice I have shown that it exists in both the Stock Table and Order Details Table. Obviously the price must exist in the Stock Table because it relates to the stock item, and must be retrieved at the time of creating an order. However, it should also be saved as a field in the Order Details Table as well, because this will be the price that must be printed on the invoice, and any sales reports. By doing this, the price at the time of placing the order will always be applied, even if the actual price of the item changes in the Stock Table at a later date.

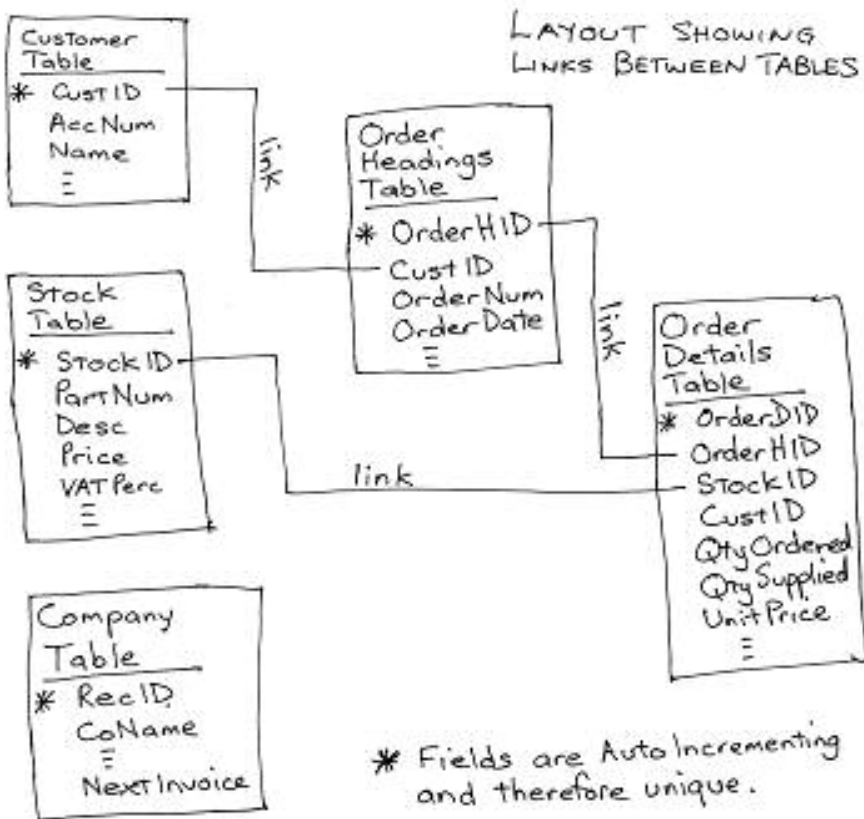
And I thought this would be an easy example..!

- The VAT/GST and invoice totals are calculated during the print procedure. Of course the VAT/GST percentage must exist in, and be retrieved from the Stock Table and applied in the calculation.

Now that we have analysed the output required from the system (at least for orders/invoices), we can start to determine the database requirements.

Determining the Database Requirements

After getting a reasonable idea of what must be stored in the database, you can start to determine the Tables required. Get out a pencil and paper and start by drawing a diagram like the one shown below. As you'll notice, I always allow for an AutoIncrementing field (counter) in each Table which serves two main purposes. Firstly, it used as the link to build relationships between Tables, and secondly it is used as a key in the Primary Index to ensure we don't get 'duplicate' errors in the Table.



The Layout above gives you a simple visual overview of the database we require to develop the Order Entry system. You can now concentrate on deciding what fields (columns) should be included in each Table.

Based on our sample invoice, we know the fields needed to produce picking slips/invoices, but we should think ahead and consider other things like flagging a customer as STOP CREDIT, or flagging a stock item as DELETED, etc. This will allow us to implement controls in the system at a future date. When I use the word control here I mean it in the business sense, not windows controls©.

The following tables show what I have determined to be required for our Order Entry example. The field definitions are based on the Microsoft Access database.

The Database Tables

When you design the layout or schema of the database, you must consider how many Tables are needed to store the data without duplication. By this I mean that you should not have two Tables with the same data in them. For example, if you have the customer's name in the Customer Table, do not put the customer's name in the Orders Table as well. This is a bad design.

It's bad for two main reasons:

- This duplication takes up a lot of disk space and can affect performance.
- It will create a maintenance nightmare if you change the customer's name. Instead of just changing it in the Customer Table, you will have to update the change in every record in the Orders Table as well.

