# 2. Database Objects

## Intro

Starting with mathematical number crunching in an engineering setting, most of the data I worked with was not stored; it was initially input and calculated using equations. The results could range from a few lines to several thousand lines. Getting input from the user during execution was near to impossible.

After college, I began to see the world of databases. The logic used was still sequential, but, unlike the programs I had written in the past, now *storing* information, as opposed to calculating it, was the primary purpose.

With the advent of Windows and the overwhelming markets for a graphical user interface (GUI), I moved to Access when it came out in the early 90s. It took time to translate my sequential way of thinking to that of event-driven system acting on objects.

Nearly every day since I started working with Access, I learn more about it. It is an incredible application. This document is just the tip of the iceberg… but it's a good start.

*"The more I learn, the more I realize I don't know."*

— Albert Einstein

## An Access Application is a Container

Folks often use the term "Application" and "Database" interchangeable when referring to Access. Technically, "application" would be the correct term but you will usually see "database" being used in this document.

Analogous to a bucket of seashells, an Access application is a container for many different types of objects. The main objects can be divided into 6 categories: Tables, Queries, Forms, Reports, Macros, and Modules.

Like a toaster or a washing machine that is composed of other distinct parts, each main object type in Access contains other object types.

### Tables

The most important objects are tables, which is where data is stored and defined. Tables are composed of rows (records) and columns (fields). Within each table, indexes (lists that Access makes much like indexes in the back of a book to make finding things easier) can be defined.

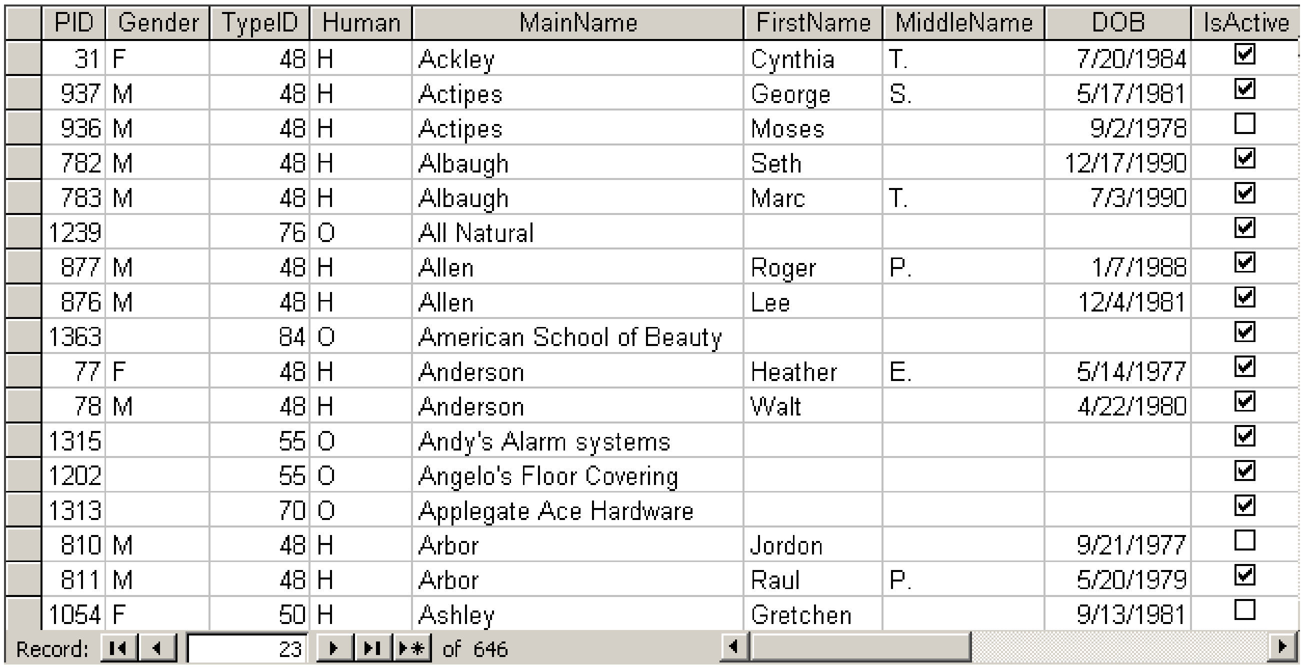
As a general rule, Tables are what all other objects are ultimately based on and act upon. Tables can be resident or linked to other sources. When you open a table, you see a *datasheet view* of the information.

Tables are connected to each other by defining *Relationships*. While you do not have to define a relationship to link tables, it is best to do so.

## *Datasheet View of a Table*

Each row of data is a record. Each column is a field. Within a record, the field describes an attribute of the record. The field names, or captions, are displayed across the top as column headings. In my opinion, you should not use captions in table definitions because they mask the real names. In other words, it is best to see the real field names when you look at the datasheet view of a table.

**Figure 2-1 Datasheet View of a Table**



| PID | Gender | TypeID | Human | MainName | FirstName | MiddleName | DOB | IsActive |
|---|---|---|---|---|---|---|---|---|
| 31 | F | 48 | H | Ackley | Cynthia | T. | 7/20/1984 | ✓ |
| 937 | M | 48 | H | Actipes | George | S. | 5/17/1981 | ✓ |
| 936 | M | 48 | H | Actipes | Moses | | 9/2/1978 | ☐ |
| 782 | M | 48 | H | Albaugh | Seth | | 12/17/1990 | ✓ |
| 783 | M | 48 | H | Albaugh | Marc | T. | 7/3/1990 | ✓ |
| 1239 | | 76 | O | All Natural | | | | ✓ |
| 877 | M | 48 | H | Allen | Roger | P. | 1/7/1988 | ✓ |
| 876 | M | 48 | H | Allen | Lee | | 12/4/1981 | ✓ |
| 1363 | | 84 | O | American School of Beauty | | | | ✓ |
| 77 | F | 48 | H | Anderson | Heather | E. | 5/14/1977 | ✓ |
| 78 | M | 48 | H | Anderson | Walt | | 4/22/1980 | ✓ |
| 1315 | | 55 | O | Andy's Alarm systems | | | | ✓ |
| 1202 | | 55 | O | Angelo's Floor Covering | | | | ✓ |
| 1313 | | 70 | O | Applegate Ace Hardware | | | | ✓ |
| 810 | M | 48 | H | Arbor | Jordon | | 9/21/1977 | ☐ |
| 811 | M | 48 | H | Arbor | Raul | P. | 5/20/1979 | ✓ |
| 1054 | F | 50 | H | Ashley | Gretchen | | 9/13/1981 | ☐ |

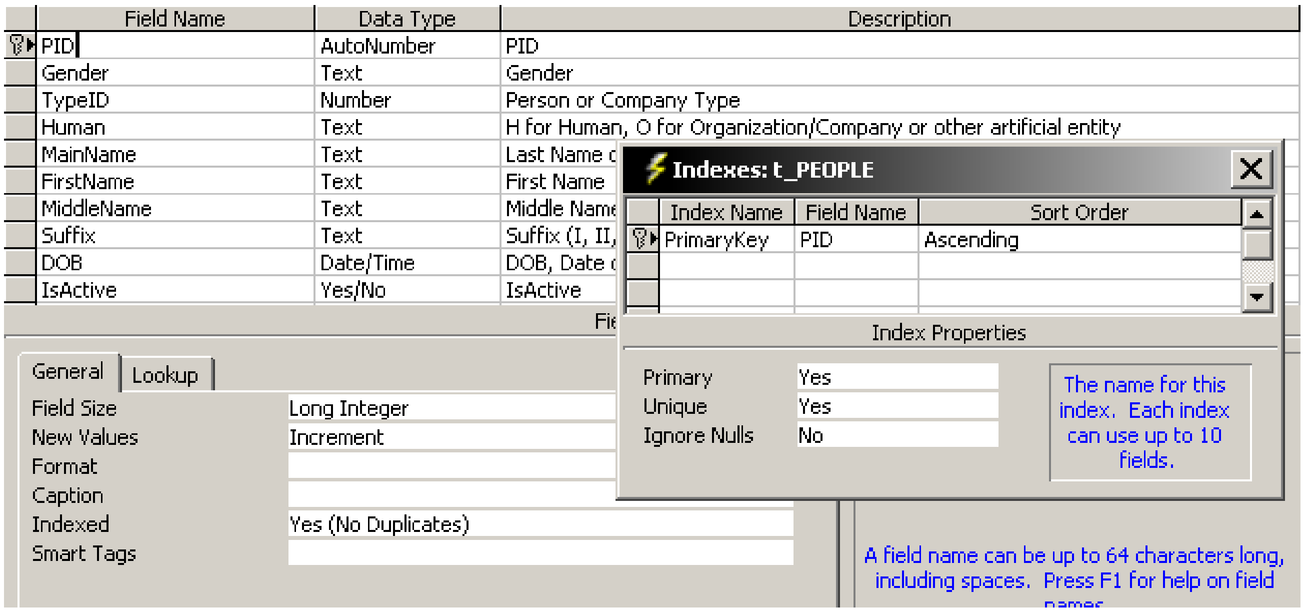
Record: 23 of 646

## *Design View of a Table*

When you design a table, you create the field names, specify what type of data the field will hold (like text, number, or date), create a description that explains more about the field, and specify other properties such as size, format, default value, and whether or not the field will be indexed.

Each table should have one field (or a combination of fields) that will be unique to each record. This field or combination can be designated as a Primary Key. It is common to use an AutoNumber for the primary key. An AutoNumber is a special form of the Long Integer data type whereby Access automatically assigns the values. Although, to define a field to be a Long Integer, you use Number as the Data Type and Long Integer as the Field Size, Access does actually treat a Long Integer as a specific Data Type.

**Figure 2-2 Design View of a Table**



| Field Name | Data Type | Description |
|---|---|---|
| PID | AutoNumber | PID |
| Gender | Text | Gender |
| TypeID | Number | Person or Company Type |
| Human | Text | H for Human, O for Organization/Company or other artificial entity |
| MainName | Text | Last Name c |
| FirstName | Text | First Name |
| MiddleName | Text | Middle Name |
| Suffix | Text | Suffix (I, II, |
| DOB | Date/Time | DOB, Date c |
| IsActive | Yes/No | IsActive |

General | Lookup

| | |
|---|---|
| Field Size | Long Integer |
| New Values | Increment |
| Format | |
| Caption | |
| Indexed | Yes (No Duplicates) |
| Smart Tags | |

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Indexes: t_PEOPLE

| Index Name | Field Name | Sort Order |
|---|---|---|
| PrimaryKey | PID | Ascending |

Index Properties

| | |
|---|---|
| Primary | Yes |
| Unique | Yes |
| Ignore Nulls | No |

The name for this index. Each index can use up to 10 fields.

