

VISUAL BASIC DATABASE TUTORIAL

BY KARL MOORE

PART ONE

INTRODUCTION

Welcome to the very first part in this wizzy Visual Basic Database tutorial!

I'm your crackingly cool host, Karl Moore and over the next few weeks we'll be covering everything you ever wanted to know about databases, but were too afraid to ask.

Whether you're wanting a Christmas bonus or just need to boost your Visual Basic knowledge – I'm here to help.

Don't forget that I love to hear feedback! Feel free to send me a message, abusive or otherwise by clicking on the "Post Feedback" link at the end of the page.

Now let's get that wizzo brain cap on – as we prepare to answer the following searching questions:

- Erm... so what's a database?
- Why don't those tables have any legs?
- What's Microsoft Access got to do with it?
- Why is the grass green?
- How do I build my own mini database program in VB?
- What is the meaning of life?

Well, let's get the philosophical stuff out of the way first. The meaning of life is 42 and grass is green 'cause all the other rainbow colours are absorbed and only the green reflected, a process known as subtractive colour mixing.

Ha - and they said this tutorial wasn't going to be interesting!

(Ed groans)

WHAT'S A DATABASE?

When I first entered the geeky database scene, I shivered at the "d" word. Eugh, who wants to play around with databases? Certainly not me, I just wanted to program.

But you soon realise no matter what type of program you're creating, databases can be pretty cool things.

A database is essentially just a store of information. They usually come in the form of a simple file (just like a Microsoft Word file, say). You can shove information into this store or retrieve it from the store, with virtually no code at all.

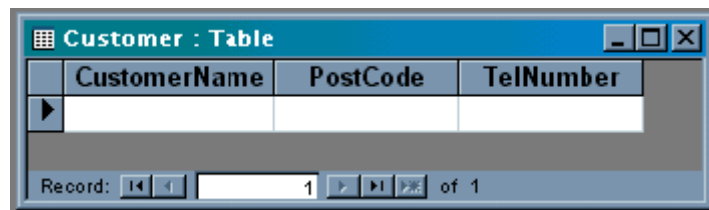
Top Tip: You can make friends and impress the opposite sex at geeky cocktail parties by saying "DB" instead of database.

Hmm, doesn't sound terribly complex does it? Erm, that's because it's not. Most database wizards just like to overcomplicate things in a bid to scare off any programming newbies.

Now, listen up. A database may include many different tables. You can imagine a table as one worksheet in an Excel workbook.

Each column of the worksheet may hold something different. Column A, for example, could hold a customer name, column B may hold the customer postal code and column C may hold the customer telephone number.

That's all a table is – a set of pre-defined "slots" or "boxes" into which you throw information. Each of those boxes has a descriptive name, such as "TelNumber".



Because we're supercool geeks, we're not allowed to call those boxes... err, boxes. We need to call them "fields", because it's the done thing and boosts your street-cred.

Just like an Excel worksheet, you can add new entries to the table – by simply filling in a new set of fields. So each time you need to add a new customer, you just fill in the "CustomerName", "CustomerPostCode" and "CustomerTelephoneNumber" fields and Bob's your Uncle!

Actually, he is my uncle – but that's totally irrelevant to the current discussion.

	CustomerName	PostCode	TelNumber
	Karl Moore	EC1 9XL	0845 333 8144
	Chen Chun Chak	HK32 6SU	01923 6

Record: 1 2 of 2

Top Tip: You cannot eat your dinner off a database table. This is considered highly uncool in the database world. Also, it's inadvisable to graze sheep in a table field.

OK, let's run over those wizzy terms once more:

- **Database** – a bunch of tables
- **Tables** – store numerous rows of information
- **Fields** – the little boxes inside a table

A database can also contain relationships and queries. You can even have a relationship with a query, but the Church doesn't commend it. Either way, that's pretty geeky stuff – so we'll cover it later.

LET ME LOOSE!

Don't tell me, your boss has asked you to develop a supercool database system and handed you a deadline of yesterday.


You don't want to hear me babble about database theory - you need to get stuck in! Well, I can take a hint – so let's get down and dirty, and develop our first Visual Basic database-integrated application... with absolutely no code at all!

1. Start Visual Basic – if you're unsure about this, check out my Visual Basic tutorial [here](#).



Standard EXE

2. Choose the "Standard EXE" option and click OK

3. A regular blank form should appear on your screen. Double-click on the data control in your Toolbox – the one that looks like this . It should appear on your form.

4. Change the Name property of the data control to "datCustomers".

5. Click on the ellipsis next to the DatabaseName property of the data control and select the "Nwind.mdb" file. Mine is located at C:\Program Files\Microsoft Visual Basic\VB98\Nwind.mdb – though yours may be different. This is a sample database file that is distributed with Visual Basic!

6. Move a little further down in the Properties window and click on the RecordSource button. After a few seconds, a list will appear – these are all the tables and queries in your database. The one entitled "Customers" is a table. Click it!

7. Now add three text boxes to your form – name them txtCompanyName, txtContactName and txtPostalCode respectively.

8. Change the DataSource property for each of the text boxes to the name of your data control (probably datCustomers, if you've been good!)

9. For each of the separate text boxes, click the DataField property – you will see a list of words appear. These are all the fields within the 'Customers' database table. Change this property for each of your three text boxes as so:

- txtCompanyName - CompanyName
- txtContactName - ContactName
- txtPostalCode - PostalCode

10. Now hit F5 to run your program!

Congratulations! You've just created your very first database application!

Try clicking those buttons on the data control. They'll move you backwards and forwards among the rows in that Customers table.

A screenshot of a Visual Basic form titled "Form1". The form has a light blue background. In the center, there is a data control displaying three rows of customer information. The first row contains "Eastern Connection", the second row contains "Ann Devon", and the third row contains "WXC 6FW". At the bottom of the form, there are four small navigation buttons: a double left arrow, a single left arrow, a single right arrow, and a double right arrow.

If you're likening this to an Excel spreadsheet, you can imagine that with each click you're moving down or up one row and displaying all the customer information on that line.

Try changing one of the company names and moving forward a few records – then moving back. You should notice that your changes have been saved!

That's what a database table is all about. They allow you to add "rows" of information to a table, edit stuff currently sitting in the table, remove entire rows or bunches of rows from a table... even find rows in a table!

Top Tip: *Instead of using the word "row" to describe a line of information in a table, try using the term "record" instead. Don't ask why, just trust me... I'm a programmer.*

CREATING A DATABASE

That's all fine and dandy, but how do you actually create a database?

Unfortunately, just to confuse us all, there are many different types of databases. First off, there's an expensive whopper of a database system called SQL Server, which is used in corporations that need to store huge wads of information. There's also that thing they call Oracle, which is another database format.

But one of the most exciting (and cheap!) types of database is an Access database. Remember when we changed the DatabaseName property of our data control to "Nwind.mdb"?

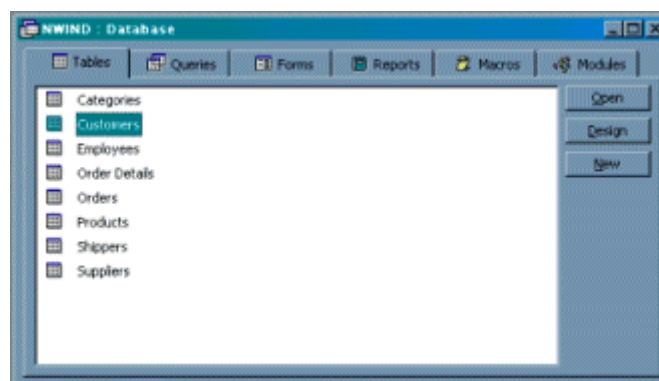
That Nwind.mdb file is actually an Access database.

The best way to create your own Access database is by using, unsurprisingly, Microsoft Access. This tool ships with the professional editions of Office 97 and enables you to graphically design your own tables and individual fields.

If you don't already have Microsoft Access, add it to your Christmas list – and be a good boy. If you do have it, brilliant!

We'll delve into the intricacies of creating your own database next week – but for now let's explore the existing Nwind.mdb database:

1. Click on Start, Programs, Microsoft Access
2. Select "More Files..." then click OK
3. At the Open dialog box, select the Nwind.mdb file – don't forget, by default it is installed at C:\Program Files\Microsoft Visual Studio\VB98\Nwind.mdb
4. Click OK when the boring "Welcome to Northwind" screen appears
5. A box should appear on your screen, looking a little like this:




6. Click on the tab entitled "Tables". These are all the tables in your database – including the Customers one we browsed earlier in this tutorial
7. Double-click on the Customers table. You should see something like this:

Customers Table				
Customer ID	Company Name	Contact Name	Contact Title	
ALFK	Alfreds Futterkiste	Maria Anders	Sales Representative	Oslo
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Aida
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mata
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 H
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Bergu
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forst
BOLID	Bonold père et fils	Frédérique Citeaux	Marketing Manager	24, pl
BOLID	Bonold Comidas preparadas	Martin Sommer	Owner	C/ As
BONAP	Bon app'	Laurence Labihan	Owner	12, ru
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	29 To
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Faunt
CACTU	Cardes Comidas para llevar	Patricia Simpson	Sales Agent	Cemb
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Siera
CHOPS	Chop-suey Chinese	Yang Wang	Owner	Haupt
COMM	Comércio Mineiro	Pedro Alonso	Sales Associate	Av. d
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative	Berke
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator	Wals
DUMON	Du monde entier	Janine Labruno	Owner	67, ru
EASTC	Eastern Connection	Ann Devon	Sales Agent	36 Ki
ERNSH	Ernst Handel	Roland Mendel	Sales Manager	Kirch
FAMIA	Familia Arquibaldo	Ana Cruz	Marketing Assistant	Rua C
FISSA	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	Accounting Manager	C/ M

That's a table! The names at the very top of the screen are field names. Under that, you'll find all the actual information. Each singular row of information is known as a record. If you move right down to the bottom of the table, you can also add new customers to the Customers table.

Try it now!

1. Click on the "Add New"  button on the frame around the customer data
2. Fill in all the boxes for the new record, inventing wacky company names and telephone numbers as you go along

KARLM	Karl's Super-Duper Enterprises	Karlos Moore	Head Honcho	
Record: 101 of 102				

3. Now close the table and re-open it
4. Scour down the list and you should now see your customer listed alphabetically
5. Click on any field in your record, then click Edit, Delete Record
6. You should be asked if you want to delete the selected record. Click Yes... and it should disappear!

Well done – you've just looked behind the scenes of a real database, added a record and then manually removed it!

CONCLUSION

This week, we've figured out exactly what a database is and how to create a simple link between an Access file and your Visual Basic application. We also edited a record, added a new record and deleted a record! Heck, we even learned about subtractive colour mixing.

Next week, the tutorial will get even juicier. Yessiree, we'll:

- Create our own database
- Knock together our own tables
- Define our own fields
- Figure out the real meaning of life
- Discover what 'Structured Query Language' is all about
- Plus.. take a peek at queries!

Until then, I'll say cheerio and toodle-pip - this is Karl Moore signing off for tonight. Goodnight!

PART TWO

INTRODUCTION

Hey nonny nonny and welcome to the second part in this wizzy Visual Basic Database tutorial.

Just in case you've forgotten, I'm your shockingly handsome host Karl Moore - and if you missed part one, read it now.

Don't forget that I'd love to hear what you think of this series - if you consider it terrible, tell us. I won't be offended. Much. Just use the comments section.

Anyway, enough small talk. This week we'll be taking a geek-peek at:

- The incredibly nerdy Structured Query Language
- Creating our own live VB querying program
- Producing SQL statements... the easy way!

So without further ado, my fellow geeks and geekesses, let's proceed onwards and upwards...

TALK TO ME, PLEASE!

My psychologist always tells me to imagine smooth relaxing sands and blue rippling seas, as I unveil the maniacal details of my crazed programming life. And that's just what we'll be doing in this section - except we don't have a couch. Erm, nor an official Institute of Psychology certificate.

But let's just imagine you wanted to talk to your database - perhaps you want to ask it the telephone number of one of the contacts in your table.

Top Tip: *Just in case your memory is as useful as the Pope's wedding tackle, don't forget that a field is a 'box' inside a table - numerous fields make up one table. One or more tables make up a database.*

In a perfect world, we'd be able to say "What's the telephone number for Johnny Briggs?" - and the computer would reply, "It's 517-000-238, Karl, you stunningly wonderful person".

Welcome to an imperfect world.

In the world of databases, we talk in a language called SQL, or Structured Query Language.

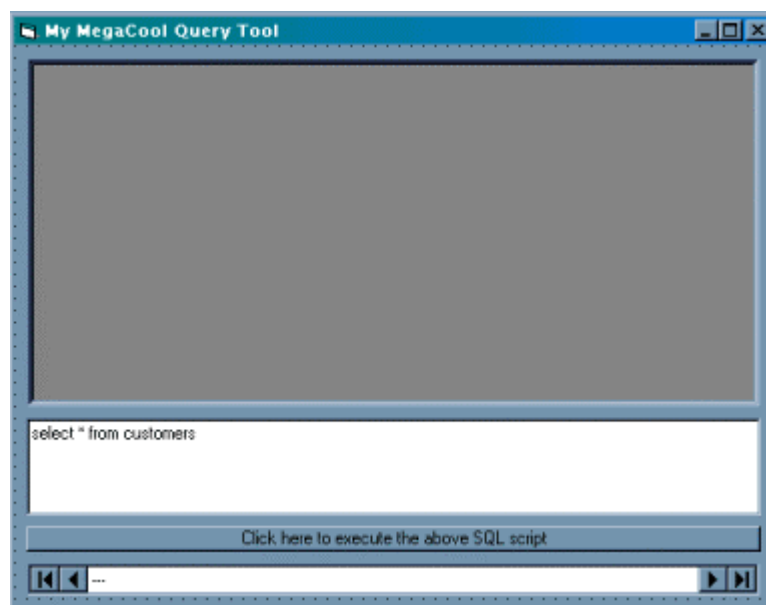
Top Tip: *Never call SQL "Structured Query Language", unless you're writing a report - or are called Karl Moore. Phrases like "I'm just writing a lil' bit of SQL" sound much more impressive and are guaranteed to improve your social status.*

To pick a random example, a sentence such as "Show me all the contact names" in English, translates into something like "select ContactName from Contacts" in SQL.

LET'S CODE, GEEK-TO-GEEK

In order to demonstrate the SQL language, boot up Visual Basic and follow these steps to create our own mini-querying tool:

1. Create a Standard Exe
2. Click Project, Components - in the list that appears, check the item entitled 'Microsoft FlexGrid Control 6.0' (or similar) and click OK. The FlexGrid control allows us to easily display the answers to our database questions
3. Next, add the following controls onto your form so your design roughly matches the below image. Don't worry about naming conventions or whatever at the moment.
 - Text Box
 - Command Button
 - Data control
 - MSFlexGrid



4. In the Properties window for the FlexGrid control, click on the 'DataSource' property. A list of Data controls will appear - it should just contain the 'Data1' control you added above. Select it.
5. Change the DatabaseName property of the Data control to point to that Nwind.mdb database we talked about last week. Mine is located at "C:\Program Files\Microsoft Visual Studio\VB98\Nwind.mdb"

6. Now insert the following code behind your command button:

```
On Error GoTo OhDear

    Data1.RecordSource = Text1.Text
    Data1.Refresh

Exit Sub

OhDear:
    MsgBox "Euston, we have a problem!"
```

7. Err, that's it!

Brilliant. Now let me briefly explain what we just did.