Also a Table is like an object, where it contains only relevant data about that object. Makes sense, doesn't it. This is why you separate customer data in a Customer Table, stock data in a Stock Table and order data in an Orders Table. If you wanted to link each stock item to a supplier, then you would also have a Supplier Table.

Table Name: Customer

Field Name (Column)	Type	Size	Required	Comments
CustID	Counter (int)		Yes	Unique Key - Used in Primary Index. Also used to link with other Tables.
AccCode	Text (string)	6	Yes	Used in Primary Index so it should be unique.
Name	Text (string)	45	Yes	
Addr1	Text (string)	45		
Addr2	Text (string)	45		
Country	Text (string)	40		
Category	Text (string)	2	Yes	A code to allow customer grouping for reports, etc. B=Business E=Education P=Personal
Contact	Text (string)	40		
Phone	Text (string)	20		
Fax	Text (string)	20		
email	Text (string)	100		
DeliveryAddr1	Text (string)	45		Default delivery address line 1
DeliveryAddr2	Text (string)	45		Default delivery address line 2
DeliveryAddr3	Text (string)	45		Default delivery address line 3
DateCreated	Date		Yes	Retrieved from windows - not editable
LastUser	Text (string)	20	Yes	Retrieved from windows - not editable
StopSupply	Integer (int)		Yes	Stop supply flag. 0=OK 1=Stop Supply

Customer Primary Index: AccCode+CustID

Table Name: Stock

Field Name (Column)	Type	Size	Required	Comments
StockID	Counter (int)		Yes	Unique Key - Used in Primary Index. Also used to link with other Tables.
PartNum	Text (string)	10	Yes	Used in Primary Index so it should be unique.
Barcode	Text (String)	13		Barcode number for scanning. Should be text so that you can have leading zeros and dissect into Country/Supplier/Item if required.
Desc	Text (string)	50	Yes	
WHLoc	Text (string)	10		Warehouse location for picking orders. Can be arranged as: Pos Description 1-2 Warehouse number 3-5 Aisle number 6-8 Shelf number 9-10 Bin number
Cost	Single (float)		Yes	Your unit cost in dollars
Sell	Single (float)		Yes	Your unit sell in dollars
VAT	Single (float)		Yes	The percentage to apply for tax. EG. 12.5 = 12.5%
MinOrdQty	Single (float)		Yes	Minimum order quantity. Default to 1
DateCreated	Date		Yes	Retrieved from windows - not editable
LastUser	Text (string)	20	Yes	Retrieved from windows - not editable
Deleted	Integer (int)		Yes	Deleted stock flag. 0=OK 1=Deleted Note: When a stock item is deleted, you just set the flag, not delete the record.

Stock Primary Index: PartNum+StockID

Table Name: OrderH

(This is the Order Heading Table)

Field Name (Column)	Type	Size	Required	Comments
OrderHID	Counter (int)		Yes	Unique Key - Used in Primary Index. Also used to link with other Tables.
CustID	Long (int)		Yes	Used in Primary Index. This field contains the value of CustID from the Customer Table which creates the link.
OrderNum	Text (String)	12		Customer's Order Number.

DeliveryAddr1	Text (string)	45		Delivery address line 1
DeliveryAddr2	Text (string)	45		Delivery address line 2
DeliveryAddr3	Text (string)	45		Delivery address line 3
DateCreated	Date		Yes	Retrieved from windows - not editable
LastUser	Text (string)	20	Yes	Retrieved from windows - not editable
Invoiced	Integer (int)		Yes	Invoice printed flag. 0=Not printed 1=Printed Note: Set this flag during invoice printing.

OrderH Primary Index: CustID+OrderHID

Table Name: OrderD

(This is the Order Details Table)

Field Name (Column)	Type	Size	Required	Comments
OrderDID	Counter (int)		Yes	Unique Key - Used in Primary Index.
OrderHID	Long (int)		Yes	Used in Primary Index. This field contains the value of OrderHID from the OrderH Table which creates the link.
CustID	Long (int)		Yes	The CustID is saved in the Table as a check.
StockID	Long (int)		Yes	This field contains the value of StockID from the Stock Table which creates the link.
QtyOrdered	Single (float)		Yes	Quantity ordered.
QtySupplied	Single (float)		Yes	Quantity supplied. This might be less if there was no stock on hand, or higher if adjusted to the minimum quantity as set in the Stock Table.
Sell	Single (float)		Yes	The unit sell price in dollars Ex VAT
VATAmount	Single (float)		Yes	The unit VAT amount in dollars which will be added to the unit Sell.
DateCreated	Date		Yes	Retrieved from windows - not editable
LastUser	Text (string)	20	Yes	Retrieved from windows - not editable

OrderD Primary Index: OrderHID+OrderDID

Well these simplistic Tables give you the idea, but you would define the fields according to how you will use them. Some of the fields that I have made Single (float) such as the quantities, you might make Long because you don't sell stock in fractions (eg 1.5). If you sell a stock item by the metre, and the customer can buy 1.5 metres, then you need to make the field a Single as I have done.