Without tables, a database normally has little value as this is the where data is typically stored (exceptions are database properties, INI and other external files and the registry). Designing tables is discussed in greater detail in the Normalization section of this document.

## Queries

The most common use of a Query is to SELECT information, specify criteria (filter), and sort. Queries show information from one or more tables as records (rows) and fields (columns). Records can be sorted; they can also be filtered so that only certain records show. For example, let us assume you are printing a Denver phonebook from a database that contains information for the whole country; you can specify criteria to limit records to those whose city is 'Denver'. Then you can sort the list alphabetically by last name then first name.

If you frequently find yourself looking at the same information (perhaps for different criteria such as a date range or a region), it would be good to define a Report instead of using a Query to show the data. When a Report is based on a Query, that query is the RecordSource.

The best object to use for changing data is a Form. A Query can be used for a form RecordSource. Word of caution: even though a query can pull from many tables, good practice dictates that you should only change data in one table on each form. If you wish to add or update data in a related table at the same time, it is best to use a form/subform(/subsubform) scenario.

Another thing a query could be used for is to specify the RowSource of a combo or listbox. As for Properties... in most instances where you can use a table, you can also use a query – and vice versa. This is why tablenames and querynames cannot be the same.

Queries give you a way to quickly answer ad-hoc questions: list all sales for a particular quarter; identify duplicate information so you can fix it; find unmatched records so you can fill in missing data; switch (translate) rows into columns using a Crosstab (Pivot in Excel) so you can see, for instance, sales in columns under the months -- derived from daily amounts that are being stored.

In addition to *select*ing data, queries can perform *actions* to manipulate data such as Delete (get rid of Records), Update (modify records), Append (add records), and Make Tables.

The statement defining a Query is stored using SQL (Structured Query Language). Many SQL statements can be displayed graphically on the QBE (Query-By-Example) grid in Design View.

As with tables, opening a query that selects information uses the *datasheet view* to display the information.

In this example, three tables have been used to show information:

1. t_PEOPLE has fields describing a person such as first name, last name, gender; or main name for company. Each record is uniquely identified using a field called PID (People ID).

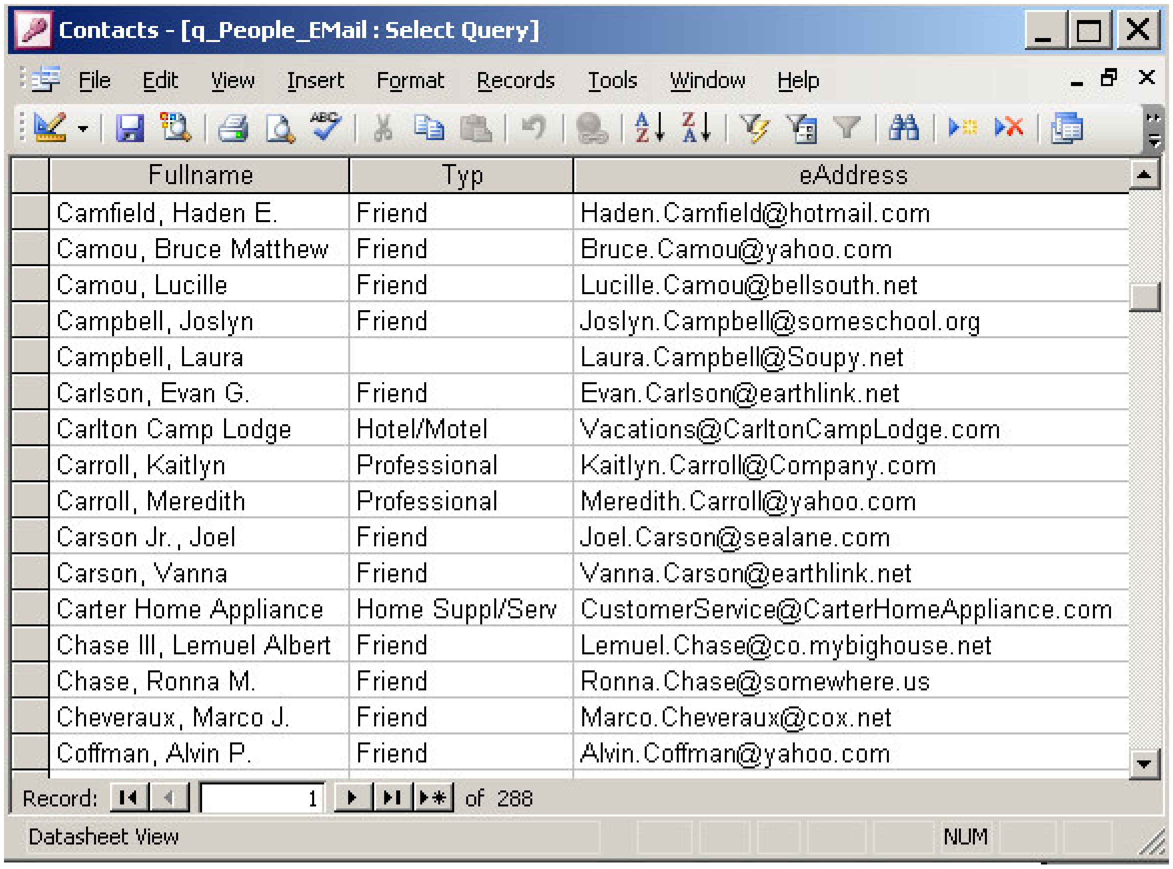
2. t_Types is used to categorize records such as: Friend, Family, Professional, Auto, Hospital, and Art Supplies. Each record in this table is uniquely identified using a field called TypeID. By storing a corresponding TypeID in the People table, a short number can be stored in the People table and the longer text can be displayed at any time from the Types table . The Types table is commonly referred to as a lookup table.

3. t_eAddresses stores email address. It is linked to the People table using a field called PID, which is short for PeopleID. Since PID is such a common field, it is abbreviated.

## *Datasheet View of a Select Query*

The datasheet view of a query looks like the datasheet view of a table.

**Figure 2-3 Datasheet View of a Select Query**



Contacts - [q_People_EMail : Select Query]
File Edit View Insert Format Records Tools Window Help
Record: 1 of 288
Datasheet View
NUM

| Fullname | Typ | eAddress |
|---|---|---|
| Camfield, Haden E. | Friend | Haden.Camfield@hotmail.com |
| Camou, Bruce Matthew | Friend | Bruce.Camou@yahoo.com |
| Camou, Lucille | Friend | Lucille.Camou@bellsouth.net |
| Campbell, Joslyn | Friend | Joslyn.Campbell@someschool.org |
| Campbell, Laura | | Laura.Campbell@Soupy.net |
| Carlson, Evan G. | Friend | Evan.Carlson@earthlink.net |
| Carlton Camp Lodge | Hotel/Motel | Vacations@CarltonCampLodge.com |
| Carroll, Kaitlyn | Professional | Kaitlyn.Carroll@Company.com |
| Carroll, Meredith | Professional | Meredith.Carroll@yahoo.com |
| Carson Jr., Joel | Friend | Joel.Carson@sealane.com |
| Carson, Vanna | Friend | Vanna.Carson@earthlink.net |
| Carter Home Appliance | Home Suppl/Serv | CustomerService@CarterHomeAppliance.com |
| Chase III, Lemuel Albert | Friend | Lemuel.Chase@co.mybighouse.net |
| Chase, Ronna M. | Friend | Ronna.Chase@somewhere.us |
| Cheveraux, Marco J. | Friend | Marco.Cheveraux@cox.net |
| Coffman, Alvin P. | Friend | Alvin.Coffman@yahoo.com |

In Figure 2-3, the Fullname column shows a combination of:

- Main name
  In the case of a human, this is the last name. For a company, it is the company name.

- First name and Middle name or initial
  Some human records have middle name or initial specified and some don't. Companies don't have first names or middle names, so that information is ignored – as well as the commas and spaces that act as separators.
- Suffix
  Some have suffix (Jr., III) and some don't.

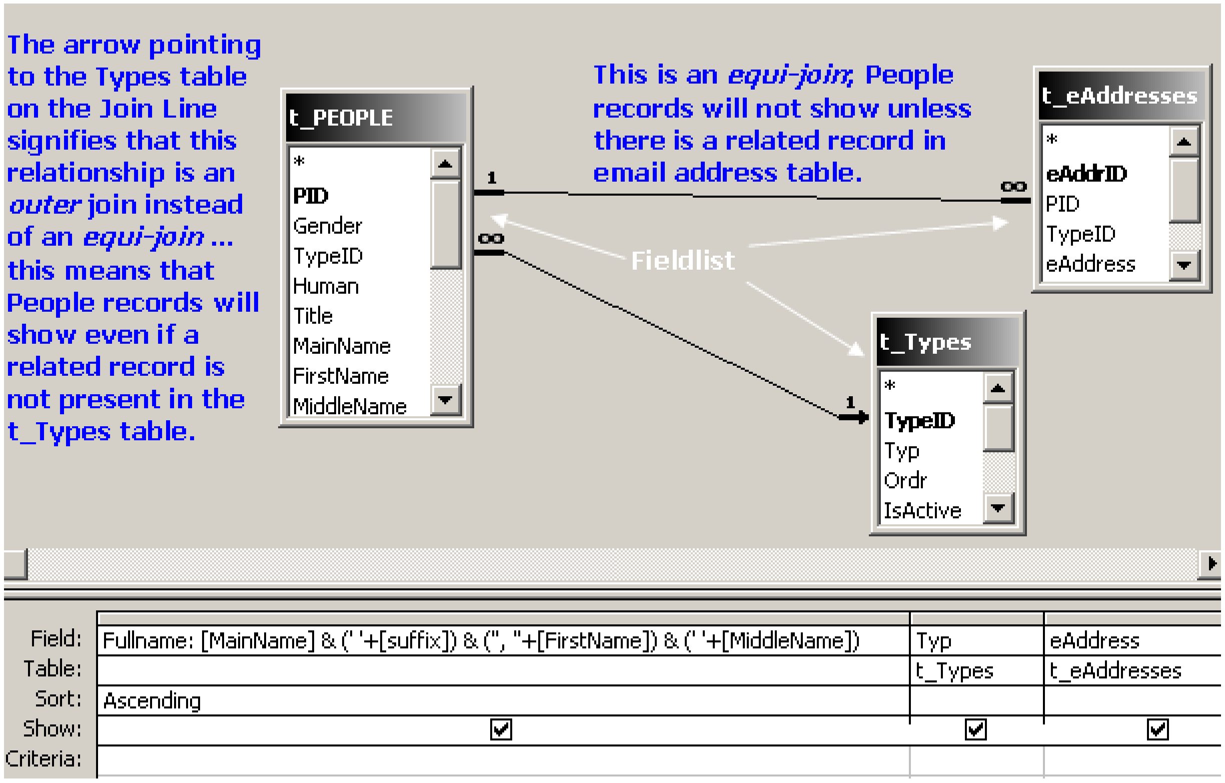
In the second column, type of contact, the word 'Typ' is used since 'Type' has a special meaning to Access (reserved words will be discussed later)

The third column shows the email address.