A BRIEF EXPLANATION

First off, we added the FlexGrid component to our project. This is just another control - like the CommandButton or CheckBox controls - that allows us to add a bit of specific functionality to our project.

The FlexGrid control allows us to very simply display all the records currently held by the Data control. In other words, if we were to tell the Data control to look directly at the Customers table (by changing its RecordSource property to Customers), the FlexGrid would display field in the Customer table.

Because we're true database divas, we're going to get all technical now. We're going to ask the database a question in SQL and tell it to display the results in the FlexGrid. That's where the bit of code steps in.

What does the code do? It simply sets the RecordSource property of the Data control to whatever is sitting in your text box. So if the text box held the text 'Customers', it would fetch everything from the Customers table. But if the text box contained an SQL question, likewise, it will ask the question and display the results.

The second line of control then simply refreshes the Data control - which means all linked items, such as the FlexGrid - are kicked and told to update themselves.

TIRED YET?

Phew! I'm glad that's out of the way. Hit F5 to run the application!

Type 'Customers' in the text box and hit your command button. You should notice that your grid has suddenly filled with dozens of weird-sounding businesses.

That's brilliant... but let's pretend we only wanted to show your user the contact names for each company. Ahh. Problem.

Actually, it isn't. We can do this using one of those database questions in Structured Query Language!

Top Tip: Don't call database questions 'database questions'. Sure I might do it, but that's because this is **my** tutorial and I have permission. You should probably call them **SQL statements**

OK, tap the following into your text box and click the command button:

```
Select ContactName from Customers
```

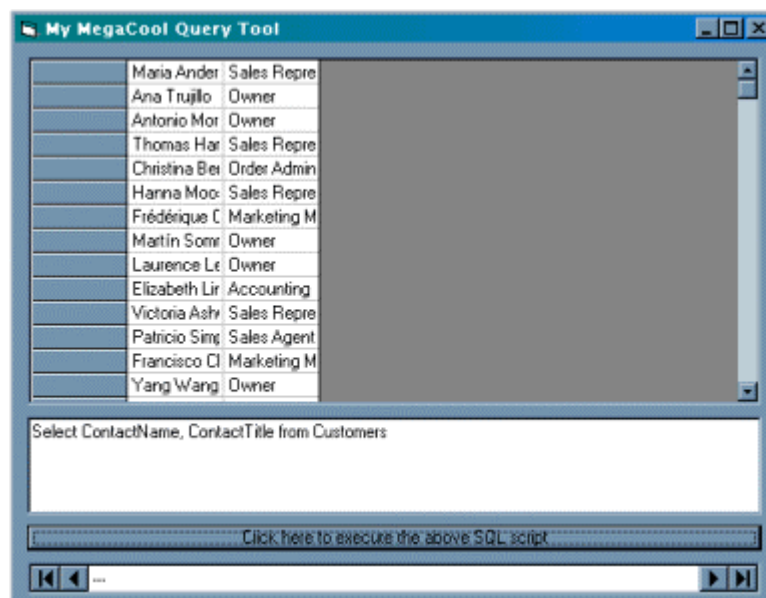
Don't worry about capitalisation and the like - just tap it in and hit the button. You should've got a list of all the contact names - don't worry if the FlexGrid is cutting off the end of names, that's just a sizing issue.

In English, this statement is saying to the database: "Give me everything from the 'ContactName' field, which lies inside the Customers table... OK?"

Here's another example:

```
Select ContactName, ContactTitle from Customers
```

This one is saying "Show me everything in the 'ContactName' and 'ContactTitle' fields, which are hiding in the Customers table... now!"



But just before you start telling all your friends how simple SQL is... errm, don't.
It can get much more complicated... yah booh, sucks to you, SQL!

WHO YOU CALLING A CRITERION?

You pick up the telephone and the caller says, "Grassy ass! My name is Maurizio Moroni and I have a question, pleasssse!"

Hmm, you think. You'd really like to get a job with some cool company - so if this dodgy sounding Maurizio is the owner of a customer company, then you'll probably want to sweet talk him. Of course, if he's just a sales representative or some such, you'll want to tell him to stop bothering you.

And wahay, SQL can step in here to help. Let's take a peek at another SQL statement:

```
Select ContactTitle from Customers
where ContactName="Maurizio Moroni"
```

This one is saying "Show me the contents of the 'ContactTitle' field where the 'ContactName' is Maurizio Moroni"

And what is he? Just a sales representative, so you'd better hang up that telephone. Here's a challenge - tell me what position Marie Bertrand holds - and for which company!

So, that "where FieldName=Criteria" bit is amazingly useful yet surprisingly simple. Except, I bet you were wondering about those "quotes" weren't you? Why do we have to enclose Maurizio's name in quotation marks?

When programming in Visual Basic, you enclose strings in "quotes". And it's snap-snap-snippet-snap in SQL. Similarly, numeric values are not enclosed by "quotes"... not in Visual Basic nor in SQL, for example:

```
Select UnitPrice from [Order Details]
where OrderID=10254
```

It's all about data types - if the database recognises a particular field as a text field, it expects "quotes". If it's numeric, it doesn't. If it's a date, it's completely different. Don't worry about this for now - we'll figure an easier way of creating SQL statements in the next section.

Crikey, I'm babbling aren't I? Am I side-tracking? Is Karlos off on another tangent? Err, possibly, yes.

Well, I'll shutup now. In the next section, we'll be creating more of these statements... but in a really cool way that takes virtually no effort. Sounds good to me...

SQL, THE EASY WAY

Do you want the bad news or the good news?

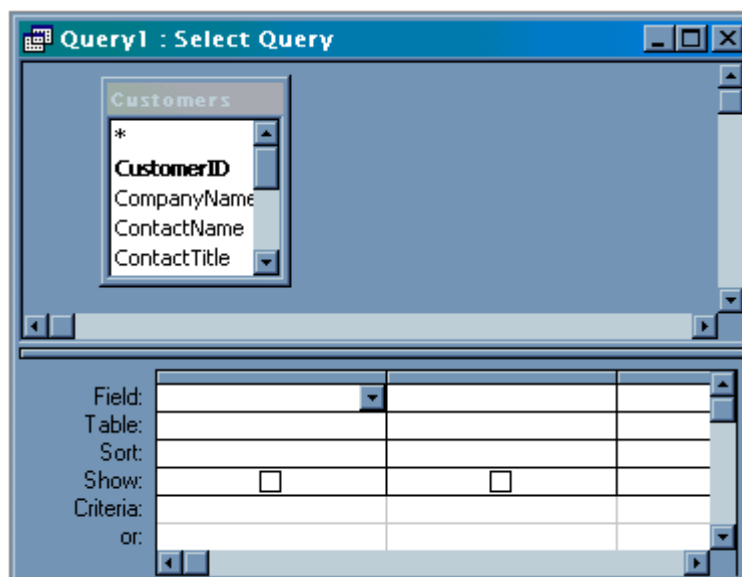
Hmm, bad news first. I thought, being a personal friend of mine, you ought to know that I'm actually listening to a Barry Manilow CD as I type. I know, I know... but some of his songs really aren't that bad. Oh, and I don't mind Chris Rea neither.

Oh boy, I'm never going to get that date I wanted back in the [Beginning Visual Basic tutorial](#), am I?

Anyway, enough of that - here's a bit of good news. Access has a built-in translator that allows you to design those SQL statements using a really groovy user interface - and yes, that means you never really have to learn about SQL, you idle person you.

Start Microsoft Access and open the Nwind.mdb database we used in the last section. Click on the Queries tab, then select Design View and click OK. When prompted to add a table, select Customers, click Add, then Close.

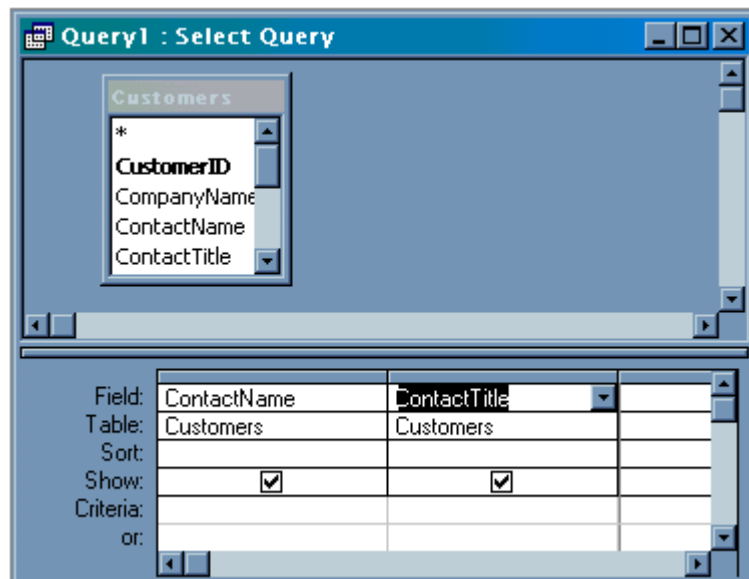
Your screen should look something like this:



Despite all those strange asterisk symbols and criteria boxes lying all over the place, you'll soon get the hang of using this 'query' builder.

Top Tip: *If you have a sore throat, try drinking warm honey mixed with fresh lemon juice.*

Double click on the ContactName field in the 'Customers' table. Then double click the ContactTitle field. They should both appear on your screen, as so:



You've just designed your first query. You're asking the database "Show me everything in the 'ContactName' and 'ContactTitle' fields in the Customers table" - just as we did in the last section.

Click on View, Datasheet View. Your screen should look something like this:

Contact Name	Contact Title
Maria Anders	Sales Representative
Ana Trujillo	Owner
Antonio Moreno	Owner
Thomas Hardy	Sales Representative
Christina Berglund	Order Administrator
Hanna Moos	Sales Representative
Frédérique Citeaux	Marketing Manager
Martin Sommer	Owner
Laurence Leblanc	Owner
Elizabeth Lincoln	Accounting Manager
Victoria Ashworth	Sales Representative
Patricio Simpson	Sales Agent

Now click on View, SQL View. You should see this:

```
SELECT Customers.ContactName, Customers.ContactTitle
FROM Customers;
```

"What??", I hear you cry. Yes, with just a couple of clicks you have automatically generated the SQL statements we poured over in the last section.

Our original statement looked like this:

```
Select ContactName, ContactTitle from Customers
```

But both statements - the manual SQL and Access-generated SQL - do exactly the same thing. So when generating more complex queries, you might as well head straight for Access... and do it the easy way!

CONCLUSION

This week we covered a lot of ground; we've demystified the horrid world of Structured Query Language, and figured how to create simple SQL statements - the easy way. Trust me, that's a lot of stuff in the database world.

What will we be learning next week? Oh, don't ask me. It's Saturday morning, I'm on my third cup of coffee and desperately trying to get over the night before... I really don't plan that far ahead!

Though if you were to really push me, I'd expect we'll probably delve into SQL just a leeeetle more, then sneak off into the crazy world of queries.

But until next week, whatever it may have in store, this is Karl Moore signing off - Goodnight!

PART THREE

INTRODUCTION

With a zip and a zap and a zibidy-doink, welcome to the third part of this wizzy Visual Basic database tutorial.

As ever, I'm your rather geeky host, Karl Moore - and if you've missed the previous two instalments, be sure to check them out before continuing.

Oh, and don't forget to tell us how we're doing. If you love the series, tell us using the below feedback form. If you hate the series, tell us using the comments section. If you're after a date, e-mail me - Karl@karlmoore.com ;-))

But hey-ho, enough patter - time to get on with the real stuff! This week, we'll be covering:

- Creating SQL statements in Access
- How old the earth is
- Storing statements as "queries"
- Making a **Real** Visual Basic Database Application!

Let's start with geographical affairs; the earth is approximately 4.6 billion years old. Or is it 6.4 billion years? Either way, it's about the same age as my Grandma. Err, just kiddin' Grandma!

[Ed: No sweets for you this weekend, Karl...]

Enough of this rambling - what about the nerdy stuff? Corr, I can see you getting excited already. So fellow geeks and geekesses, lend me your ears (plus any money you have lying about)... and let us proceed!

MAKE IT EASY ON YOURSELF

I love taking shortcuts; using super-glue to attach those new shelves, stealing code from the Web for my latest project... and using Microsoft Access to generate my SQL statements.

You may remember that we finished the last instalment telling you how to generate simple SQL statements in Microsoft Access, as opposed to writing them by hand (boring!).

Top Tip: Remember, an "SQL statement" is just a database question - for example, "select ContactName from Customers" means "Show me the contents of every 'ContactName' field in the Customers table"

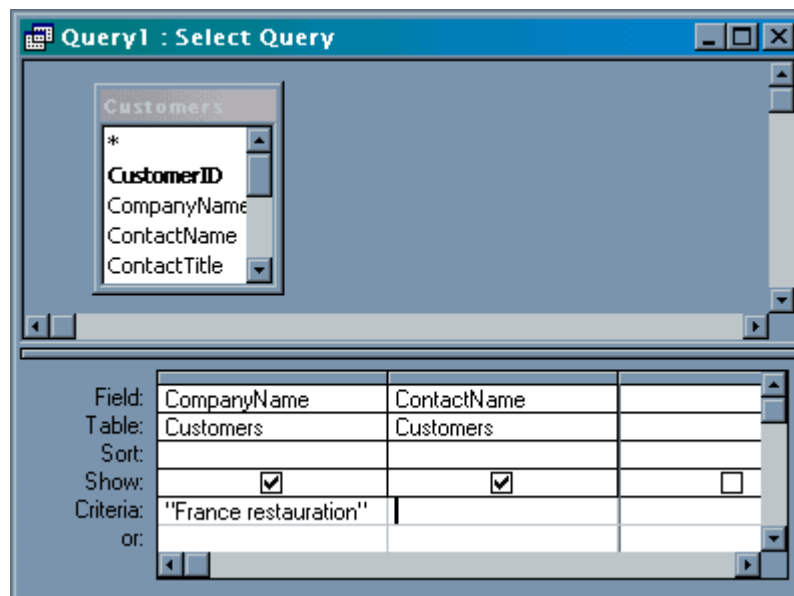
Well, in this section, I'll be giving you a few brief pointers on how to go that little bit further. I shan't drone on about this for ages and pages, but the tips you learn now will help in our later projects. So let's continue where we left off...