But as with all programming, there are many ways to produce the same result. Some ways are just easier than others. By the way, I didn't include a Company Table because I only mentioned it as an option.

Linking Tables in the Database

If you have a database with only one Table, then you have no need for links. However once you start adding more Tables that somehow relate to each other (like our Orders example), then you need to provide a linking system.

Believe it or not, linking the data between Tables is simple. You just need to grasp the concept.

Generally, most Tables that you include in a database will have a 'one to many' link, which means that one record in one Table will link to many records in another Table. This is the case in our Order Entry example.

In the Customer Table, you have only one record per customer (like a master Table). In the Orders Table, you have many records that belong to a customer (like a transactions Table). To find all the Order records for a customer, you need to have a 'customer' link between the two Tables.

So what is the link?

A link is a field (or column in database talk) that you include in your Tables for the purpose of linking Tables. I call these fields 'record ids', because in each record the data (or value) in this field WILL BE UNIQUE. Relational databases have a data type that is auto incrementing, which means that every time you add a record to the table, the number that is saved in this field is the next record number as allocated by the database engine. This field will always be unique for each record, and you cannot change it's value...perfect for linking purposes.

As you recall earlier in this document, I included a picture of my hand drawn database layout showing the links between Tables. You will notice that the first field in every Table was an auto incrementing field (marked with an *). I strongly suggest that you do the same, even if you don't think it's required. The reason for this is that even if you don't use this field for linking purposes, you will definitely need it in the Primary Index of the Table (and every Table must have a Primary Index).

Note: You only ever have one field that is an auto incrementing data type in a Table.

How are these link fields actually used to link Tables?

Ok. In every Table you have included the auto incrementing field as the first field. I always end the field name with ID, as in CustID, StockID, OrderHID, etc.

If we add our first customer to the Customer Table, the value in CustID will automatically be 1 (one). Let's say the customer name is Peter.

Now we want to create an order for Peter (and link it to Peter). Note that I am only going to save the Order Heading in the following steps, not the stock items as well.

STEP 1

In our imaginary Order Entry program, we first get the user to find the record for 'Peter' from the Customer Table. In the program code, we define a variable called curCustID to hold the value of CustID, and bind this variable to the recordset CustID field.

Eg. DEF curCustID as INT

Now when the customer record for Peter is the current record, curCustID has the value 1 (one). *Don't let the user manually change the value of curCustID.*

STEP 2

The user now enters the Order Heading data into editboxes. They enter the Order Number and Delivery Instructions.

STEP 3

The user now clicks the [Save Order] button.

In the program code, you must bind the curCustID variable to the CustID field in the OrderH Table before saving. This will save the value 1 (one) as the CustID in the OrderH Table, thus creating the link. *This link is between Customer and OrderH Tables.*

The same process applies to adding stock items to the order.

You first find the Order Heading record from the OrderH Table, and make it the current record.

Bind the OrderHID field to a variable curOrderHID.

In a loop, add each stock item, making sure you bind the curOrderHID with the OrderHID field in the Order Details Table (OrderD). *This link is between OrderH and OrderD Tables.*

Phew, trying to describe linking is harder than I thought.

To summarise all of the above, to create a link between records in two Tables, you get the record ID from the record in one Table and save it in a field in the record of the second Table.

How do I use the links in the real world?

Let's say you want to retrieve a dataset of all orders for a customer (Peter in this example).

STEP 1

First get the user to find the record for 'Peter' in the Customer Table. Save the CustID field value in the curCustID variable as before.

STEP 2

Execute the SQL statement:

```
hStmt = dbExecSQL(pdb, "SELECT * FROM OrderH WHERE CustID="+str$(curCustID))
```

STEP 3

Load a ListView control with each record while looping through the hStmt recordset.

Anyhow, I'll leave it at that for now and hope you get some value from this.