### *Design View of a Select Query*

We are starting you out with a slightly complicated example; the simple one comes next <smile>. Figure 2-4 shows the QBE (Query-By-Example) grid and fieldlists for the data displayed in the select query of Figure 2-3.

**Figure 2-4 Design View of a Query**

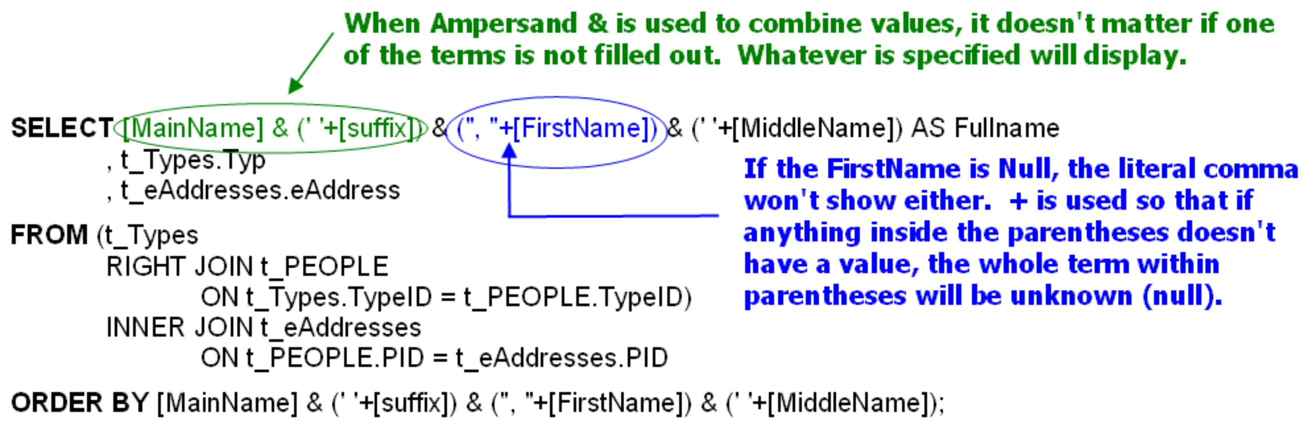

The arrow pointing to the Types table on the Join Line signifies that this relationship is an *outer* join instead of an *equi-join* ... this means that People records will show even if a related record is not present in the t_Types table.
This is an *equi-join*, People records will not show unless there is a related record in email address table.
Fieldlist
t_PEOPLE
*
PID
Gender
TypeID
Human
Title
MainName
FirstName
MiddleName
t_eAddresses
*
eAddrID
PID
TypeID
eAddress
t_Types
*
TypeID
Typ
Ordr
IsActive
1
∞
∞
1

| | | | |
|---|---|---|---|
| Field: | Fullname: [MainName] & (' '+[suffix]) & (", "+[FirstName]) & (' '+[MiddleName]) | Typ | eAddress |
| Table: | | t_Types | t_eAddresses |
| Sort: | Ascending | | |
| Show: | ✓ | ✓ | ✓ |
| Criteria: | | | |

## *SQL View of a Select Query*

The SQL statement is what Access stores internally to render a query.

**Figure 2-5 SQL View of a Select Query**



When Ampersand & is used to combine values, it doesn't matter if one of the terms is not filled out. Whatever is specified will display.

```
SELECT [MainName] & (' '+[suffix]) & (", "+[FirstName]) & (' '+[MiddleName]) AS Fullname
        , t_Types.Typ
        , t_eAddresses.eAddress
FROM (t_Types
        RIGHT JOIN t_PEOPLE
                ON t_Types.TypeID = t_PEOPLE.TypeID)
        INNER JOIN t_eAddresses
                ON t_PEOPLE.PID = t_eAddresses.PID
ORDER BY [MainName] & (' '+[suffix]) & (", "+[FirstName]) & (' '+[MiddleName]);
```

If the FirstName is Null, the literal comma won't show either. + is used so that if anything inside the parentheses doesn't have a value, the whole term within parentheses will be unknown (null).

When you specify the calculated field to be sorted, this is the SQL that Access will construct. It is more efficient, however, to use this as your Order By clause:

**ORDER BY** [MainName], [FirstName], [MiddleName];

The **semi-colon** at the end of the SQL statement indicates the end of the statement.

## *Calculated Field*

This is a calculated field and will be called 'Fullname'

**[MainName] & (' '+[suffix]) & (", "+[FirstName]) & (' '+[MiddleName]) AS Fullname**

This equation *concatenates* (combines) information from main name, first name, middle name, and suffix into one place. Literal values, such as spaces and commas are concatenated where needed. Note that literal values can be enclosed in single or double quotes … the quote marks just have to be balanced.

*Definition:* **Concatenate** is used in database terminology to mean 'to combine', or connect, or link together. In databases, one reason it is important to separate data into the smallest possible unit is because it is *much* easier to concatenate than it is to parse.

*Definition:* **Parse** is a term meaning 'to separate'. If you store someone's full name in one field and decide you want just the first name, you would need to parse that out from the whole field.

***Difference between + and &***

**&** and **+** are both *Operators*

The standard *Concatenation Operator* is ampersand (&). If a term that is concatenated is Null (has no data; unknown), all terms will display if you use ampersand.

The *Addition Operator* is the plus sign (+) … but, even if one of the terms has a value, the result will be Null if any term is Null (kind of like multiplying by 0). As in math, what is enclosed in parentheses will be evaluated first.

```
Null + "anything" = Null
Null & "anything = "anything"

"something " + "anything" = "something anything"
"something " & "anything" = "something anything"
```
*no difference because both of the terms have a value*

```
Null + "" = Null
Null & "" = ""

(Null + " ") & "Lastname" = "Lastname"
(Null & " ") & "Lastname" = " Lastname"
```
*in the second case, the parentheses do not make a difference, each term is concatenated -- and note the space in the result before Lastname*

Do you see the difference between using + and using & ? For instance, if you want to add a space between first and last name but you are not sure that first name will be filled out, you can do this:

```
(Firstname + " ") & Lastname
```

What is in the parentheses is evaluated first -- then it is concatenated to what comes next

You might also want to do this:

```
(Firstname + " ") & (Middlename + " ") & Lastname
```

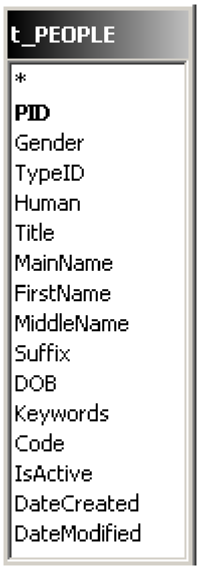
Combining + and & in an expression gives you a way to make the result look right without having to test if something is not filled out.

What if firstname is filled but nothing else? There will be a space at the end. Usually, this is not a problem but if you want to chop it off, you can wrap the whole expression in the Trim function, which truncates leading and trailing spaces.

```
Trim((Firstname + " ") & (Middlename + " ") & Lastname)
```

### *Fieldlist*

**Figure 2-6 Fieldlist**


t_PEOPLE
*
PID
Gender
TypeID
Human
Title
MainName
FirstName
MiddleName
Suffix
DOB
Keywords
Code
IsActive
DateCreated
DateModified

A *fieldlist* is a list showing fields that are in a table or query.

The titlebar shows the name of the table or query and its fields are listed below.

To add a fieldlist to a query:

Click the Show Table icon

OR

from the menu, choose →

Query, Show Table …

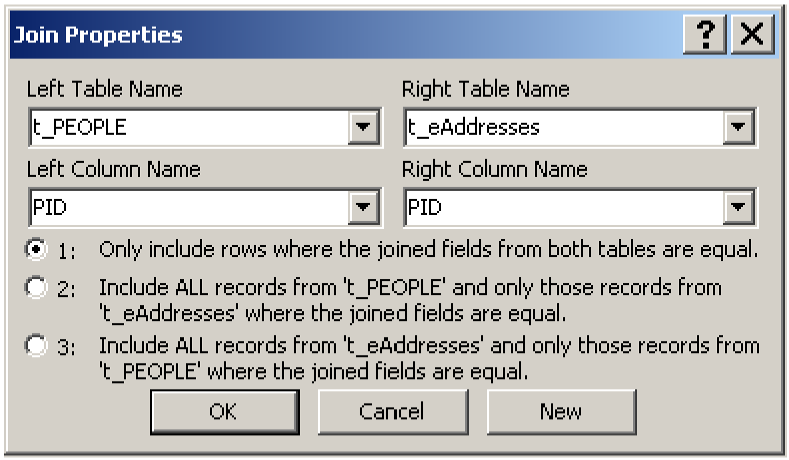
### *Joins*

A *join* is a line between two fieldlists showing how they relate to each other.

In the query design example, the line between the table with People information and the table with eMail addresses is an *equi-join* (aka *INNER JOIN).* This means that only people records that also have an email address will be included.

When you Right-Click on a join line, you can choose to edit the Join Properties. You can also Double-Click a join line to bring up the properties window directly.

**Figure 2-7 Join Properties – Equi-Join**

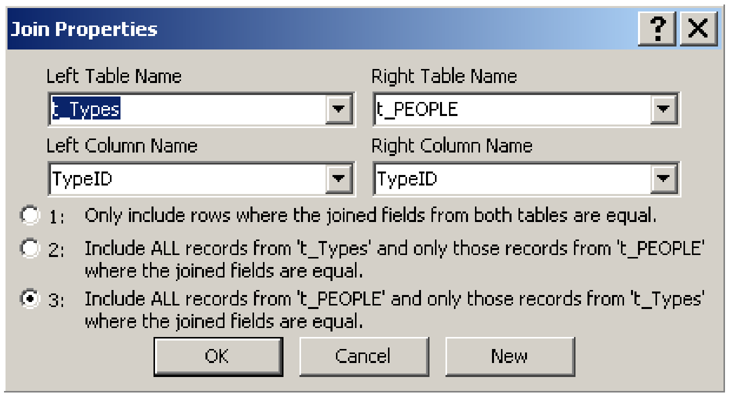

Join Properties

Left Table Name: t_PEOPLE
Right Table Name: t_eAddresses
Left Column Name: PID
Right Column Name: PID

[● 1: Only include rows where the joined fields from both tables are equal.]
○ 2: Include ALL records from 't_PEOPLE' and only those records from 't_eAddresses' where the joined fields are equal.
○ 3: Include ALL records from 't_eAddresses' and only those records from 't_PEOPLE' where the joined fields are equal.