SPECIFYING CRITERIA

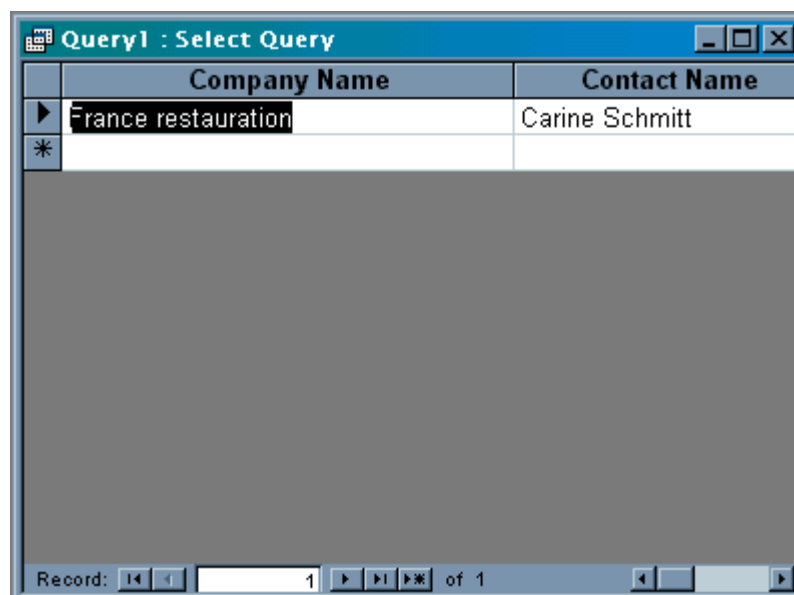
Sometimes you don't want a database to return every single record. For example, you may simply want to show the orders for one particular customer.

In Microsoft Access, you can specify criteria for a particular field by placing the value underneath the fieldname.

For example, the below query on our Nwind.mdb database shows the ContactName for records where the CompanyName is equal to *"France restauration"*



When we enter Datasheet View, we get this:



And when we nip into SQL View, we get this:

```
SELECT Customers.CompanyName, Customers.ContactName
FROM Customers
WHERE (((Customers.CompanyName)="France restauration"));
```

And that's just saved us having to write and debug our own SQL statement. Access did all that hard work for us.

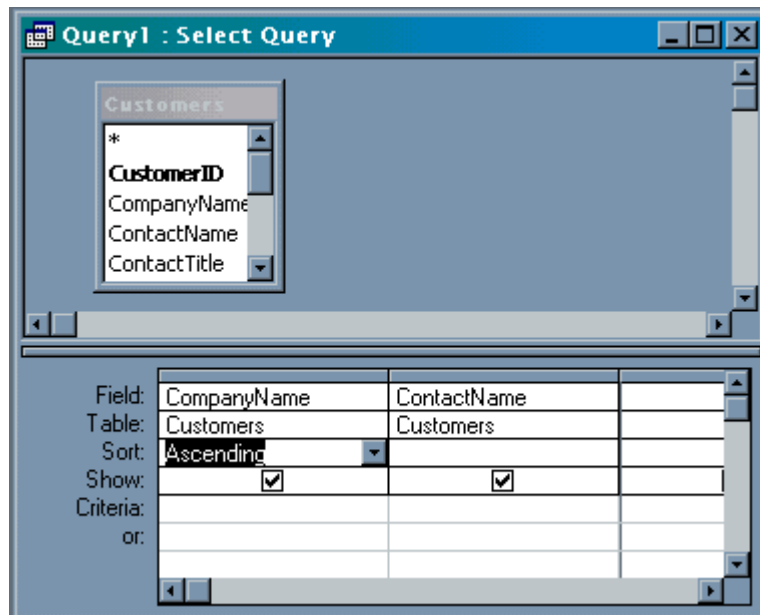
Top Tip: *You will often use SQL statements in your Visual Basic applications. We'll be using them later in this week's tutorial...*

SORTING RECORDS

OK, so you've produced that list of customers - and now your boss wants them in alphabetical order. Argh! Don't you just want to **murder** picky users?

Still, once again, Microsoft Access can help out.

Change the query we created in the last section to look like the below. I've simply removed the "France restauration" criterion and changed the "Sort" box to "Ascending"



Now run the query by entering into Datasheet View. You should find that all the companies are now listed in alphabetical order.

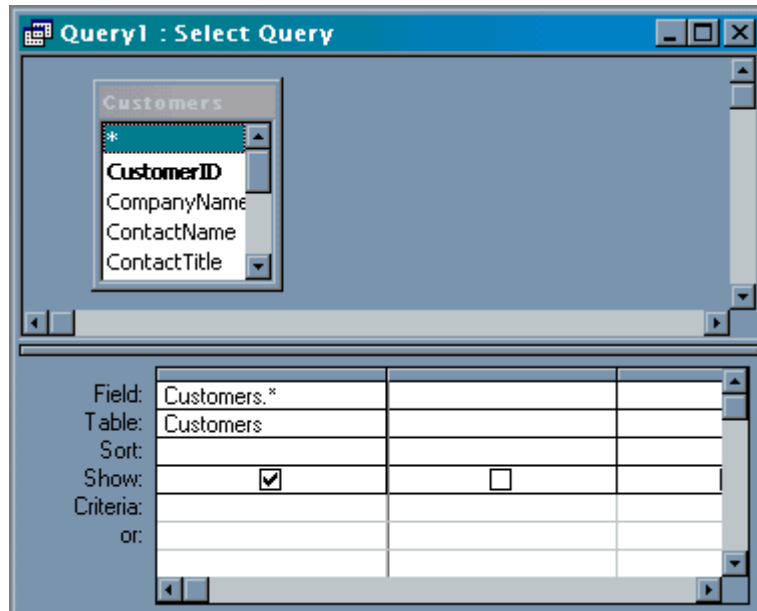
The SQL statement for this looks like:

```
SELECT Customers.CompanyName, Customers.ContactName
FROM Customers
ORDER BY Customers.CompanyName;
```

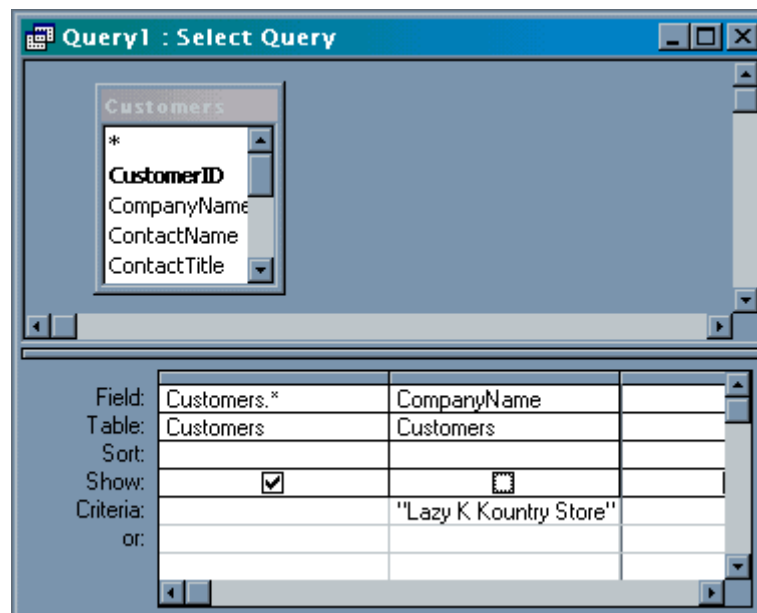
You can even reverse that order by changing 'Ascending' to 'Descending'.

DISPLAYING EVERYTHING

To return all the fields in a table, we can double click on the asterisk in the Customer table, like so:



This tells Access you want to return every field (*) from the table. You could then perhaps add criteria to the query, like this:



The SQL statement for this query is:

```
SELECT Customers.*  
FROM Customers  
WHERE (((Customers.CompanyName)="Lazy K Kountry Store"));
```

OK, enough babbling about Structured Query Language - if you're interested in how Access can help you design queries that count records, sum field values and so on, press F1 whilst designing your query to launch the help.

Go on, press it! I promise I won't tell anyone. Except perhaps my Uncle Bob. And his wife. And his wife's personal assistant. And his wife's personal assistant's pet snail, Strangely Brown. And his wife's personal assistant's pet snail's in-house lover, Also Curiously Brown.

On a related note, try looking up the "=Date()" function and using it as criteria in a query.

PAH! QUERIES, FEARIES...

Hmm, queries, queries, queries. Remember that SQL program we created just last week? The one into which we type a few "select * from customers" statements and clicked a button to view the results?

Well, a query is basically one of those SQL questions... **stored** inside a database, under a particular name.

If you've still got the last Access query open, hit File and Save. Tap in the name "qryContacts" and click OK.

Erm, there we have it. You've just created your first query.

Close the current window and returned to the "Nwind: Database" screen. Take a peek under the 'Queries' tab and double-click on your query. Wallah!

A query is just like a table - although in realistic terms, it probably only grabs a few elite records matching your criteria. But it can be treated **just** like a table.

For example, if you pop back into that SQL program we created and type in "select * from qryContacts" - you'll find it will display everything returned by the qryContacts query.

Or you could try simply stating, "select ContactName from qryContacts" - which would only return the ContactName fields from the query. Go on, try it!

Top Tip: Some database whizz-kids also call queries, 'views'. Just nod and ignore them... after all, **we** know who's right, don't we?

The advantage of queries is they allow you to store SQL statements outside of Visual Basic. Let's say you had a customer order table with a True/False value depending on whether payment had been received. You could create a query that shows you all the customers that haven't paid - and display that in your program - without having to store all the SQL in Visual Basic.

And if you ever need to make changes to the query - you can simply edit it. No recompiling required!

Well that's enough of that. Next, let's dive into Visual Basic and create our first **real** database-integrated application!

CREATING YOUR CUSTOMER BROWSER APP

In this section, we'll be putting everything we've learned so far into practice. Except the bit about tectonic plates. Yessiree, we'll be creating our very own supercool customer browser application!

Before we jump straight into the dirt, let me outline the project. We'll be creating an application that allows you to browse through each customer and view general details of each order they have placed.

On top of this, we'll add functionality to add, delete and search for customers.

Anyway, you want to design a form that looks something like this:

OrderDate	RequiredDate
25/09/95	23/10/95
03/11/95	01/12/95
13/11/95	25/12/95
15/02/96	14/03/96
15/04/96	27/05/96

OK, so it ain't gonna win any design awards, but I'm no Christian Dior.

Now you could design and set all the properties for this project from scratch, but I wouldn't advise it - particularly when you can download the form by simply [clicking here](#).

BORING EXPLANATION BIT

Let me explain how I created the Customer Browser. Well, it all started when I bought a computer and installed Windows 98. Then I purchased Visual Basic and played around a bit. Then I read a few books, sat a few courses and watched a few videos. Then I did some other things.

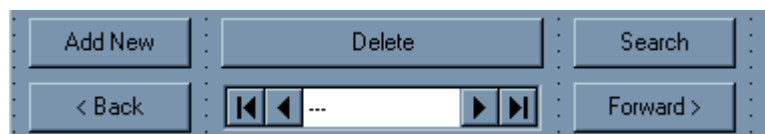
And then I created the Customer Browser. How?



First off, I added a Data control and changed the DatabaseName property to point to my Nwind.mdb database. I also changed the RecordSource property to the Customers table.



Then I added individual text boxes and "bound them" direct to the Data control. I did this by changing the DataSource property to the name of the above Data control, then changing the DataField property to the table field I wanted to 'connect' this text box with.

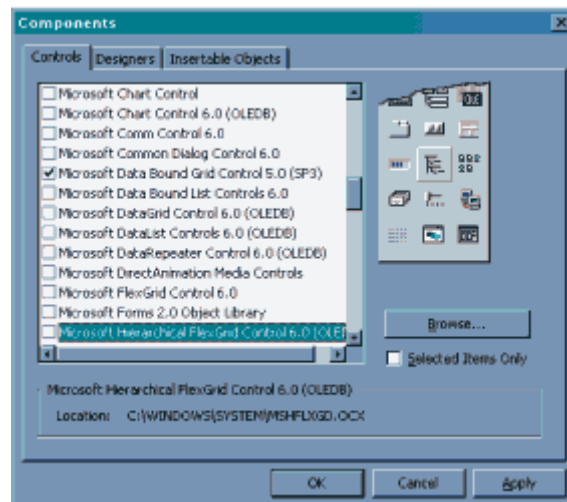


Next, I added six boring (**really** boring!) command buttons and changed their Caption properties. I'll add code behind these later.

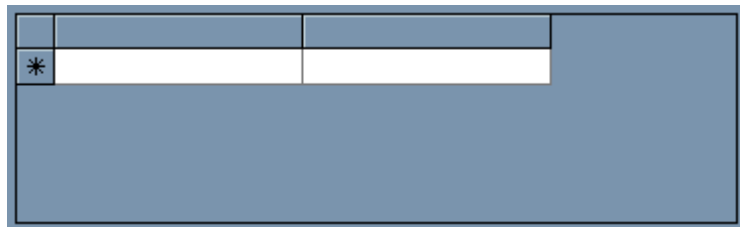
So far, I've created everything I need to browse the entire Customers table. But what about the orders?



First, I added another Data control to the project. The main control above will handle the customers, whilst this one will retrieve the orders.



Next, I added the DBGrid control to my Toolbox by clicking Project, Components and selecting 'Microsoft Data Bound Grid Control 5.0'.



Then I added an instance of the control to my form, as above - changing its DataSource property to the name of my second Data control.

Now the design is over and it's time to code, geek-to-geek. Here's the code you'll find underneath each command button:

ADD NEW

```
datCustomers.Recordset.AddNew
'Go on, add a new customer!
'Just fill in the customer fields
'and move to another record. Then
'try using the Search button to
'find the customer again!
```

DELETE

Due to the way Microsoft have setup the Nwind.mdb database, you can't delete customers that have orders. If you do, you'll receive an error. You must first delete all the orders before deleting a customer. This type of database integrity is maintained using "relationships", which we'll cover later in this tutorial.

On Error Resume Next

```
datCustomers.Recordset.Delete
'Delete the record

datCustomers.Recordset.Requery
'Then 'requery' the Recordset -
'which will 'refresh' it.
'It should show everything it did
'before, minus the deleted record
```

< BACK

```
datCustomers.Recordset.MovePrevious
```

FORWARD >

```
datCustomers.Recordset.MoveNext
```

SEARCH

```
Dim CompName as String

CompName = _
InputBox("Which company would you like to search for?")
'Get name from user, via InputBox

datCustomers.Recordsource = "select * from Customers " & _
"where CompanyName = '" & CompName & "'"
'Merge that name with an SQL string

datCustomers.Refresh
'Refresh the customer Data control
```

You'll notice that for the majority of buttons, we're simply using a method of the Data control's **Recordset** object, such as datCustomers.Recordset.Delete

The Recordset object in datCustomers is the bit that actually holds all the records. As its name implies, a Recordset is simply a "set of records". So when you want to delete a record, you simply fire that method of the Recordset.

Top Tip: You know how you can declare and work with strings in Visual Basic? Well, you can also declare and work direct with Recordset objects in code, something we'll be covering later in this tutorial

Of course, you can always implement error handling and all that jazz into your command button code, but I haven't bothered in other to keep things simple. And because I'm a complete sloth.

Now things may seem a little sticky when it comes down to the Search button code, but it's really pretty straight-forward. The code is doing this:

- Grabs the company name from the user (InputBox stuff)
- Changes the RecordSource property to an SQL string, with the company name inserted in the middle
- Refreshes the Data control to display the new bunch of records in the Recordset

There, that wasn't too bad, was it?

ORDERS SECTION

Well that's all fine and dandy, but we've still not thought about the customer orders section at the bottom of our form. In English, we want it to display both the order date and the actual shipping date. But how do we do that in code?

The Reposition event of the Data control fires every time the user moves to another record. And that's just what we want - to update the Orders section every time we move to another Customer. So, our code looks like this:

```
Private Sub datCustomers_Reposition()  
  
    datOrders.RecordSource = "select OrderDate, " & _  
        "RequiredDate from Orders where " & _  
        "CustomerID = '" & _  
  
    datCustomers.Recordset.Fields("CustomerID") & "'"  
  
    datOrders.Refresh  
  
End Sub
```

This code fires every time our first Data control - datCustomers, the one that holds customer information - is "repositioned". In the code, we're changing the RecordSource property of our second Data control to retrieve the OrderDate and RequiredDate fields for all Orders that have a CustomerID same as the one in the datCustomers.

The code - *datCustomers.Recordset.Fields("CustomerID")* - allows us to retrieve a singular field within our Recordset. In this case, we're asking it to give us the CustomerID field.

So, in brief, every time the user moves from one record to the next, the Reposition event occurs. You respond to this by refreshing the Orders Section and filling it with all OrderDate and RequireDate fields that have the same CustomerID as the current Customer record.

Whew!

CONCLUSION

This week we entered real nerd mode. We learned a few neat tips for generating SQL statements using Microsoft Access, figured out what queries are all about - and even built our own customer order browsing application.

Next week, we'll be adding greater functionality to that application - with database theory and query joins all on the menu. But I promise it won't get geeky nor boring. Really.

So join us next week for the fourth instalment of this Visual Basic Database tutorial. Until then, this is your surprisingly handsome host, Karl Moore, saying goodnight for tonight. Goodnight!

PART FOUR

INTRODUCTION

Good morning and welcome to yet another fantabulous instalment in this Visual Basic Database Tutorial!

I'm your surprisingly groovy host Karl Moore, and if you've missed the previous three parts, where've you been? Check them out:

- Part One [\[links removed\]](#)
- Part Two
- Part Three
- Part Four (**Ed: That's THIS one, Karl!**)

Don't forget that I'd love to hear what you think of this series - if you consider it terrible, tell us. I won't be offended. Much. Just use the comments section.

"So what Visual Basic delights will we encounter this week?", I hear the ecstatic surfers cry. We'll...

- Create a database application **completely** in code!
- Do some other stuff in code
- Figure out the BOF and EOF acronyms
- Then we'll do a bit more stuff in code
- And finally, we'll take a look at a few interesting database properties and methods

As you can see, it's all awfully exciting stuff. So come on chaps, let's crack that whip, scream "Tally Ho! Barf Barf!" and set sail for some real geeky VB'ing...

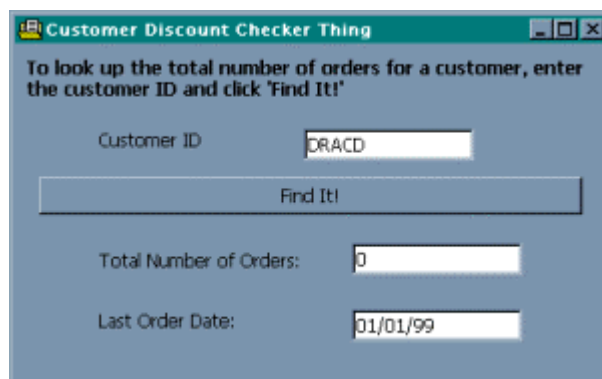
THE PROJECT

Right, fellow Visual Basic geeks - this is the section which separates the boys from the men, the girls from the women, and that dodgy unknown down the street from his even dodgier looking brother. Yes, we're about to access a database completely in code! Muhahahaha!!

[Ed: Mr Producer, are you *SURE* we can't cut his contract early?]

The annoying customer services department is on the phone. Again. They'd like a program into which they can tap a customer ID and have it display the total number of orders for that customer, along with the last order date. Why? To calculate the amount of discount to give, apparently.

So let's start by designing our wizzy Visual Basic form. Here's something I prepared a little earlier:



Quickly knock together the above in your copy of Visual Basic. I've not set any special properties, just added a few labels and text boxes.

I've named my three text boxes -

- **txtCustID** - the one that holds the customer ID
- **txtTotalNumber** - will display the total number of orders
- **txtLastDate** - will display the last order date

If you stick with these names, it'll make code writing much easier later in this tutorial.

So that's the design out of the way... now we need to think about code.

FANCY AN OBJECT?

Previously, we've accessed databases solely through the Data control. But this time we're doing it in code, using **objects**.

Just as you "Dim XYZ as String", we're going to do the same with a collection of special objects that allow us to play with our database. For example, you might write "Dim XYZ as Recordset" - which is an object that holds a set of records - then later say "XYZ.Delete", to delete one particular record.

I remember back in the golden olden days of my programming life, when I first saw database access code. **Argh!** I mean, who'd want to write all that code when you have the Data controls?

After a bit of use, it becomes clear that those Data controls aren't particularly cool. After all, you often want just one little bit of information from the database - and then you're finished with it. **Without** code, you'll have a clunky Data control constantly loitering on one of your forms. **With** code, you can simply tell Visual Basic, "access the database, get me that information, close the database"... the end.

In brief, accessing databases via code is amazingly flexible. Oh, and it's something else to brag about at the next Visual Basic group meeting.

DAEWOO?

Before we start coding with the database objects I mentioned earlier, we need to tell Visual Basic we're going to use them. To get those objects into our project, we need to set a "reference" to them - by clicking Project, References. You should be presented with a list longer than Bill Clinton's "To Do" pad.

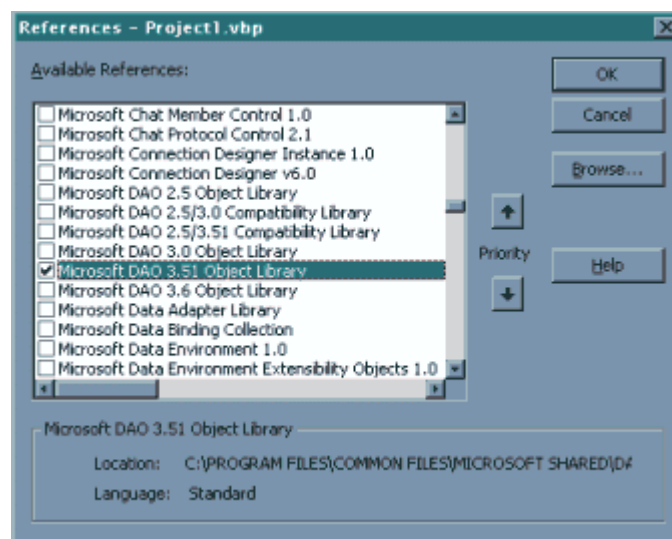
Scroll down to the ones starting 'Microsoft DAO'.

Top Tip: DAO stands for Data Access Objects, though it's considered better taste to use the confusing three-letter acronym instead

You'll notice quite a few there. DAO 2.5, 3.0, 3.51, 3.6 - Zzzzz.....

Every now and then, Microsoft likes to unveil new versions to access the latest database formats. For example, you should open Access 97 databases using DAO 3.51. And if you're wanting to plug into an Access 2000 database, you'll want to choose DAO 3.6.

We're going to access the Nwind.mdb database again, which is an Access 97 database. So click the checkbox next to 'Microsoft DAO 3.51 Object Library' and click OK.



Brilliant! In the next section, we'll get down to the real groovy stuff, something even more exciting than the 'Bumper Book of Italian Marriage Laws'... *actual database code!*

[Sarcastic Ed: Gee, I'm getting SO excited already...]

THE REAL CODE!

Hold onto your frilly knickers folks, this is where we dive straight in at the deep end, with a tonne of real-world code.

You should find the below code fairly easy to walk through, with comments every step of the way. Add this behind your 'Find It!' command button and give it a test run!

```
Private Sub cmdFind_Click()  
  
    Dim db As Database  
    'This is the object that will hold the connection  
    'to our database  
  
    Dim rs As Recordset  
    'This is the object that will hold a set of  
    'records coming back from the database  
  
    Dim SQLString As String  
    'This is just to temporarily hold the SQL string  
  
    Set db = OpenDatabase("c:\Microsoft Visual Studio\Nwind.mdb")  
    'This activates the database object, telling it  
    'to link to the Nwind.mdb database. Note that  
    'you may have to change this path depending on  
    'where Visual Basic has been installed on your PC.  
  
    SQLString = "SELECT Orders.CustomerID, " & _  
        "Count(Orders.OrderID) " & _  
        "AS NoOfOrders From Orders GROUP " & _  
        "BY Orders.CustomerID " & _  
        "HAVING (((Orders.CustomerID)='" & _  
        txtCustID.Text & "'))"  
    'This SQL statement was created in Access. It simply returns  
    'the number of orders for a particular customer using the  
    ''Count' feature on the 'Total' line. If you'd like to use  
    'Count, but are a little unsure about it - search Access help -  
    'it's very simple!  
  
    Set rs = db.OpenRecordset(SQLString)  
    'This ties the recordset object with the database object.  
    'You're telling it to set the recordset object to whatever  
    'the "db.OpenRecordset" function returns. And that function  
    'will return a set of records according to the SQL statement  
    'you pass it.  
  
    txtTotalNumber.Text = rs.Fields("NoOfOrders")  
    'Simply throws the value in the 'NoOfOrders' field  
    'from the Recordset, direct into the txtTotalNumber  
    'text box
```

```

SQLString = "SELECT Orders.CustomerID, " & _
            "Last(Orders.OrderDate) " & _
            "AS LastOrderDate From Orders GROUP " & _
            "BY Orders.CustomerID " & _
            "HAVING (((Orders.CustomerID)='" & _
            txtCustID.Text & "'))"
'We've already figured out the number of orders - so
'this is the SQL statement that finds out the last order
'date

Set rs = db.OpenRecordset(SQLString)
'This is the second time we've seen this statement. Here,
'it says the Recordset object to hold the records
'from our new SQLString statement

txtLastDate.Text = rs.Fields("LastOrderDate")
'Here, we're taking the information from the 'LastOrderDate'
'field and placing it in the txtLastDate text box

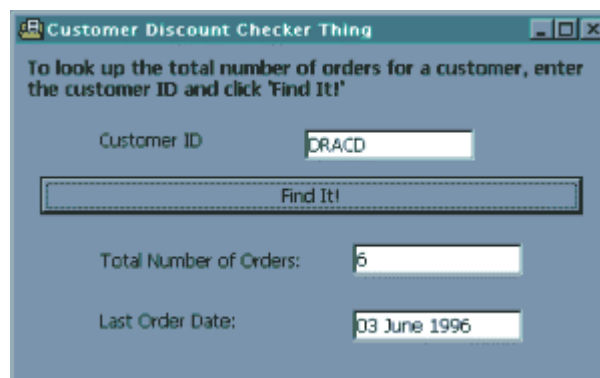
txtLastDate.Text = Format(txtLastDate.Text, "Long Date")
'Now we're just formatting to make it look pretty

rs.Close
'Close the Recordset

db.Close
'Close the Database

End Sub

```



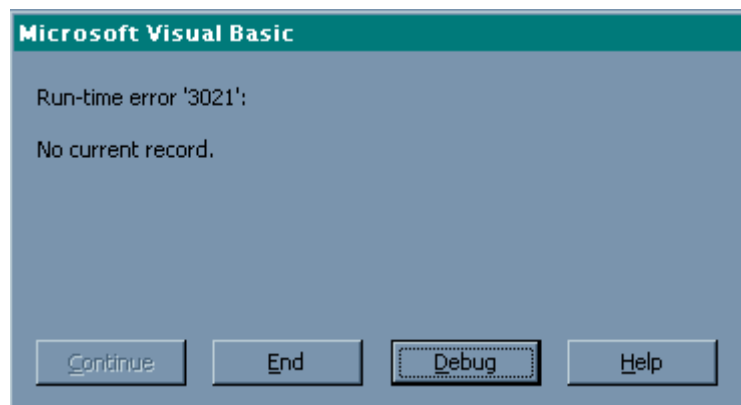
Wow, groovy or what! Try looking up the details for QUEDE, GALED, MAGAA or perhaps even ROMEY.

ARRGGGHHHH!

So your end users are merrily tapping away at the call centre, easily pulling up customer details with your supercool program. Suddenly, a terrifying scream echoes through the corridors. Move over Poirot, this ain't no murder... it's much more serious.

Enter stage - **RUNTIME ERROR!**

Try entering an incorrect customer ID into your program, such as KARLOS, and hitting the Find button. You should get this awful message:

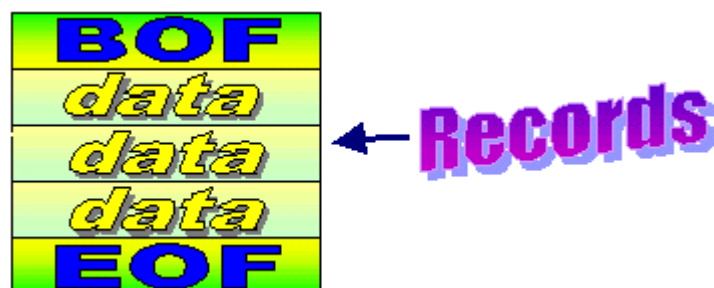


Hit the End button and let's discuss this like the adults we're not.

Runtime errors occur when you've told Visual Basic to do something it really can't handle. In this case, it's screaming: "No current record". But why?

In Visual Basic, each set of records has two special markers in it - known as the Beginning Of File (BOF) and the End Of File (EOF).

The BOF sits just before the first record and the EOF, just after the last record. In graphical terms, it looks a little like this:

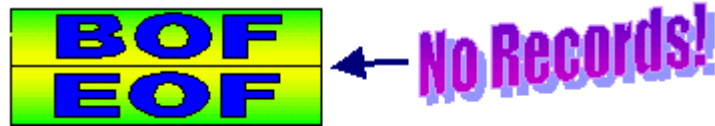


OK, so I'm no Picasso - but you get the idea. And at least **I've** still got both my arms.

[Ed: Ears, Karl. Picasso cut off his EAR!]

If you remember, we set the Recordset object to hold the results of our SQL statement. Most of the time, this will contain details such as the number of orders. But when the customer ID can't be found in the orders table - meaning the customer is either non-existent or hasn't yet placed an order - an empty set of records is returned to the Recordset object.

In graphical terms, that empty set of records looks like this:



So when we say `txtTotalNumber.Text = rs.Fields("NoOfOrders")` - Visual Basic replies with "Whoah, hold on a minute. There's no current record for me to grab your 'NoOfOrders' field from!"

We can check to see whether there is a current record by examining the BOF and EOF properties of the Recordset. When you're sitting on an actual record, both the BOF and EOF properties are **false**. If we are at the beginning **and** end of a Recordset after we execute an SQL string (BOF=True **and** EOF=True), we know it didn't return any records.