An equi-join is not always desired. Another fieldlist in the example shows the type person (or company) . What if the type is not filled out (TypeID field in the people table) but we have an email address? In that case, we still want the person and the email address to show. How do we do that? We can use an *outer* join, which is also called a *Left* join or a *Right* join depending on how the SQL statement is written.

**Figure 2-8 Join Properties - Right Join**



Join Properties

Left Table Name: t_Types
Right Table Name: t_PEOPLE
Left Column Name: TypeID
Right Column Name: TypeID

1: Only include rows where the joined fields from both tables are equal.
2: Include ALL records from 't_Types' and only those records from 't_PEOPLE' where the joined fields are equal.
[✓ 3: Include ALL records from 't_PEOPLE' and only those records from 't_Types' where the joined fields are equal.]

## Data Example

To better understand what is happening with the Join lines, assume that we have the following data in our tables:

**t_People**

| PID | Fullname | TypeID |
|---|---|---|
| 768 | Camfield, Greyson D. | |
| 769 | Camfield, Haden E. | **48** |
| 35 | Camou, Bruce Matthew | **48** |
| 36 | Camou, Lucille | **48** |
| 1214 | Campbell Car Wash | **58** |
| 970 | Campbell, Joslyn B. | **48** |
| 971 | Campbell, Laura | |
| 709 | Cannon, Dalton M. | **48** |
| 710 | Cannon, Quinn G. | **48** |
| 898 | Carlson, Dylan H. | **48** |
| 899 | Carlson, Evan G. | **48** |
| 1253 | Carlton Camp Lodge | 81 |
| 1340 | Carpets To Go | **70** |
| 1007 | Carroll, Kaitlyn | **50** |
| 1006 | Carroll, Meredith V. | 50 |
| 59 | Carson Jr., Joel | **48** |
| 60 | Carson, Vanna S. | **48** |

*Red indicates there is no corresponding PID in the email addresses table – so this name will not show in the query.*

**t_Types**

| TypeID | Typ |
|---|---|
| **48** | Friend |
| **50** | Professional |
| 54 | Store |
| 55 | Home Suppl/Serv |
| 56 | Sports |
| 57 | Books |
| **58** | Auto |
| 69 | Music Store |
| **70** | Bldg Supplies |
| 71 | Dance |
| 76 | Health Store |
| 77 | Games |
| 78 | Computer/Electronics |
| 79 | Flowers |
| 80 | Coins |
| **81** | Hotel/Motel |

*Bold means that this **TypeID** is used in the list of people that are shown*

**t_eAddresses**

| PID | eAddress |
|---|---|
| 35 | Bruce.Camou@yahoo.com |
| 36 | Lucille.Camou@bellsouth.net |
| 59 | Joel.Carson@sealane.com |
| 60 | Vanna.Carson@earthlink.net |
| 65 | Ronna.Chase@somewhere.us |
| 66 | Lemuel.Chase@mybighouse.net |
| 731 | Marco.Cheveraux@cox.net |
| 769 | Haden.Camfield@hotmail.com |
| 899 | Evan.Carlson@earthlink.net |
| 970 | Joslyn.Campbell@someschool.org |
| 971 | Laura.Campbell@Soupy.net |
| 1006 | Meredith.Carroll@yahoo.com |
| 1007 | Kaitlyn.Carroll@Company.com |
| 1253 | Vacations@CarltonCampLodge.com |

*Not all people shown have email addresses*

In this example, 'Campbell, Laura' does not have a type specified, but she *does* have an email address – so she will still show in the query.

The results of our earlier query example run on this data would net the following results:

| Fullname | Typ | eAddress |
|---|---|---|
| Camfield, Haden E. | Friend | Haden.Camfield@hotmail.com |
| Camou, Bruce Matthew | Friend | Bruce.Camou@yahoo.com |
| Camou, Lucille | Friend | Lucille.Camou@bellsouth.net |
| Campbell, Joslyn B. | Friend | Joslyn.Campbell@someschool.org |
| Campbell, Laura | | Laura.Campbell@Soupy.net |
| Carlson, Evan G. | Friend | Evan.Carlson@earthlink.net |
| Carlton Camp Lodge | Hotel/Motel | Vacations@CarltonCampLodge.com |
| Carroll, Kaitlyn | Professional | Kaitlyn.Carroll@Company.com |
| Carroll, Meredith V. | Professional | Meredith.Carroll@yahoo.com |
| Carson Jr., Joel | Friend | Joel.Carson@sealane.com |
| Carson, Vanna S. | Friend | Vanna.Carson@earthlink.net |
| Carter Home Appliance | Home Suppl/Serv | CustomerService@CarterHomeAppliance.com |

Only people that *have* an email address are displayed, regardless of whether or not their type has been specified.

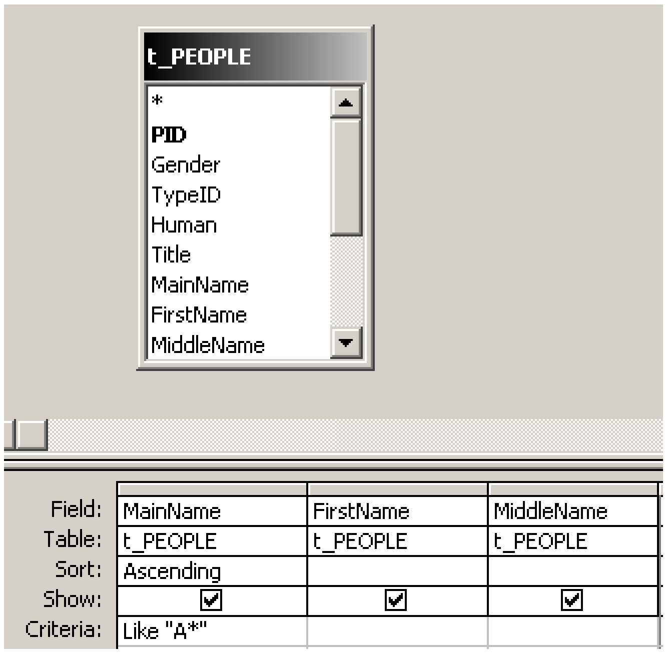
### *Simple Query Example*

Now that we started you out with a complicated query that pulled data from three tables and used different types of joins … how about an easy example?

Figure 2-9 shows a simple list of people sorted by the main name (last name for humans or company name for companies) and then the first name.

**Figure 2-9 Simple Query Example**

| MainName | FirstName | MiddleName |
|---|---|---|
| A & M Surplus | | |
| Ackerman | Ian | T. |
| Ackerman | Sidney | |
| Ackley | Cynthia | T. |
| Ackley | Steven | |
| Actipes | George | S. |
| Actipes | Moses | |
| Albaugh | Marc | T. |
| Albaugh | Seth | |
| All Natural | | |
| Allen | Lee | |
| Allen | Roger | P. |
| American School of Beauty | | |
| Anderson | Heather | E. |
| Anderson | Walt | |
| Andy's Alarm systems | | |
| Ace Hardware | | |
| Arbor | Jordon | |
| Ashley | Gretchen | |
| Astrid's Upholstry | | |
| Austin | Teresa | |
| Ayres | Graham | |



| | | | |
|---|---|---|---|
| Field: | MainName | FirstName | MiddleName |
| Table: | t_PEOPLE | t_PEOPLE | t_PEOPLE |
| Sort: | Ascending | | |
| Show: | ✓ | ✓ | ✓ |
| Criteria: | Like "A*" | | |

**The MainName field is limited to names starting with 'A'**

## *Date Functions in Queries*

In our table of people, one of the fields is DOB (Date of Birth). Suppose we want to list birthdays sorted by month and then day? This example shows some of the cool things you can do with dates! By the way, you can usually use the same functions in Excel that you see for Access!

Here are some of the results:

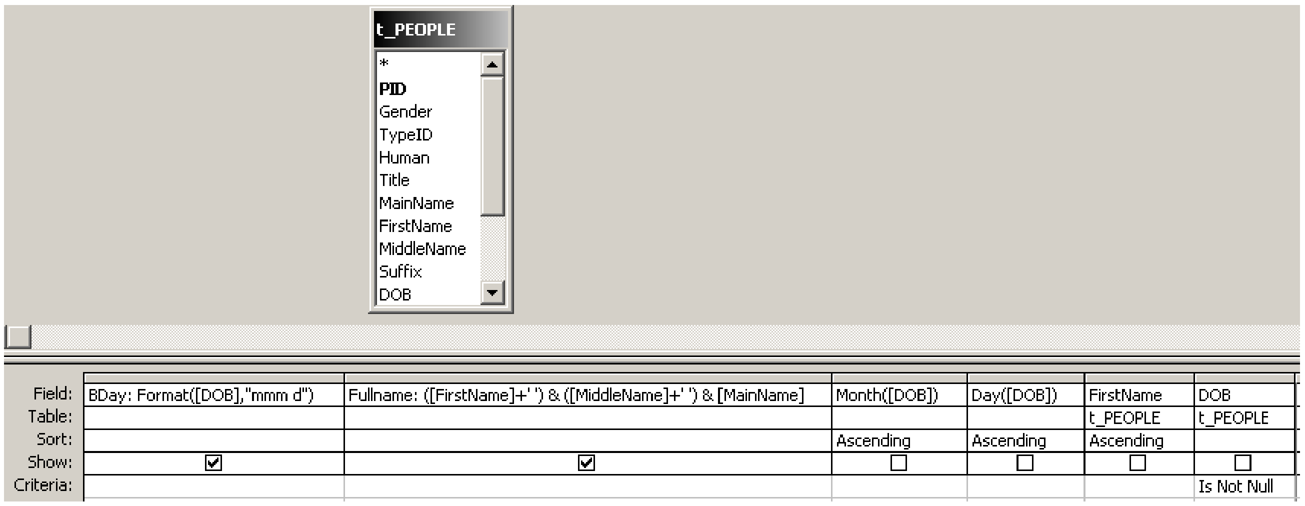
| BDay | Fullname |
|---|---|
| Jan 25 | Tyson Snider |
| Jan 26 | Alfonso A. Tenney |
| Jan 27 | Gary E. Summey |
| Jan 27 | Rudy Bradley |
| Jan 28 | Pedro R. Mcintyre |
| Jan 30 | Anna E. Deming |
| Jan 30 | Roberto K. Gaylord |
| Feb 2 | Cade R. Miller |
| Feb 2 | Fiona E. Renner |
| Feb 4 | Joshua B. Robinson |
| Feb 5 | Braxton L. Hansford |

Notice that January shows before February … even though, alphabetically, February comes first! How is that done?