So let's edit our Visual Basic code to reflect that. Add the bolded code behind your command button:

```
Private Sub cmdFind_Click()  
  
    Dim db As Database  
    'This is the object that will hold the connection  
    'to our database  
  
    Dim rs As Recordset  
    'This is the object that will hold a set of  
    'records coming back from the database  
  
    Dim strSQL As String  
    'This is just to temporarily hold the SQL string  
  
    Set db = OpenDatabase("c:\Microsoft Visual Studio\Nwind.mdb")  
    'This activates the database object, telling it  
    'to link to the Nwind.mdb database. Note that  
    'you may have to change this path depending on  
    'where Visual Basic has been installed on your PC.  
  
    strSQL = "SELECT Orders.CustomerID, " & _  
        "Count(Orders.OrderID) " & _  
        "AS NoOfOrders From Orders " & _  
        "GROUP BY Orders.CustomerID " & _  
        "HAVING (((Orders.CustomerID)='" & _  
        txtCustID.Text & "'))"  
    'This SQL statement was created in Access. It simply returns  
    'the number of orders for a particular customer using the  
    ''Count' feature on the 'Total' line. If you'd like to use  
    'Count, but are a little unsure about it - search Access help -  
    'it's very simple!  
  
    Set rs = db.OpenRecordset(strSQL)  
    'This ties the recordset object with the database object.  
    'You're telling it to set the recordset object to whatever  
    'the "db.OpenRecordset" function returns. And that function  
    'will return a set of records according to the SQL statement  
    'you pass it.  
  
    If rs.BOF = True And rs.EOF = True Then  
        'Obviously the customer cannot be found in the  
        'orders table, so let's tell the user - and close  
        'the recordset/database connections  
        MsgBox ("Cannot find customer - " & _  
            "txtCustID.Text & " - " & _  
            "in the Orders table!")  
        rs.Close  
        db.Close  
        Exit Sub  
    End If  
  
    txtTotalNumber.Text = rs.Fields("NoOfOrders")  
    'Simply throws the value in the 'NoOfOrders' field  
    'from the Recordset, direct into the txtTotalNumber  
    'text box
```

```

SQLString = "SELECT Orders.CustomerID, " & _
            "Last(Orders.OrderDate) " & _
            "AS LastOrderDate From Orders " & _
            "GROUP BY Orders.CustomerID " & _
            "HAVING (((Orders.CustomerID)='" & _
            txtCustID.Text & "'))"
'We've already figured out the number of orders - so
'this is the SQL statement that finds out the last order
'date

Set rs = db.OpenRecordset(SQLString)
'This is the second time we've seen this statement. Here,
'it says the Recordset object to hold the records
'from our new SQLString statement

txtLastDate.Text = rs.Fields("LastOrderDate")
'Here, we're taking the information from the 'LastOrderDate'
'field and placing it in the txtLastDate text box

txtLastDate.Text = Format(txtLastDate.Text, "Long Date")
'Now we're just formatting to make it look pretty

rs.Close
'Close the Recordset

db.Close
'Close the Database

End Sub

```

TOP PROPS

When it comes down to databases, you can do virtually everything in code. Here are a few of the top properties and methods of the Recordset object for you to play around with:

- **MoveNext** - Moves to the next record in the Recordset
- **MovePrevious** - Moves to the previous record
- **MoveFirst** - Moves to the first record
- **MoveLast** - Moves to the last record
- **Edit** - Enters edit mode, for changing fields
- **Update** - Saves any edits
- **AddNew** - Adds a new, empty record
- **Delete** - Deletes the current record
- **RecordCount** - (Number) Returns the number of records currently accessed (move to the last record to get total number of records)
- **BOF** - (Boolean) Hit when you step before the first record in a Recordset
- **EOF** - (Boolean) Hit when you pass the last record in a Recordset

Indeed, using these simple properties and methods you could very simply replicate all the functionality of the Data control.

Go on, have a go! Try to build a simple Customer browsing application, similar to the one we created in the third tutorial.

CONCLUSION

This week, we've covered loads of stuff. We built a mini-application completely in code, added some BOF/EOF checking and got wise to the numerous helpful properties and methods of the Recordset object.

Next week, we'll be covering... erm, other stuff. I'll be droning on about database theory and the correct way to do, erm, more stuff in databases. Then I'll talk about joins and stuff, followed by even more database stuff. Ahh, at least that's clear then.

But until next week, this is your host, Karl Moore signing off for tonight. Goodnight!

PART FIVE

INTRODUCTION

Welcome to the fifth instalment in this Visual Basic Database tutorial, exclusively hosted on VB-World.

As ever, I'm your amazingly cool host Karl Moore. In fact, as we're getting to know each other a little more each week, I'll let you call me Karlos. May I call you "reader"?

Don't forget that I love to hear your feedback. If you're enjoying the tutorial, then I want to know! My e-mail address is karl@karlmoore.com

And if you're not enjoying the tutorial, the Editor wants to know. Hah, like I'm going to give you **his** e-mail address?

[Ed: It's webmaster@vb-world.net]

[Karlos: DOH!]

Just don't tell him you don't like my humour. Otherwise I'll be out of a job, with three iguanas and a mangy looking cow in its late 40s to support.

Anyway, enough pitter-patter. On with the content! This week we'll be taking a look at:

- How best to design a database
- Creating a table
- Creating a chair
- Primary keys
- Foreign keys
- House keys
- Relationships (Dr Ruth, anyone?)

So onwards, good sir... let's get down and dirty on the disco floor with the database diva... (that's apparently me, by the way).

THE GOOD, THE BAD AND THE UGLY

Let's face it. There are three types of people in this life; the good, the bad and the ugly.

You and I fall into the first bracket; the stunningly good looking and wonderfully intelligent group. The Editor falls into the second band. And the average database design falls into the last, rather disgusting category.

This week, I'd like to give your database design a makeover – and move it straight into the upper echelons of the "Good".

So lets start with a case study. Sit back and imagine we're running a pet hospital. You need to record all the information about me. And my pet iguanas. And my pet cow.

Based on the information I've given you already, you'd probably store my name and address all on one row. And on that same line, you'd add details of my pet. Your basic table might look like this:

ID	Name	Address	Pet	Breed
121	Karl Moore	The Infirmary	Wiggles	Iguana

But what about my other pet iguana, Green Thing?

Hmm, I guess you could change your table to look like this:

ID	Name	Address	Pet	Breed	Pet	Breed
121	Karl Moore	The Infirmary	Wiggles	Iguana	Green Thing	Iguana

That's brilliant. Well, in a kind of non-brilliant way.

You see, you've forgotten my third iguana, Strangely Brown. Oh dear.

So how do you get around this? Sure, you could add a third column. But then you've forgotten about my pet cow, Daisy. Doh!

THE RIGHT WAY

"So what should you do?", I hear you cry. Enter stage – *a secondary table*.

Let me explain; imagine having one table holding all my basic information like this:

ID	Name	Address
121	Karl Moore	The Infirmary

And then a secondary table holding information about each of my pets – with the details of each animal on a new line – like this:

ID	Pet	Breed
121	Wiggles	Iguana
121	Green Thing	Iguana
121	Strangely Brown	Iguana
121	Daisy	Cow

That's great, but what links Wiggles the iguana to its owner, me, Karl Moore?

You probably noticed my customer ID number in the first table. And likewise, the second table also holds an ID number

So any pets in the second table with the ID number of 121 belong to the owner with an ID number of 121 also. Simple, eh?

Top Tip: No self-respecting geek calls those matching values, err, matching values. Instead the ID field in the main table is known as the *Primary Key* – and the ID field in the second table is known as the *Foreign Key*.

And a good database design really isn't awfully difficult to program around. Let's say you allow your user to browse through the main list of owners – and have a list of pet names for that owner displayed in a DataGrid at the bottom of your screen.

You'd simply tell your program, in geeky programming terms: "Every time the user moves a record in the main customer table, display all the details from the pets table – where the ID number in the pets table is the same as the current customer's ID number".

OK, so it might sound a teensy-weeeensy bit complicated. But it ain't. Really.

At least, not for mega supercool geeks like you and I...

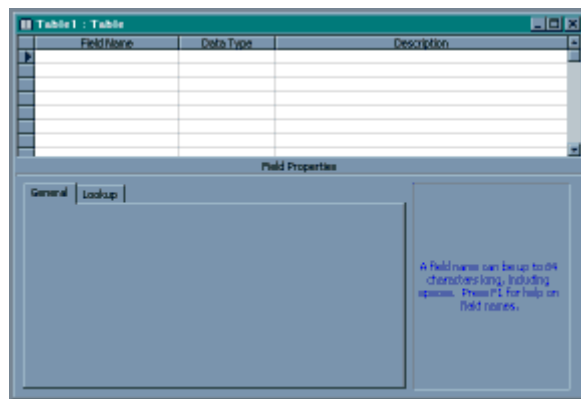
DESIGN THAT TABLE!

So, in summary, it's good practice to throw anything you may need to store multiple values for, in a separate table. You can then link that information back to its 'owner' via a 'key', which is typically some form of ID number.

So let's put this theory into practice. Start Microsoft Access and create a new database – say, c:\surgery.mdb

Click on the Tables tab, hit the New button, select Design View and click OK.


You should be presented with the table design window, which allows you to define the fields you want in your table. It should look a little like this at the moment:



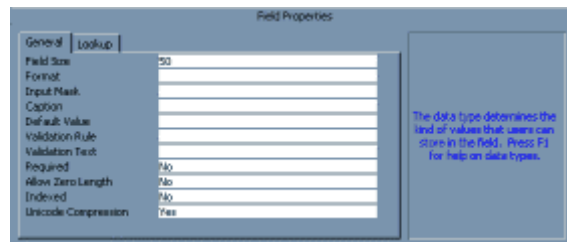
Your cursor should be flashing in the first 'Field Name' box at the moment. We'll call this field 'OwnerID' – so type it in and press Tab.

You should now be in the 'Data Type' box. This is where you tell Access what type of information you want the field to hold. For example, you could tell it to hold normal Text, or perhaps a Date, or perhaps a Number, or perhaps... an AutoNumber.

Select the latter. An AutoNumber is an automatically generated number that will be inserted into the field each time a new record is entered into the table. And it's completely unique, too – which is just what we want for the OwnerID field.

Now this field will be the 'Primary Key' – in other words, that main number which uniquely identifies an owner. And to make sure Access knows this, click on the 'Primary Key' button  on your toolbar. You should see a small key appear beside the Field Name.

Click down into the next 'Field Name' box and enter 'OwnerName'. Tab across and select the 'Text' data type. The bottom half of your screen should look like this:



This lists all the properties of that one field. You might, for example, change the 'Default Value' property to 'John' – meaning that 'John' will be automatically inserted into that field as a default name every time you add a record.

Or you may change the 'Field Size'. At the moment, our field will only hold up to 50 characters. But we're going to change that to allow for owners with particularly long names – go on, up the value to 100 characters!

Also, change the 'Required' property to Yes. That means a user cannot add a new record to the table without this field – OwnerName – being completed.

Add another Field Name and call it 'Address'. Again, change the Field Size to 100 and the Required property to Yes.

Your screen should look like this right now:




Click File and Save. Enter the name "Owners" and click OK. Now close the table and follow the exact same process to create a table with the following specifications:

- Table Name - Pets
- 1st Field Name - OwnerID
- 1st Field Data Type - Number
- 2nd Field Name - PetName
- 2nd Field Data Type - Text
- 3rd Field Name - Breed
- 3rd Field Data Type - Text

When saving the table, you may get prompted to add a Primary Key – just click No. This is the secondary table, which contains the **Foreign** Key (OrderID) we mentioned earlier.

We've almost completed the database design; we've created two tables, with an AutoNumber in one and a number field for that AutoNumber in the second. All that remains is for us to tell Access that there is a relationship between the first Owners table and the secondary Pets table.

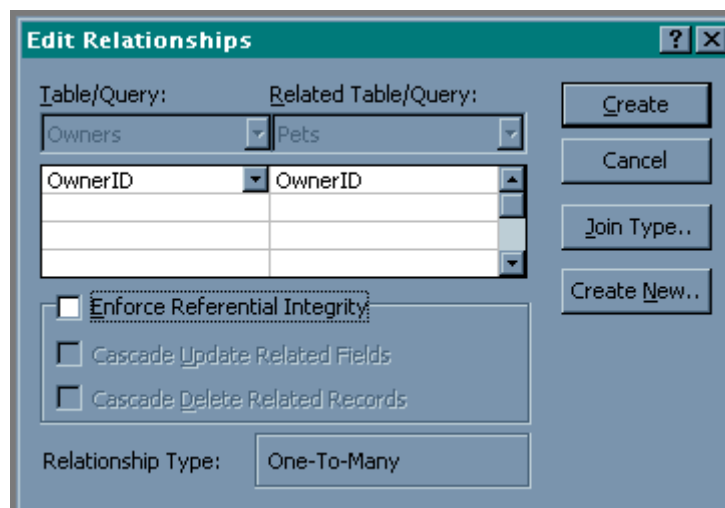
Return to the main Database window and click the Relationships button  on the toolbar.

You'll be prompted with a box asking which tables you wish to show in your Relationships diagram. Double-click on both Owners and Pets, then click Close. Your screen should look something like this:



At this point, we want to tell Access there is a relationship between the Owners table and Pets table. Drag the OwnerID field in the Owners table over to the OwnerID field in the Pets table and let go of the mouse button.

You should be prompted with the following:



Notice that Access has determined the relationship type as being 'One-To-Many' – meaning there will be one occurrence of the OwnerID number in the first Owners table and possibly numerous instances of that number in the Pets table.

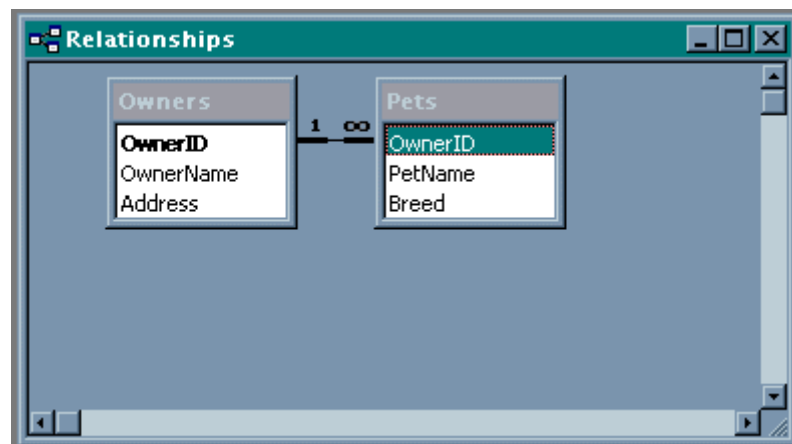
And that's right – one owner can have multiple pets. Just as one company department may have many employees. Just as each customer order may have numerous individual order items.

So this One-To-Many relationship isn't **just** useful in the veterinary world.

Check the 'Enforce Referential Integrity' button; this will ensure your data stays in tip-top condition. In other words, your users won't be allowed to enter a value in the OwnerID field of the Pets table that doesn't exist in the Owners table. After all, you can't really associate a pet with an owner that doesn't exist!

All the referential integrity thing does is **enforce** that rule.

Click OK. Your screen should look something like this:



Click on File, Save to store the relationship.

And that's it... you've completed your database design!