How do we show 'Jan 25' from a date that looks like this: 1/25/47 ?

Here is the design view of the birthday query (DOB is DateOfBirth):

**Figure 2-10 Design View of Query Showing Birthdays**



| | | | | | | |
|---|---|---|---|---|---|---|
| Field: | BDay: Format([DOB],"mmm d") | Fullname: ([FirstName]+' ') & ([MiddleName]+' ') & [MainName] | Month([DOB]) | Day([DOB]) | FirstName | DOB |
| Table: | | | | | t_PEOPLE | t_PEOPLE |
| Sort: | | | Ascending | Ascending | Ascending | |
| Show: | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ |
| Criteria: | | | | | | Is Not Null |

There are only 2 columns that show (note the checkbox) – **Bday**, which displays as a month and day (mmm d) and **Fullname**. The other columns on the grid are for sorting or filtering.

The Month function is used to return the month number (1-12) from DOB. It is on the grid but the 'Show' box is not checked – notice the 'Ascending' is specified in the Sort cell.

The Day function is used to return the day number from DOB for sorting purposes. It also does not show.

If more than one person has a birthday on the same day, the list will then be sorted by FirstName. Choosing just first name for the sort is more efficient than specifying the calculated column.

Sorting is from left to right in the QBE grid for the three ascending sorts specified.

The Bday column uses the Format function to return the 3 character month abbreviation (mmm) and then the day of the month (d).

Records are only shown if DOB has a value – Is Not Null means the field is filled out <smile>

## Query Criteria

*To be written later*

## Forms

Forms provide a way to present data and/or menus and control the user's interaction. Forms can be *bound* or *unbound*. A Bound form is connected to a table or query by its RecordSource property (like a table or query – and remember that queries get their data from tables) and can be used to find, display, edit, and add data. Unbound forms, which have no RecordSource, can be used as menus or to hold subforms.

Forms are divided into different sections. The form header and footer sections display at the top and bottom. There are also optional Page Header and Page Footer sections that show at the top and bottom of each page when a form is printed.

The Detail section shows for each record.

In each section are controls. Textbox, combobox, listbox, checkbox, option group, option button, and image controls can be used to display and collect information . These types of controls can be bound to a field, or unbound, similar to binding a form to a RecordSet. Command button controls are often used for menus. Tab controls are used to organize information. Other types of controls include label, line, rectangle, as well as controls with complex information such as subform/subreport, calendar, video, and sound.

When a form is bound, controls that hold data can also be bound; in other words, this means that the ControlSource is a field in the table or query. Binding a form and then binding (connecting) controls on the form to fields in the RecordSource allows you to take advantage of many of the built-in features of Access like record navigation and data manipulation.

### *Different Ways to View Data in a Form*

Forms can be set up to show one record at a time or multiple records at a time. They can also pivot information in a table layout or show graphs . The default in Access, is to allow all views for a form. However, if you are creating an application for other users, it is generally best to restrict the allowed view. We will focus on the 3 main types of forms:

- Single Form View
- Continuous Form View
- Datasheet View

### *Field vs Control*

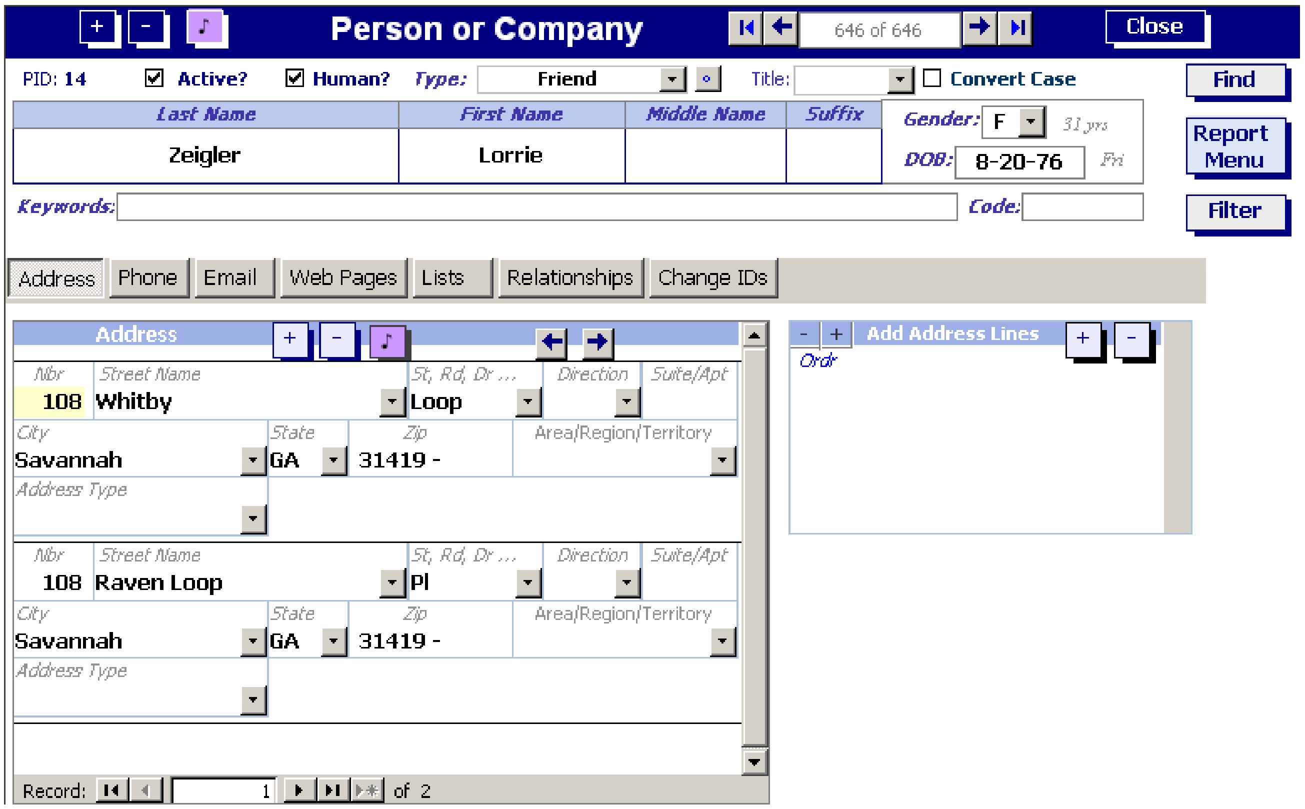
It is important to understand the difference between a field and a control. A field is a column in a table. A calculated field is an expression. A control can be a container for a field – but not all controls can be bound to fields –examples of controls that cannot be bound are label, line and rectangle controls.

The only way for a field to be placed on a form (or anything else for that matter) is in a control.

### *Form View*

This is an example of a main form with subforms (creating a main form and subforms is covered in more detail later in this document). The main form is based on the People table. Subforms are used to enter information into related tables such as Addresses, Phones, and Email. A tab control is used to organize the subforms on the main form. Records displayed in subforms are linked to records from the parent table displayed on the main using the unique identifier for the parent table which, in this case, is PID.

**Figure 2-11 Form View - Main Form and Subforms**


Person or Company

646 of 646

PID: 14 ☑ Active? ☑ Human? Type: Friend Title: ☐ Convert Case

| Last Name | First Name | Middle Name | Suffix |
|---|---|---|---|
| Zeigler | Lorrie | | |

Gender: F 31 yrs
DOB: 8-20-76 Fri

Keywords: Code:

| Nbr | Street Name | St, Rd, Dr ... | Direction | Suite/Apt |
|---|---|---|---|---|
| 108 | Whitby | Loop | | |

| City | State | Zip | Area/Region/Territory |
|---|---|---|---|
| Savannah | GA | 31419 - | |

Address Type

| Nbr | Street Name | St, Rd, Dr ... | Direction | Suite/Apt |
|---|---|---|---|---|
| 108 | Raven Loop | Pl | | |

| City | State | Zip | Area/Region/Territory |
|---|---|---|---|
| Savannah | GA | 31419 - | |

Address Type

Record: 1 of 2

### *Design View of a Form*

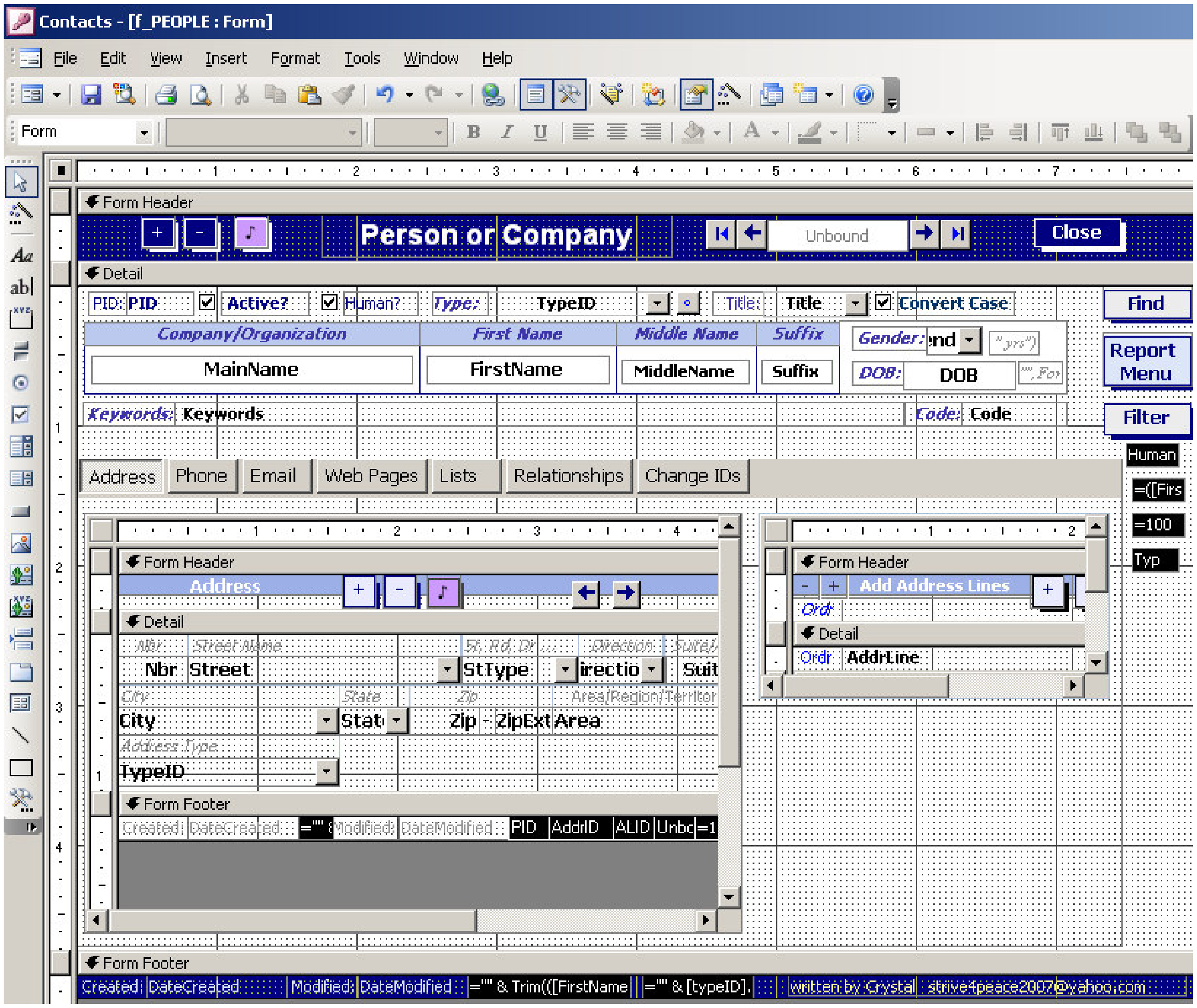
This form uses 3 sections, Form Header, Detail, and Form Footer. Hidden controls (those with the Visible property set to No) use a white foreground color on a black background – that way they show up well in design view.

Figure 2-12 shows:

- Titlebar across the top indicating you are looking at an Access form named f_PEOPLE in a project named Contacts.
- Menu bar (File, Edit, View, …)
- The Form Design and Formatting toolbars are displayed on two rows.
- The Toolbox is docked on the left.
- The horizontal ruler across the top and vertical ruler on the left (from the menu: View, Rulers)
- Grid, which shows up as dots to help line up controls (from the menu: View, Grid)

- Three sections: Form Header, Detail, and Form Footer
- A Tab Control for switching between different types of information such as Addresses, Phone, and Email
- Two subform controls on the Address tab: one for main address information and the other for additional address lines.

**Figure 2-12 Design View of a Form**



Subform controls are used to contain forms for Addresses and extra lines for addresses. Being an Aussie, Microsoft Access MVP Allen Browne asked how I would handle foreign addresses and the example he gave me had about 6 lines! As an American with somewhat standard addressing, I wasn't familiar with some of the outlandish addresses that some of you have! Well, one of the wonderful things about Access is that a table to hold what I call "standard" addresses could be set up – and then another related table allowing as many more lines as desired!

There are other subforms on other tabs – Phone, Email, Web Pages, Lists, and Relationships. On the "Change IDs" tab is a combobox to pick an ID to change all related references to so that a duplicate person can be deleted without losing any of the additional information that may have been entered.

While I realize all you have is this document and not the Contact Management application, I am leaving it up to your imagination to see some of the many ways a form can be used. If you do wish to see the Contact Management system this document references, it is available for download from:

**Contacts -- Names, Addresses, Phones, eMail, Websites, Notes**
http://www.utteraccess.com/forums/showflat.php?Cat=&Board=48&Number=1428638

To download files from UtterAccess, you need to be a member – but it is free to join and is a great resource!

## Reports

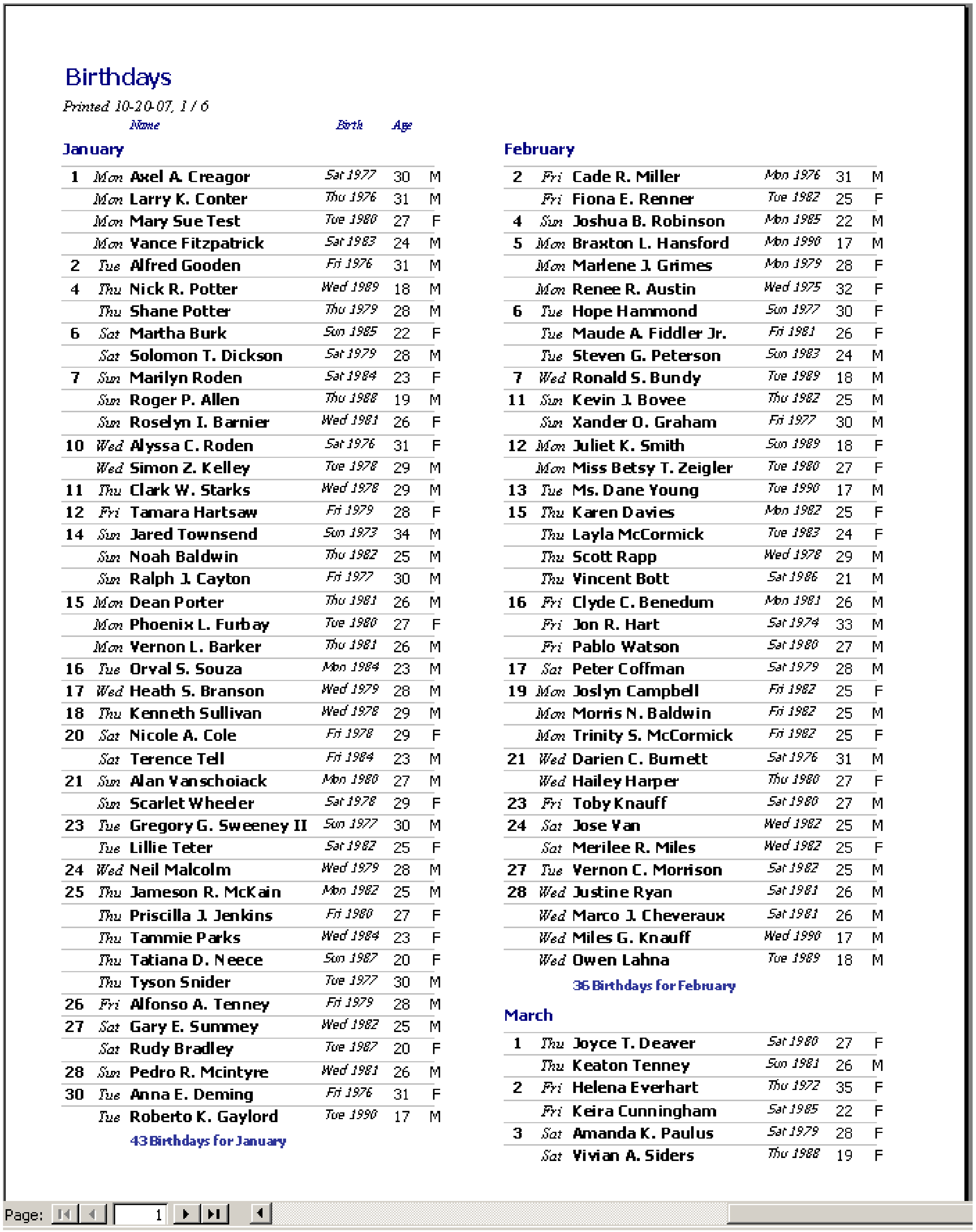
Reports are used to format, calculate, and display data. The power of reports is in being able to sort and group up to 10 levels, make everything look nice with extensive formatting capabilities, and execute code in several places. Like forms, reports contain various types of controls.

### *Print Preview of a Report*

Figure 2-13 shows a report with 2 columns. Its RecordSource is a query called q_Birthdays. In the lower-left corner is a navigation control to go to different pages of the report.

The number of birthdays listed is reported at the end of each month. Information is grouped by month and then sorted.

**Figure 2-13 Print Preview of a Report - Two Columns**



## Birthdays

*Printed 10-20-07, 1 / 6*

**January**

| | | Name | Birth | Age | |
|---|---|---|---|---|---|
| 1 | Mon | Axel A. Creagor | Sat 1977 | 30 | M |
| | Mon | Larry K. Conter | Thu 1976 | 31 | M |
| | Mon | Mary Sue Test | Tue 1980 | 27 | F |
| | Mon | Vance Fitzpatrick | Sat 1983 | 24 | M |
| 2 | Tue | Alfred Gooden | Fri 1976 | 31 | M |
| 4 | Thu | Nick R. Potter | Wed 1989 | 18 | M |
| | Thu | Shane Potter | Thu 1979 | 28 | M |
| 6 | Sat | Martha Burk | Sun 1985 | 22 | F |
| | Sat | Solomon T. Dickson | Sat 1979 | 28 | M |
| 7 | Sun | Marilyn Roden | Sat 1984 | 23 | F |
| | Sun | Roger P. Allen | Thu 1988 | 19 | M |
| | Sun | Roselyn I. Barnier | Wed 1981 | 26 | F |
| 10 | Wed | Alyssa C. Roden | Sat 1976 | 31 | F |
| | Wed | Simon Z. Kelley | Tue 1978 | 29 | M |
| 11 | Thu | Clark W. Starks | Wed 1978 | 29 | M |
| 12 | Fri | Tamara Hartsaw | Fri 1979 | 28 | F |
| 14 | Sun | Jared Townsend | Sun 1973 | 34 | M |
| | Sun | Noah Baldwin | Thu 1982 | 25 | M |
| | Sun | Ralph J. Cayton | Fri 1977 | 30 | M |
| 15 | Mon | Dean Porter | Thu 1981 | 26 | M |
| | Mon | Phoenix L. Furbay | Tue 1980 | 27 | F |
| | Mon | Vernon L. Barker | Thu 1981 | 26 | M |
| 16 | Tue | Orval S. Souza | Mon 1984 | 23 | M |
| 17 | Wed | Heath S. Branson | Wed 1979 | 28 | M |
| 18 | Thu | Kenneth Sullivan | Wed 1978 | 29 | M |
| 20 | Sat | Nicole A. Cole | Fri 1978 | 29 | F |
| | Sat | Terence Tell | Fri 1984 | 23 | M |
| 21 | Sun | Alan Vanschoiack | Mon 1980 | 27 | M |
| | Sun | Scarlet Wheeler | Sat 1978 | 29 | F |
| 23 | Tue | Gregory G. Sweeney II | Sun 1977 | 30 | M |
| | Tue | Lillie Teter | Sat 1982 | 25 | F |
| 24 | Wed | Neil Malcolm | Wed 1979 | 28 | M |
| 25 | Thu | Jameson R. McKain | Mon 1982 | 25 | M |
| | Thu | Priscilla J. Jenkins | Fri 1980 | 27 | F |
| | Thu | Tammie Parks | Wed 1984 | 23 | F |
| | Thu | Tatiana D. Neece | Sun 1987 | 20 | F |
| | Thu | Tyson Snider | Tue 1977 | 30 | M |
| 26 | Fri | Alfonso A. Tenney | Fri 1979 | 28 | M |
| 27 | Sat | Gary E. Summey | Wed 1982 | 25 | M |
| | Sat | Rudy Bradley | Tue 1987 | 20 | F |
| 28 | Sun | Pedro R. Mcintyre | Wed 1981 | 26 | M |
| 30 | Tue | Anna E. Deming | Fri 1976 | 31 | F |
| | Tue | Roberto K. Gaylord | Tue 1990 | 17 | M |