Now if you wanted to be a real boffin, you could probably put all the breeds in a separate table and instead of inserting a text value in the Breed field – insert a number, a Foreign Key linking it back to a Breed description in another table. Don't forget that storing a number is much more efficient than storing a piece of text. And it's certainly less prone to typos.

[Ed: Err... Typos]

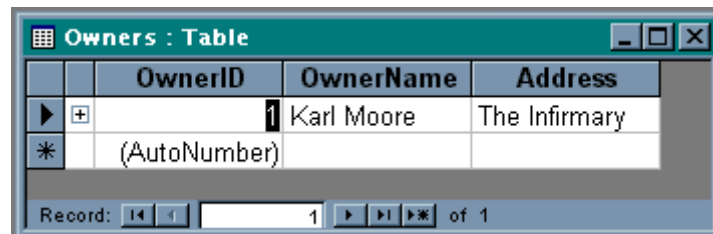
But we'll leave that till another day. For now, give yourself a bally hard pat on the back... you've completed the database design! And it's absolutely mega-cool! Hah, move over Bill Gates...

LET'S TEST IT!

Don't know about you, but most of my programming never works first time. In fact, most of it just never works. But I've a good feeling about this database design... and it's not just 'cause I spent hours testing it.

So let's put our database design through the works. First off, let's enter a new owner into the Owners table.

Open it up and throw in a little sample information. Here's mine:

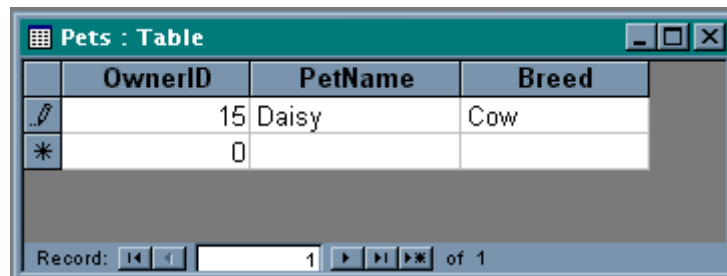


OwnerID	OwnerName	Address
1	Karl Moore	The Infirmary

Note that I've been automatically assigned a unique OwnerID of 1.

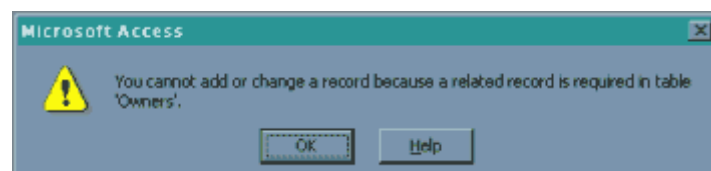
BRILLIANT! That's our Primary Key stepping in.

Now open up our Pets table and enter a little more sample information. But this time, enter number 15 in the OwnerID field. In other words, you're saying this pet belongs to the non-existent owner number 15.



OwnerID	PetName	Breed
15	Daisy	Cow

When you try to move off the record, you should receive the following groan:



Or in other words, "You can't add a pet without it having an owner, you crazy crazy crazy person!"

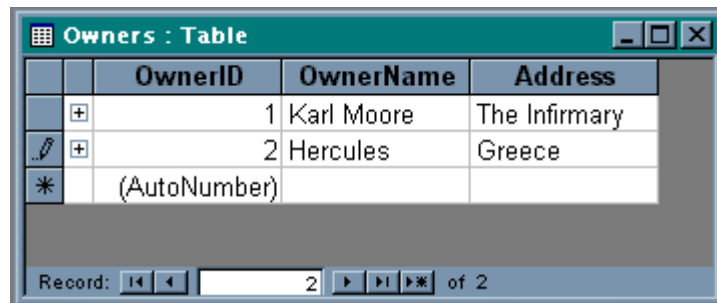
Try changing the OwnerID to 1 and adding the pet once more.

Does it work? **Bravo!!**

Try adding all my pets to your list; the three iguanas – Wiggles, Green Thing and Strangely Brown, and that 48 year old cow, Daisy.

It's worth noting that you won't be able to put anything but a number in the OwnerID field. When we designed the table, we choose the 'Number' data type – which protects it from strange dates, bits of horrid text or weird boolean values. Groovy!

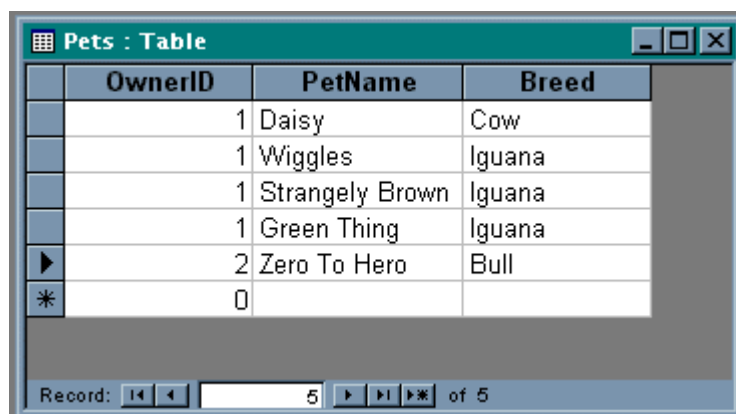
OK... go back to the Owners table and try to add another record. You'll notice the OwnerID number is automatically incremented...



	OwnerID	OwnerName	Address
+	1	Karl Moore	The Infirmary
+	2	Hercules	Greece
*	(AutoNumber)		

Record: 2 of 2

... and you should now be allowed to add entries to the Pets table using that newly-generated OwnerID number.



	OwnerID	PetName	Breed
	1	Daisy	Cow
	1	Wiggles	Iguana
	1	Strangely Brown	Iguana
	1	Green Thing	Iguana
▶	2	Zero To Hero	Bull
*	0		

Record: 5 of 5

Hurrah! Go on... shout **hurrah!!**

CONCLUSION

Wow, we sure covered a lot of ground today!

I babbled about good database design for a while, and then we delved into Access to create our very own tables – complete with keys and relationships. Finally, we tested our supercool design... and it worked!

Next week we'll be taking this project further. I'll show you how to easily build your Visual Basic application around a good database design... using a few little-known control properties.

Tune in at the same time next week for more of the same. Until then, I'm your host Karl Moore saying goodnight for tonight... goodnight!

PART SIX

INTRODUCTION

Guten Morgen and welcome to yet another instalment in this wizzy Visual Basic Database tutorial.

As ever, I'm your stunningly geeky host Karl Moore – and if you've missed any previous slots, check out:

- Part One [\[links removed\]](#)
- Part Two
- Part Three
- Part Four
- Part Five

This week, we'll be taking a sneak geek peek at how you can program your Visual Basic application around last week's groovy database design. We'll then take a little look at validation before moving on to create a simple VB-integrated report.

But be warned, those with older versions of Visual Basic may feel about as comfortable as the passengers of infamous Flight 402, with the plane designed by La Augustus of Tower of Pisa fame, technically authenticated by Crazy Joe of Slackers Associated, and piloted by Kamikaze brothers Tun Twick Tang and Tun Twick Tang Junior.

Unfortunately we're dealing with all the latest and greatest features, which only work with VB 6.0 – yeah, I know, I'm sorry. Please, stop blubbing.

Oh, and don't forget to tell us how we're doing. If you love the series, tell us using the comments section. If you hate the series, tell us using the feedback form below. If you're after a date, e-mail me - karl@karlmoore.com ;-))

Anyway, let's get on with the show, kiddo...

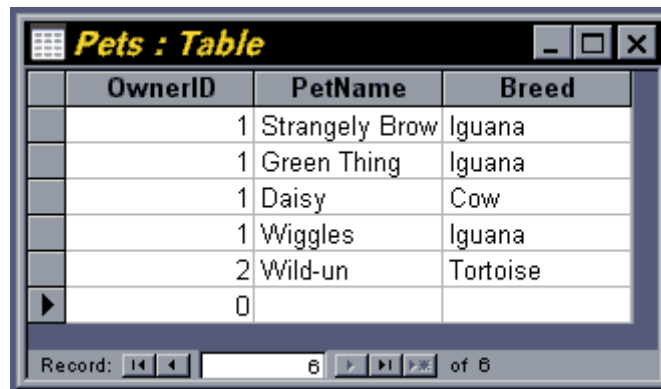
BINDING AND ALL THAT

Binding, binding, binding. Sounds a bit boring doesn't it?

I'd much prefer the word "bonding". Even better, "bondage". But unfortunately that just wouldn't go. Sure, we all know this is a family website, but the Editor's on holiday in France this week so why not? ...

Do you remember all the bondage we encountered back in the distant realms of tutorial one? How we tied that text box direct to the database! We just added a data control, threw a few text boxes on a form, set a couple o' properties... and hey presto, we're up-and-running.

But with the wizzy database design you created last week, that just isn't possible anymore. For instance, if you wanted your users to enter new pets into the Pets table, your database requires a PetName, Breed... and an OwnerID.



OwnerID	PetName	Breed
1	Strangely Brow	Iguana
1	Green Thing	Iguana
1	Daisy	Cow
1	Wiggles	Iguana
2	Wild-un	Tortoise
0		

If you remember, the OwnerID is a Foreign Key. In other words, it's a number that links this particular record to a parent record in the Owners table.

But asking your end user to input an OwnerID isn't very user friendly. And when you start creating databases that are absolutely jam-packed full of Foreign Keys, it'll become more confusing then the Arabic translation of Stephen Hawking's "Bumper Book of the Galaxy".

So you need to make it simple, by allowing them to select an owner from a list... but you should still insert the OwnerID number into the field.

How do you do this? By use of a few little-known control properties...

DAO OR ADO?

In previous tutorials, we've connected to databases using the bog standard Data control.



Now this seemingly innocent control accesses your Access database via a technology known as DAO, or Data Access Objects.

Since those halcyon days of prehistoric data access, Microsoft has released another two ways of getting at databases – Remote Data Objects (RDO) and ActiveX Data Objects (ADO).

Remember we programmed in database code a few instalments back? 'Dim db as Database' and all that? Well, we were using Data Access Objects to do that.

But RDO and ADO offer us new ways of accessing such information. How come? Well, we'll skip over RDO for the moment. But ADO supposedly allows us to access all sorts of information, above and beyond your typical Access database.

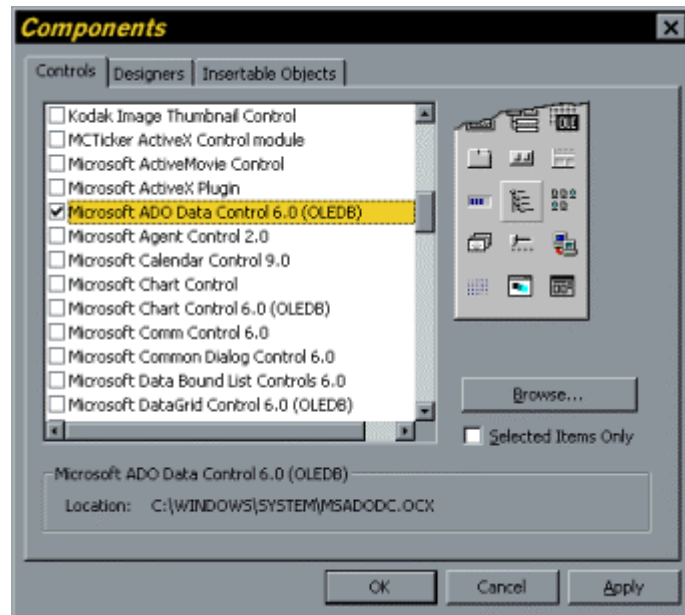
In essence, you should be able to simply connect with any data source – from SQL Server to Access to Oracle to an e-mail application to old Mainframe information – all using the same wad of ADO code.


And that code is a *little* different to the stuff we wrote earlier. Not too much, but there are unique distinctions.

Why am I telling you all this? Because those special control properties I was talking about on the previous page – the ones that'll help you design a great VB application around an ever better database design - are only available when you access databases through ADO.

So first off, start a new project and let's add the ADO equivalent of the Data control to our project:

- Select Project, Components
- Scroll down the list and check 'Microsoft ADO Data Control 6.0'
- Click OK

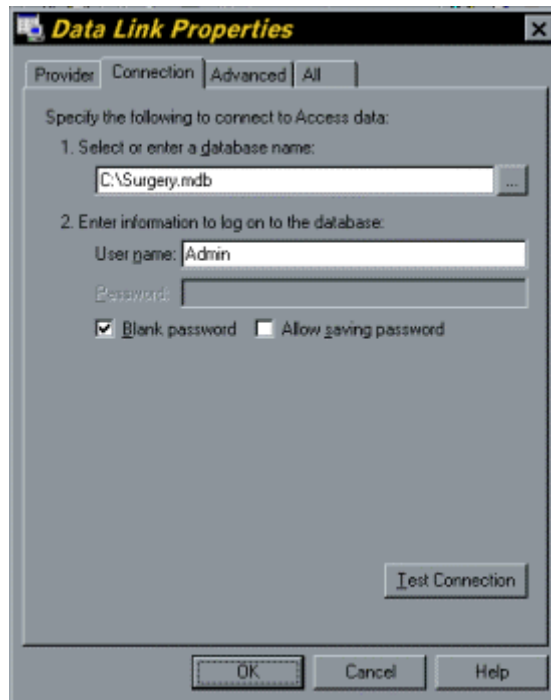


You'll notice a new control appear in your toolbox - 

- Double click on the ADODC control to add it to your form
- Change its Name property to 'datPets'

We now need to tell this control we want it to connect with the Veterinary database we created last week. This is done slightly differently using ADO – via 'connection strings':

- Click on the ellipsis beside the ConnectionString property
- Click the Build option
- On the screen that appears, select 'Microsoft Jet 3.51 OLE DB Provider' if you created the database in Access 97 – or 4.0 if you created it in Access 2000
- Click Next
- Type in the path to your database (eg, c:\surgery.mdb)
- Click OK
- Click OK on the Property Page form

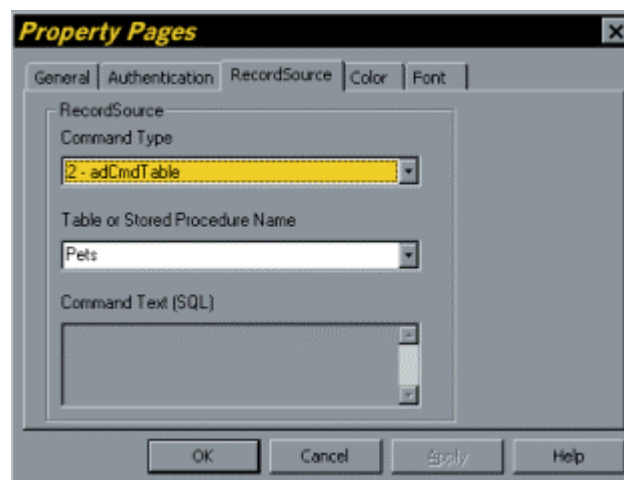


You will notice the builder inserted something similar to the following in the ConnectionString property:

- Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Surgery.mdb;Persist Security Info=False

This string tells ADO what type of database it is looking at and where it can be found. Now, as with the DAO control, we need to tell it what information to retrieve:

- Click the ellipsis next to the RecordSource property
- Select Command Type '2 - adCmdTable'
- Select the 'Pets' table from the available list
- Click OK



Now we're going to add another control to our program:

- Click Project, Components


- Scroll down, select 'Microsoft DataList Controls 6.0'
- Click OK

You should see two extra widgets appear in your toolbox -



Add the one titled 'DataCombo' to your project and change its Name property to 'dbcOwner' and its Style to '2 - dbcDropDownList'. This is the special control that will allow the user to select an owner name, but still put the OwnerID number into the database.

But before we can use that DataCombo control, we need to add another ADODC control.

- Add another  control to your project
- Change the Name property to 'datOwners'
- Set the Visible property to False
- Use the same ConnectionString in this control as you did the previous datPets
- Change its RecordSource to the Owners table

So that control is now setup to retrieve information direct from the Owners table. Now we plan to send those details to the DataCombo control – and tell it to display the owner name but when inserting into a database, use the OwnerID field.

OwnerID	OwnerName
1	Karl Moore
2	Hercule
(AutoNumber)	

We do this using a few supercool properties:

- **RowSource** – the data control from which the combo box gets its initial wad of information
- **ListField** – the field that will be listed in the combo box
- **BoundColumn** – the underlying value or 'Key' field (eg, OwnerID)

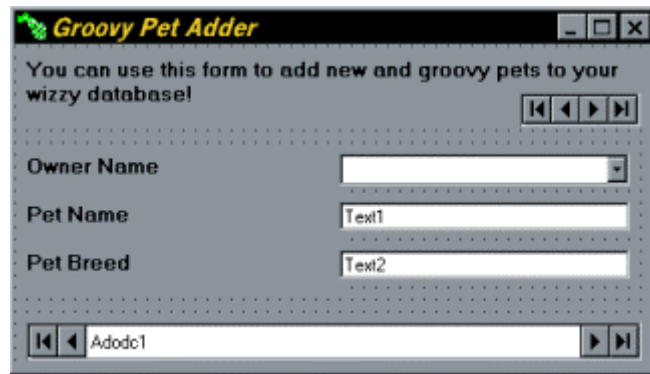
So let's change our Data Combo properties to:

- **RowSource** – datOwners
- **ListField** – OwnerName
- **BoundColumn** – OwnerID

Now add a couple of other text boxes to store the pet name and pet breed values.

Link all three of your controls up with the datPets ADODC control by changing the DataSource and DataField properties.

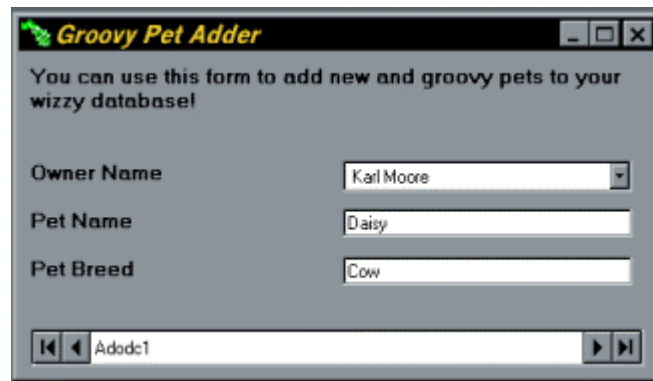
My completed application looks a little something like this:



If yours doesn't look as good as mine, well... it takes years of practice.

If it looks better than mine... well, I don't want to discuss it. I'll start getting grumpy and upset. I may even go for a sulk somewhere in the corner of cyberspace.

Anyway, hit F5 to run your application. Does it work?



Err, as this is a kinda one-way thing... I'll presume you answered with a resounding "yes!"

You should be able to scroll backwards and forwards through your Recordset using the ADODC control. You should also be able to change the owner of a pet very easily using the Data Combo box – whilst at the same time keeping physical numbers in the database, not text values.

So there we have it – a basic application built on a solid database design with absolutely no code.

Why not try creating another table to hold all the pet breeds and linking into those with another key? Go on, I dare you!

CONGRATULATIONS AND VALIDATIONS

Cliff Richard once had a number one hit that began, "Congratulations... (strum, strum) ... and Validations... (strum, strum)... lah lah lah dadada lah lah dada la lahhh!".

Actually, no, he didn't. In fact, those fictional lyrics weren't even remotely plausible. But nevertheless, it introduced the next section – validation – which is itself rather abstract.

You see, it doesn't really fit anywhere within my grand plan to boost your database knowledge. And that's why I'm going to include it here.

[Ed: I see... clear as mud]

Now, many of the common controls have a 'CausesValidation' property, which is set to True by default. This means before a user leaves the Text Box or Combo Box or whatever, the Validation event is fired.

And it's there you can check what they entered. For example, I placed the following code behind my txtBreed button from the project on the last page:

```
Private Sub txtBreed_Validate(Cancel As Boolean)
    If Trim(txtBreed.Text) = "Unknown" Then
        MsgBox "Can't accept that. Are you sure " & _
            "it's not an iguana?"
        Cancel = True
    End If
End Sub
```

If a user enters 'Unknown' into the Breed Text Box, my code tells the Text Box the value provided is unacceptable – by setting the passed Cancel value to True. And that means the focus remains on the Text Box in which the problem occurred.

If I performed no checking or set Cancel to False, the user would've been able to continue in their operations.

So the Validate event is neat for verifying information a user enters; whether you want to ensure they've typed a sensible number or you need to perform a little syntax checking, it's a pretty cool event to have around. In fact, it's cooler than Mr Cool the Cola Bear, winner of last year's I'm-Sooo-Cool competition.

Interesting, no?

[Ed: Hmm, no]

[Karl: <Tears>]

WRITING A REPORT

Some really tricky users don't just want to throw information *into* a system, they also want a little *output* in the form of reports. Sure, **we** know they're just being picky... but you'd better oblige in the faint hope of a pay rise sometime within the next millennia.

There are a few ways you can create reports via Visual Basic. The most popular are:

- Transfer query results direct to Excel for processing
- Produce reports in Access and print out occasionally
- Don't
- Use Crystal Reports (usually the cut-down version on VB CD)
- Utilise the built-in Data Report widget

In this section, we're going to check out the latter Data Report object. This allows you to create a report in Visual Basic very similar to those found in Microsoft Access – and it can be viewed, printed or exported direct through your application.

So let's have a stab at creating our first Data Report in the 'Groovy Pet Adder' program we created earlier:

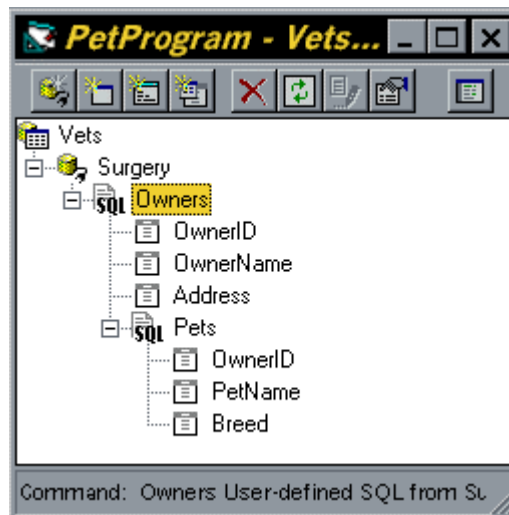
- Click Project, More ActiveX Designers, Data Environment

Every Data Report requires a source of information. In Visual Basic, this comes from a Data Environment.

Now stick a big hat on my head, call me Professor and dance around ten times chanting "He's Einstein, Shakespeare and Angela Lansbury rolled into one"... but I personally understand a Data Environment to simply be an environment. For data.

Anyway, let us continue fiddling with it:

- Click on the node titled 'DataEnvironment1'
- Change its Name property to 'Vets'
- Change the Name property of Connection1 to Surgery
- Now click on the 'Add Command' button at the top of your screen
- Change the Name property of Command1 to Owners



The Finished Product

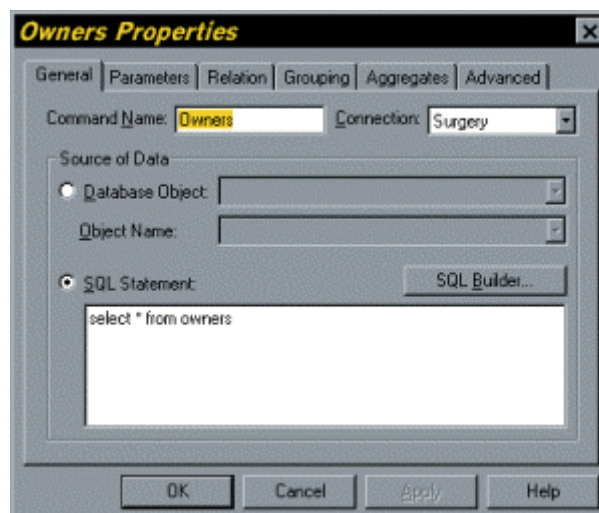
All this 'naming' ensures we keep track of the information these objects will hold.

- Right-click Surgery and select Properties
- Choose Jet 3.51 if your Surgery database was created in Access 97, or Jet 4.0 if you used Access 2000
- Hit Next
- Type in the database path and click OK

That's told the Surgery object all things below it (ie, Owners and Pets objects) will get their information from the Surgery database.

- Right-click Owners and select Properties
- Click the 'SQL Statement' radio button and enter the following SQL:

```
Select * from Owners
```



This statement simply tells the Data Environment's Owners object to retrieve all fields from the Owners table.

- Click OK
- Tap the little + next to the Owners object

You should see a list of fields your query returns – OwnerID, OwnerName and Address. But in this report, I want to display each owner followed by every pet name and breed owned by that character. So we need to tell our Data Environment about the Pets table:

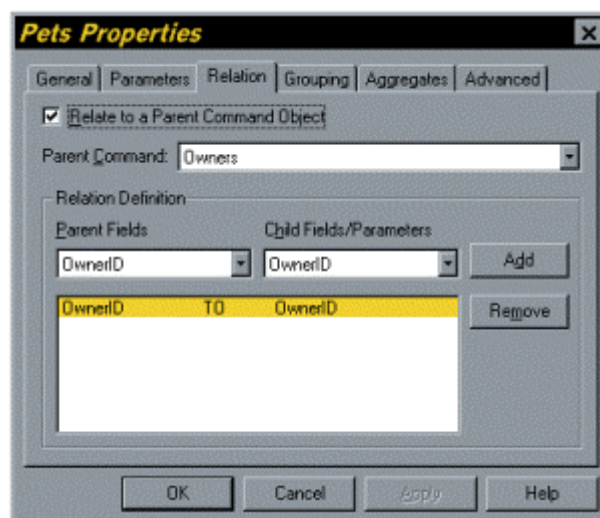
- Right-click Owners and select 'Add Child Command'
- Rename the command you just created, 'Pets'
- As with the previous command object, enter an SQL statement of:

```
Select * from Pets
```

- Click OK

So far we've told the Data Environment about the existence of both tables, but not how they're related to each other. So:

- Right-click Pets and select Properties
- Click the Relation tab – this is where you define the link between the parent object Owners and the child object Pets
- In Parent Fields combo box, select OwnerID and likewise in Child Fields. Then click the 'Add' button. This links the two lots of information together
- Click OK

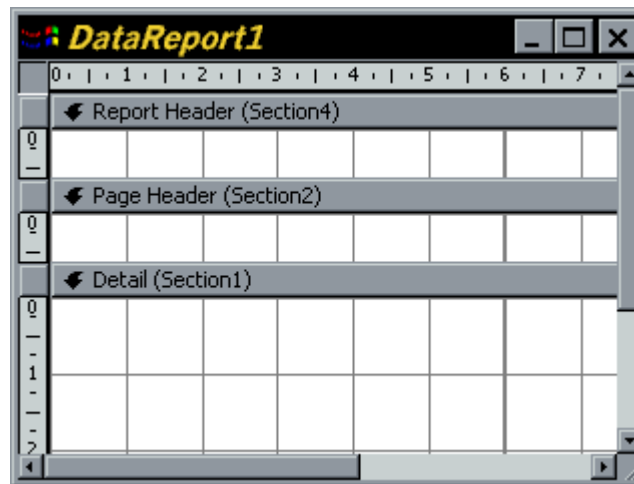


Top Tip: You can change the SQL statement behind your 'command' in code with - `YourDataEnvironmentName.Commands("Owner").CommandText = "select * from owners"`

This allows you to dynamically change the database information your report will be based on. So instead of selecting ALL the owners, you could perhaps pick out just one.

So we've sorted out the information we want to get from the database... now we just need to throw it into a report:

- Click Project, Add Data Report

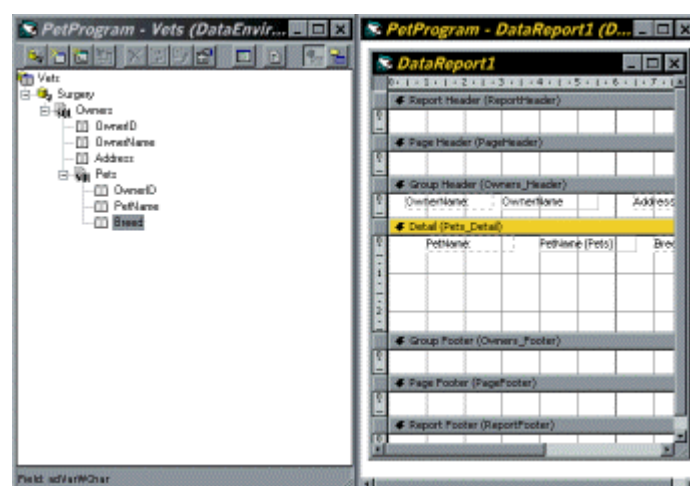


This is the report window in which you may design your report. But before telling it what to display, you need to tell it to link to your Surgery Data Environment:

- Click on the 'DataReport1' title to display relevant information in the Properties window
- Change the DataSource property to Vets and the DataMember to Owners
- Now right-click anywhere on the Data Report and select Retrieve Structure – clicking Yes when prompted

Now select Window, Tile Vertically. This should enable you to see both the Data Environment and Data Report screens at once.

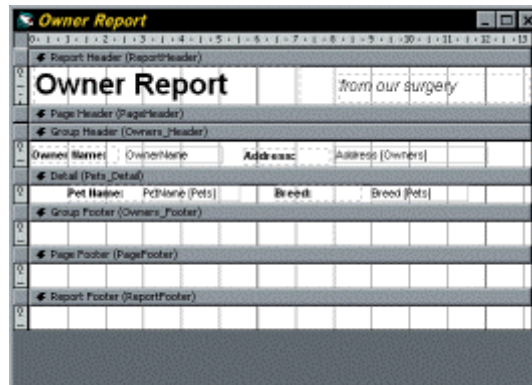
First, drag the OwnerName field from the DataEnvironment direct over to the 'Group Header (Owners_Header)' band. Do the same with the Address field.



Next, drag and drop the PetName and Breed fields over from the Data Environment into the 'Detail (Pets_Detail)' band.