43 Birthdays for January

**February**

| | | | | | |
|---|---|---|---|---|---|
| 2 | Fri | Cade R. Miller | Mon 1976 | 31 | M |
| | Fri | Fiona E. Renner | Tue 1982 | 25 | F |
| 4 | Sun | Joshua B. Robinson | Mon 1985 | 22 | M |
| 5 | Mon | Braxton L. Hansford | Mon 1990 | 17 | M |
| | Mon | Marlene J. Grimes | Mon 1979 | 28 | F |
| | Mon | Renee R. Austin | Wed 1975 | 32 | F |
| 6 | Tue | Hope Hammond | Sun 1977 | 30 | F |
| | Tue | Maude A. Fiddler Jr. | Fri 1981 | 26 | F |
| | Tue | Steven G. Peterson | Sun 1983 | 24 | M |
| 7 | Wed | Ronald S. Bundy | Tue 1989 | 18 | M |
| 11 | Sun | Kevin J. Bovee | Thu 1982 | 25 | M |
| | Sun | Xander O. Graham | Fri 1977 | 30 | M |
| 12 | Mon | Juliet K. Smith | Sun 1989 | 18 | F |
| | Mon | Miss Betsy T. Zeigler | Tue 1980 | 27 | F |
| 13 | Tue | Ms. Dane Young | Tue 1990 | 17 | M |
| 15 | Thu | Karen Davies | Mon 1982 | 25 | F |
| | Thu | Layla McCormick | Tue 1983 | 24 | F |
| | Thu | Scott Rapp | Wed 1978 | 29 | M |
| | Thu | Vincent Bott | Sat 1986 | 21 | M |
| 16 | Fri | Clyde C. Benedum | Mon 1981 | 26 | M |
| | Fri | Jon R. Hart | Sat 1974 | 33 | M |
| | Fri | Pablo Watson | Sat 1980 | 27 | M |
| 17 | Sat | Peter Coffman | Sat 1979 | 28 | M |
| 19 | Mon | Joslyn Campbell | Fri 1982 | 25 | F |
| | Mon | Morris N. Baldwin | Fri 1982 | 25 | M |
| | Mon | Trinity S. McCormick | Fri 1982 | 25 | F |
| 21 | Wed | Darien C. Burnett | Sat 1976 | 31 | M |
| | Wed | Hailey Harper | Thu 1980 | 27 | F |
| 23 | Fri | Toby Knauff | Sat 1980 | 27 | M |
| 24 | Sat | Jose Van | Wed 1982 | 25 | M |
| | Sat | Merilee R. Miles | Wed 1982 | 25 | F |
| 27 | Tue | Vernon C. Morrison | Sat 1982 | 25 | M |
| 28 | Wed | Justine Ryan | Sat 1981 | 26 | M |
| | Wed | Marco J. Cheveraux | Sat 1981 | 26 | M |
| | Wed | Miles G. Knauff | Wed 1990 | 17 | M |
| | Wed | Owen Lahna | Tue 1989 | 18 | M |

36 Birthdays for February

**March**

| | | | | | |
|---|---|---|---|---|---|
| 1 | Thu | Joyce T. Deaver | Sat 1980 | 27 | F |
| | Thu | Keaton Tenney | Sun 1981 | 26 | M |
| 2 | Fri | Helena Everhart | Thu 1972 | 35 | F |
| | Fri | Keira Cunningham | Sat 1985 | 22 | F |
| 3 | Sat | Amanda K. Paulus | Sat 1979 | 28 | F |
| | Sat | Vivian A. Siders | Thu 1988 | 19 | F |

### *Design View of a Report*

### Report Sections

Figure 2-14 shows the design view of a report with seven sections. The Report Header section is set to zero height, but it needs to show so that the Report Footer section, which counts all the birthdays listed on the report, can be used.

At the top of each page will be the information in the Page Header section. It appears that the Page Footer section has nothing … but if this report is limited, for instance to just people that are friends, the criteria is programmed to show in this area.

The Mnth Header section is used to break the report into months like January, and February.

The Mnth Footer section contains a calculated control to count the number of birthdays listed.

Finally, the Detail section is where the data for each record goes.

*Why use Mnth instead of Month for the calculated fieldname and, therefore, the section names?*

Notice in this report, we have group sections called 'Mnth Header' and 'Mnth Footer'. Why are they not called 'Month Header' and 'Month Footer'? It is important to avoid using reserved words (a link to Microsoft Access MVP Allen Browne's list to check for ' Problem names and reserved words' is listed in the Normalization section of this document).

The word "Month" has a special meaning to Access – **Month(***date***)** is a function that returns a month number, 1-12, from a date. It is a good idea to choose names that will not be confused with other things – and that is why Mnth has been spelled without the 'o' <smile>.

### Sorting and Grouping

This report has 3 levels of sorting and grouping as seen in the Sorting and Grouping window. It is grouped by the month number (even though the month name is what shows up on the report). Then, it is sorted by the day number and then fullname. A common misconception is that sort orders applied in a query, used as a RecordSource for a report, should apply. In fact, you must use the Sorting and Grouping feature to specify the desired sorting.

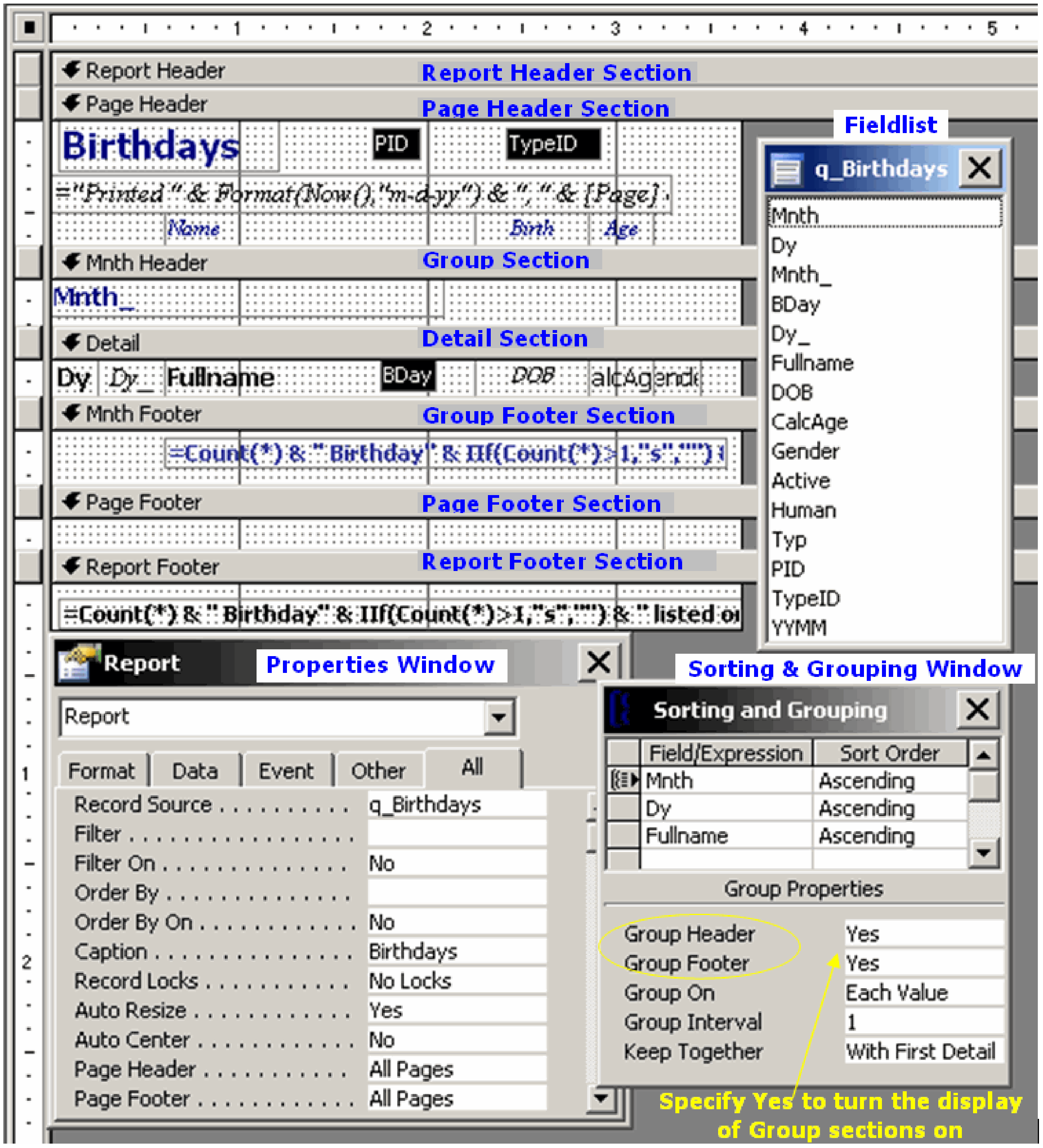
### Fieldlist

You also see the Fieldlist for the RecordSource, q_Birthdays.

### Properties

Also showing is the Properties window.