Now try formatting the report. Perhaps you'd like to bold or rename a few of the field labels, shrink the Details section or add a title using the toolbox controls.

This is what my report looked like after a little fiddling:

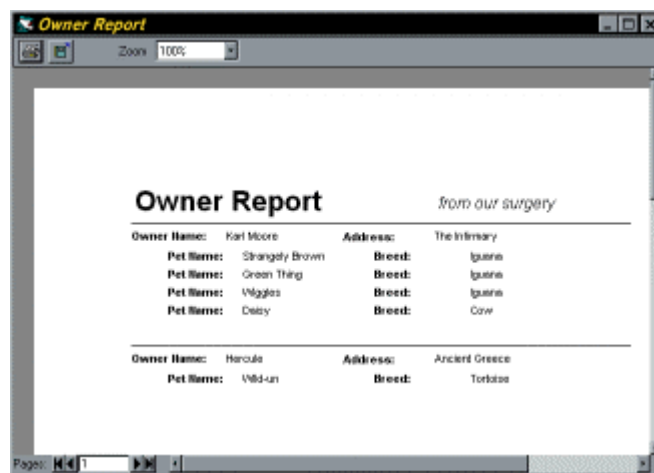


Return to the main 'Groovy Pet Adder' screen and add a command button with a caption of 'View List of Pets'. Behind that button, add the code:

- `MyReport.Show`

where 'MyReport' is, err, the name of your report.

Then run your program and hit the command button. Your report should look something like this:



Note the buttons in the top left-hand corner. Although you can remove these by setting a few properties, they allow your user to easily print the report –even export it to various file formats, including a HTML document

Cool, eh? In fact, it's cooler than Mr Cool the Cola Bear, winner of... oh darn, I've used that one.

For more information on creating a Data Report, search help for "Creating a Simple Data Report".

CONCLUSION

This week, we covered how to build your Visual Basic application around last week's supercool design. Then we dabbled with a little abstract validation before skipping on to create a simple report.

Next week, we'll be taking a peek at the built-in Visual Database Tools that ship with Visual Basic 6.0. And if you're using SQL Server or Oracle databases, you're in for a treat.

Until then, I'm your wizzy host, Karl Moore... saying goodnight for tonight. Goodnight!

PART SEVEN

INTRODUCTION

Good morning fellow geeks and geekesses! Karl Moore here as usual, attempting to glide you smoothly through the rocky mountains of Visual Basic and database design. Ahh, paints a pretty picture, don't it?

[Ed: Beautiful]

If you're unfortunate enough to have missed the previous six slots (DOH!), check them out here:

- Part One [\[links removed\]](#)
- Part Two
- Part Three
- Part Four
- Part Five
- Part Six

This is the seventh and final part of our Visual Basic Database tutorial. Today, we'll be covering:

- SQL Server, Oracle and all that jazz
- The Groovy VB Database Tools
- Where to go from here
- Recommended reading
- Maybe some other stuff I haven't thought of yet

Ready? Hold on tight...

THE ORACLE RETURNS

How much information does your typical database hold? If it's as empty as a hermit's address book, you should have no problems using Access MDB files in your projects.

But there comes a time in every programmer's life when hair starts to grow on the chest (yes, women included) and a database change is required.

That unwanted hair can crop up for many reasons. For example, Access has many limitations. For a start, it can only store up to 2Gig of information.

Another disadvantage; let's say your Access database is on a network and you send it a query. Do you know **all** the records from related tables will be sent across to your computer for processing?

That's because Access is a file, a 'desktop database' that can't think – so all the information is sent to your computer's brain to sort out. And that spells major network traffic. But other more powerful types, live 'database servers', send back only the information you need.

Also, when open to multiple users, corruption in Access databases is a distinct possibility/probability. Trust me, I've been there, done that, got the sack.

So what other types of 'database servers' are there? You may overhear some of the following mentioned at nerdy cocktail bars:

- Sybase SQL Server
- SQL Server by Microsoft (PC version of Sybase SQL Server)
- SQL Anywhere by Sybase (cut-down version of Sybase SQL Server)
- ORACLE by Oracle, surprisingly
- DB2 by IBM

Which is the best? Whoah, don't ask me that question. If I told everyone I recommend the latest version of SQL Server, I'd be hit by a thousand flames and probably a whopping great Oracle lawsuit. Err... **DOH!**

Top Tip: *In a corporate multi-user environment, the need for a big boy database server such as SQL Server becomes apparent. If you're not convinced, delve into the Access help and search for 'Access specifications' – which shows you the limitations of the .MDB format. Now visit the above database server websites and compare the two – often the big ones do stuff Access can't even think about. You'll also find that many corporations disallow the use of Access databases to store mission-critical information, opting for a more powerful database server.*

Choosing the database server to use can also take a little deciding. Consider:

- Your data storage requirements
- Your development tools (for example, Microsoft make it easier for Visual Basic programmers to connect with their own SQL Server, hint hint)
- Training and maintenance issues

Wow, I'm getting slightly serious now... and that's not my scene. So I'll end this section by summarising in a sentence; there is life beyond Access... it ain't easy, but it pays.

GROOVY DATABASE TOOLS


If you've still got your Visual Basic 6 packaging, you may notice a huge son-of-a-gun green sticker on the box, proudly claiming "**Includes Visual Database Tools**".

And this isn't just another Department-of-Justice-type lie; Visual Basic really does include a few cool tools that let you mess around with databases within the development environment.

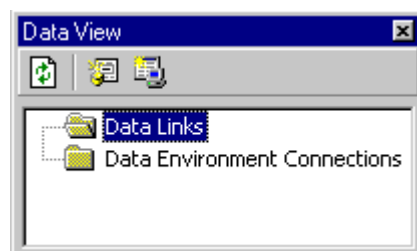
Microsoft created this groovy little feature to allow professional developers to alter the structure of major databases whilst remaining within VB – so print out this section and take it to a machine, perhaps at work, where you can access SQL Server or one of the other database servers mentioned earlier.

If you **only** have Microsoft Access, you'll feel about as left out in this section as Pope John Paul in a French brothel.

So let's try it out. No, not the brothel, but rather more excitedly, the Visual Database tools:

- Create a new Standard EXE project
- Click the small 'Data View'  icon on the toolbar

A popup window similar to the below should appear:



- Hit the 'Add a New Data Link'  button

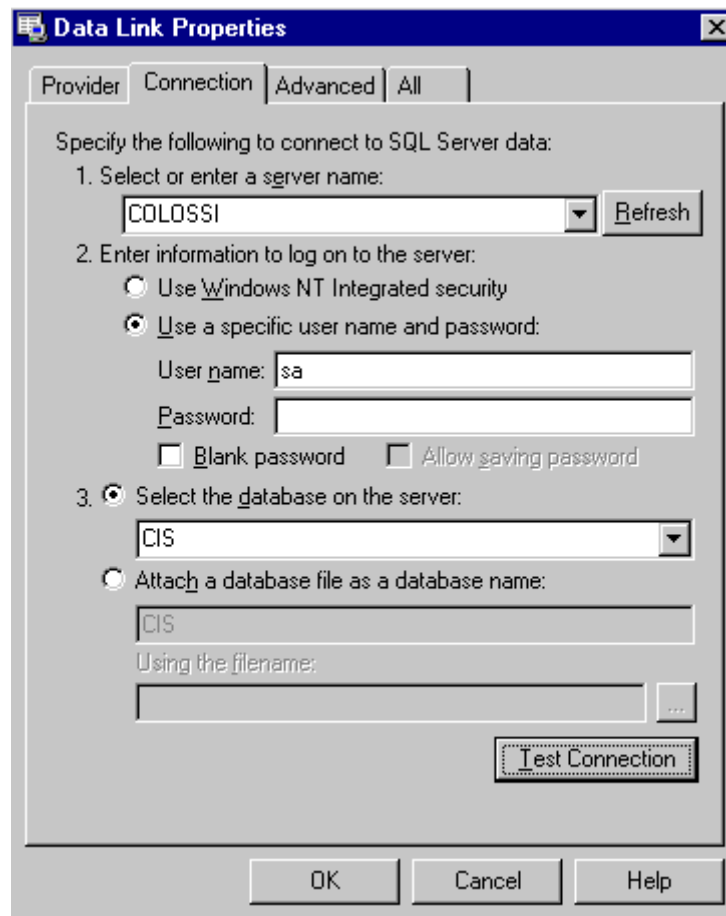
We've seen the screen in front of you before, whilst connecting the ADO control to a database.

- Select 'Microsoft OLE Provider for SQL Server' (or your database provider) from the list and click Next

If you don't have SQL Server or similar on your computer – nor on any other network machine – you should ignore all my SQL-specific comments and fill in the screen just as you did for the Access database in previous instalments. However be aware that you can't use these Visual Database tools to change the structure of an Access database – only for editing data.

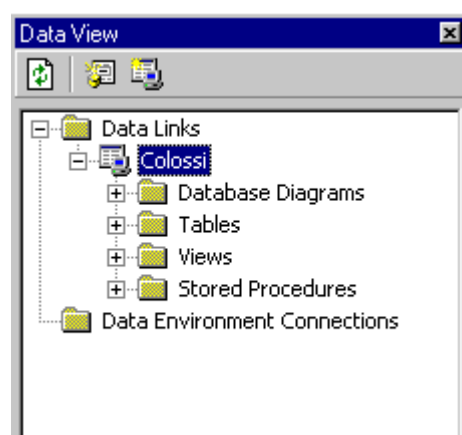
- On the next screen, enter a server name plus any login details – then select a database name

Your completed screen should look a little like this:



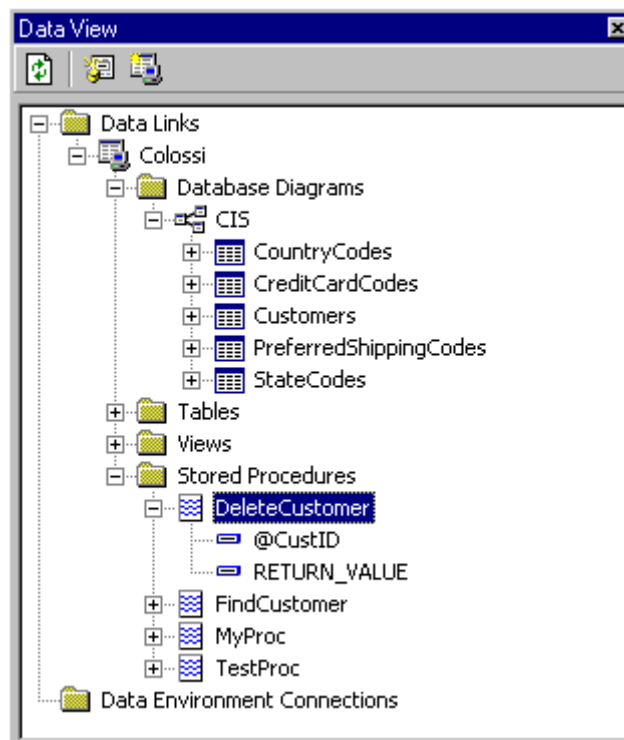
- Hit that 'Test Connection' button – and cross your fingers. It should report back that everything is A-OK
- Click OK
- You should now be in a position to christen the Data Link – I've called mine Colossi, after the server to which it is linked
- Click the little + beside your data link

The Data View window should now look something like this:



And from here you can do virtually anything with your database – define relationships, create new tables, knock out a couple of views, write stored procedures or do a few other weird things we haven't yet covered – simply by right-clicking one of the folders and selecting the 'New...' option.

Clicking the + sign next to a folder will show its contents. For example, expanding the Tables tree will display all tables within my CIS database. I can then edit the table structure, add information to the table or even delete the table – just by right-clicking and selecting the appropriate option.



In this section, I've shown you how to use the Visual Database tools to link direct into a major database format. But each type has its own little quirks, so I shan't delve too far into this subject for fear of excluding virtually all of our readers – except perhaps Crazy Jo of Mississippi and my technically au fait pet iguana, Strangely Brown.

[Ed: I remember him!]

But you now know how to access the Visual Database tools. And you'll probably need them as you progress onwards in the wacky world of enterprise development.

So how do you jump to those upper echelons of the nerd-world? Read on...

WHAT NEXT?

So what's to do from here?

If I've inspired you to take your database development skills to the next level, read on. If not, perhaps it's time to backtrack to the advice of tutorial one - move into biology and study subtractive colour mixing. Or become a Welsh sheep farmer. Or something

But if you'd like more, your next step is to start really playing around with database code. Make sure you can retrieve figures from an Access database completely in code – **without** referring to previous tutorials. Then get to know the Access query builder and its various cool uses... don't forget, it's your friend!

Your next stop is to start learning more about the more powerful databases such as SQL Server or ORACLE. As you walk down that road, you'll also bump into 'three-tier architecture' - which is just a fancy word that means splitting an application up into a few different bits, making it easier to maintain.

And if you're really geeky, you'll end up surrounded by inexplicable three-letter acronyms such as MTS, IIS, ASP and MSMQ. Hold on, the latter is a four-letter acronym, but I'll let it pass.

So where can you learn all this? I'd recommend books. I know, I know, it's difficult to find time, blah blah, you have to finish that assignment, blah blah, the toilet needs unblocking, zzzz...

Listen up, if you want to start earning mega-bucks in the world of Visual Basic and databases, it's time to put the needs of your bathroom to one side. Ideally, I'd recommend you purchase the following book:

- **Professional VB Databases** – £35.99 GBP / \$49.99 USD - by Charles William, Wrox Press

Author Charles Williams knows what he's talking about – and drags you through the ups-and-downs of SQL Server, Structured Query Language, three-tier architectures, classes, reports, data warehousing... and loads more.

If you really are keen on taking your skills to the next level, this is definitely the next step.

Alternatively, here are another couple of other bedtime reads from my bookshelf:

- **Hands On SQL Server 7 with VB6** - £37.49 GBP / \$40 USD – by Wayne Freeze, Prima Tech

Good solid book based around three major projects. Excellent way to start tinkering and discover the secrets of SQL Server. Also puts a lot of emphasis on proper planning... something I never do... ;-))

- **Teach Yourself Database Programming in Visual Basic 6** - £41.50 GBP / \$45 USD – by Curtis Smith and Michael Amundsen, Sams Publishing

Thick 900-page book that covers most of what we've discussed in this tutorial, but in more detail. Actually, about 800 pages more detail. Particularly useful if you're wanting to study more physical database code

CONCLUSION

This week, I babbled on about the different types of database out there, then we took a little look at the Visual Database tools – which could come in handy at work or later in your programming career.

I finished by chatting about what you should do next – and recommended a couple of suitable publications.

Don't forget to let me know how you get on. My personal e-mail address is karl@karlmoore.com - I look forward to hearing from you!

And that just about ends this final instalment of the Visual Basic Database tutorial. Rumour has it I'll be in a couple of months covering the scary world of ActiveX, but I'm not so sure.

Still, until the next time, this is your supercool host, Karl Moore, saying goodnight for tonight. Goodnight!

WRITING A REPORT

Some really tricky users don't just want to throw information *into* a system, they also want a little *output* in the form of reports. Sure, **we** know they're just being picky... but you'd better oblige in the faint hope of a pay rise sometime within the next millennia.