**Figure 2-14 Design View of a Report**



Report

| | |
|---|---|
| Record Source | q_Birthdays |
| Filter | |
| Filter On | No |
| Order By | |
| Order By On | No |
| Caption | Birthdays |
| Record Locks | No Locks |
| Auto Resize | Yes |
| Auto Center | No |
| Page Header | All Pages |
| Page Footer | All Pages |

| Field/Expression | Sort Order |
|---|---|
| Mnth | Ascending |
| Dy | Ascending |
| Fullname | Ascending |

Group Properties

| | |
|---|---|
| Group Header | Yes |
| Group Footer | Yes |
| Group On | Each Value |
| Group Interval | 1 |
| Keep Together | With First Detail |

## Macros

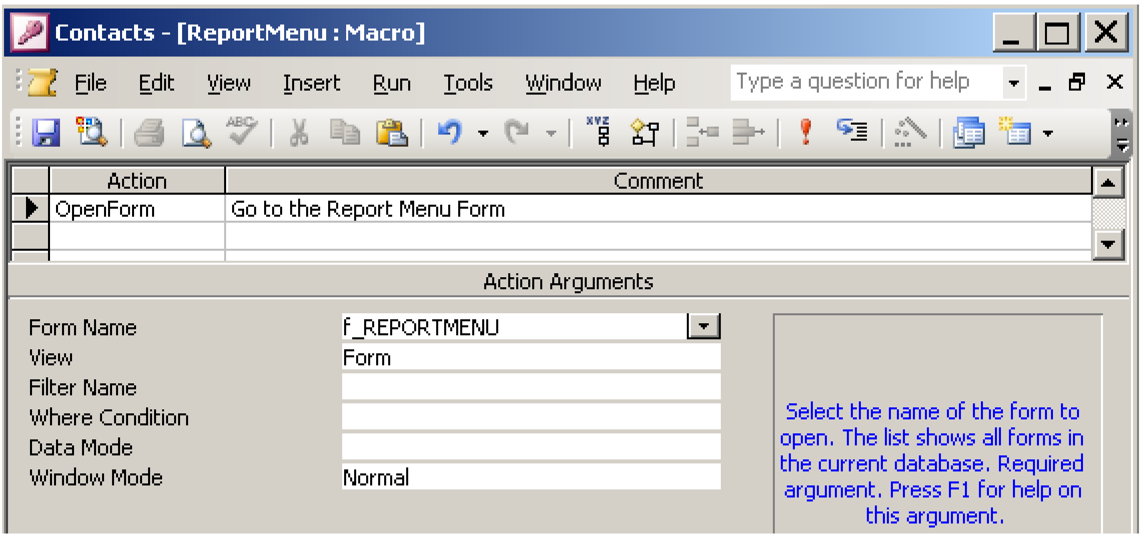
Macros provide an easy way to increase the functionality of your application without having to write code. Macros are quick and easy to create but are limited in what tasks they can perform, and errors that might occur are difficult to handle. Macros can be convenient but they do have tradeoffs.

A macro stores a list of actions to be performed. There are about 50 actions to choose from, many of which have parameters that can be specified. Traditionally, macros could not use variables or programming logic nor have error handlers. Starting with Access 2007, macros can use temporary variables (tempvars) and include error-handling. However, in all prior versions, there was no provision for tempvars or error handling. Not being able to use variables or trap for errors and present a graceful error message are a couple reasons that macros gained a bad reputation with many developers.

### *Design View of a Macro*

A typical macro action would be to open a particular form (OpenForm) when a command button is clicked. Notice that once OpenForm is specified as the Action, you are prompted for additional information in the lower pane such as the name of the form. Help is displayed on the right, depending on where your cursor is.

**Figure 2-15 Design View of a Macro**


Contacts - [ReportMenu : Macro]
File Edit View Insert Run Tools Window Help
Type a question for help
Action Arguments

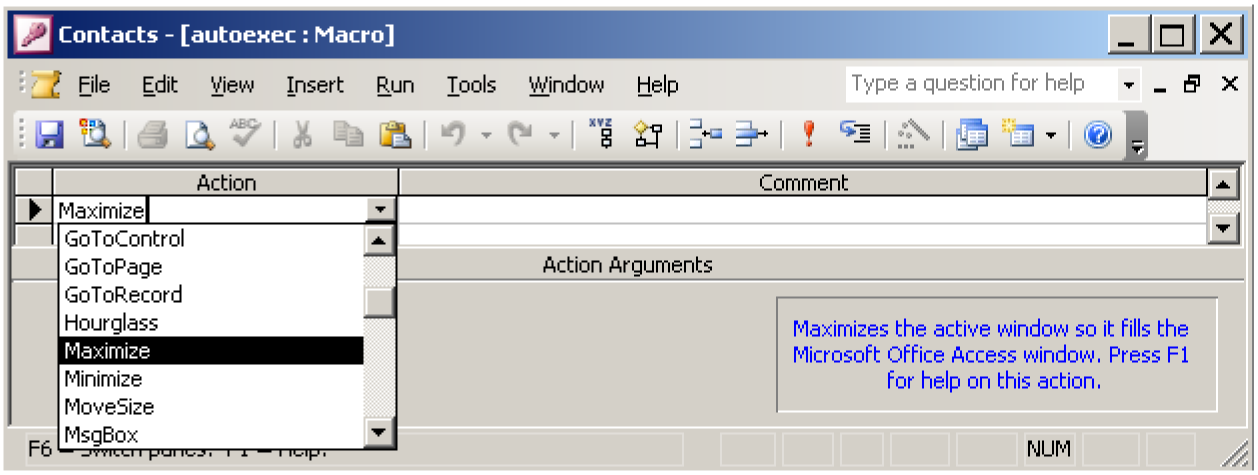
| Action | Comment |
|---|---|
| OpenForm | Go to the Report Menu Form |

| | |
|---|---|
| Form Name | f_REPORTMENU |
| View | Form |
| Filter Name | |
| Where Condition | |
| Data Mode | |
| Window Mode | Normal |

Select the name of the form to open. The list shows all forms in the current database. Required argument. Press F1 for help on this argument.

Figure 2-16 shows is a very simple macro that maximizes the windows within the Access application. Because it is named "**autoexec'**, it will be *automatically executed* when the database is opened. Notice that the Action column gives you a list of choices to pick from. In the lower pane on the right is a description of the chosen action.

**Figure 2-16 AutoExec Macro**

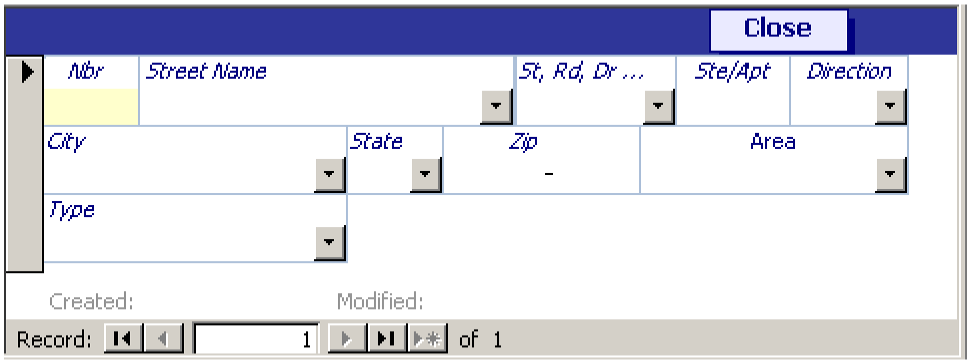


Maximizes the active window so it fills the Microsoft Office Access window. Press F1 for help on this action.

### *Macros for a Form*

In our database, we have a form to add an address as shown in Figure 2-17.

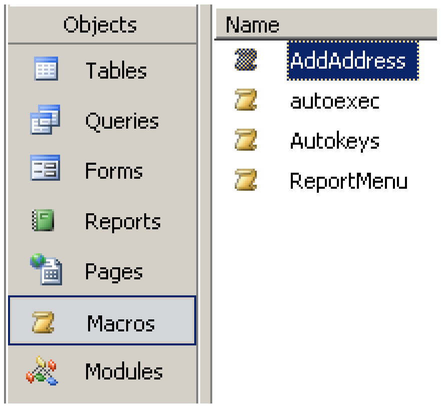
**Figure 2-17 Form to Add Address for Macro Example**



The user will type data into the form. Before the data is added or changed in the table that the form is based on, we will ask the user if they want to save changes. We will also want to let the user close the form.

To accomplish these two tasks, we will make a new macro group and call it <u>AddAddress</u> since it will be used for our AddAddress form. This is the name that will display in the Database Window.

**Figure 2-18 Macro Group Name shows in Database Window**



Our macro group will contain more than one macro so we will display the Macro Name column.

We also need to display the Condition column since we will ask the user a question and, based on their answer, we will choose to do something.

Here are the two macros we will create:

#### SaveChanges

The User will be given a message box with Yes and No buttons on the form BeforeUpdate event. If they choose No, two things will happen

1. *their changes will be undone*

```
Action = RunCommand
Command = Undo
```

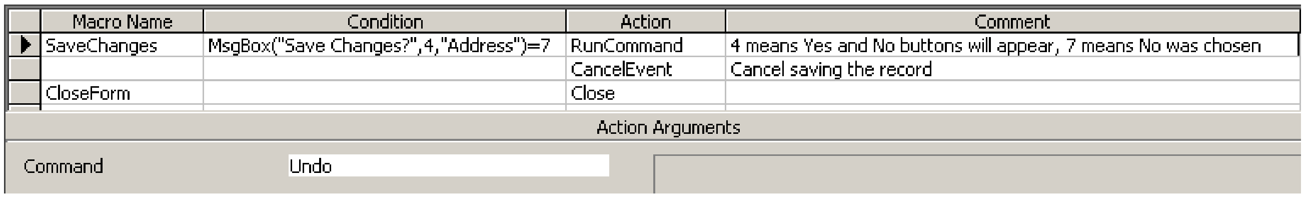
2. *the form BeforeUpdate event will be cancelled and their changes discarded*
```
Action = CancelEvent
```

**CloseForm**

We have a Close command button on the form and when the user clicks on it, the form will close.

```
Action = Close
```

**Figure 2-19 Design View of Macro showing Macro Name and Condition Columns**



| Macro Name | Condition | Action | Comment |
|---|---|---|---|
| SaveChanges | MsgBox("Save Changes?",4,"Address")=7 | RunCommand | 4 means Yes and No buttons will appear, 7 means No was chosen |
| | | CancelEvent | Cancel saving the record |
| CloseForm | | Close | |

Action Arguments

| | |
|---|---|
| Command | Undo |

### *Assigning macros to events*

When you Click in an event property and the Click on the drop-down arrow, you will see a list of macros in your database. If you wish to write code, you can choose [Event Procedure] , which is covered in the Database Objects section of this document.

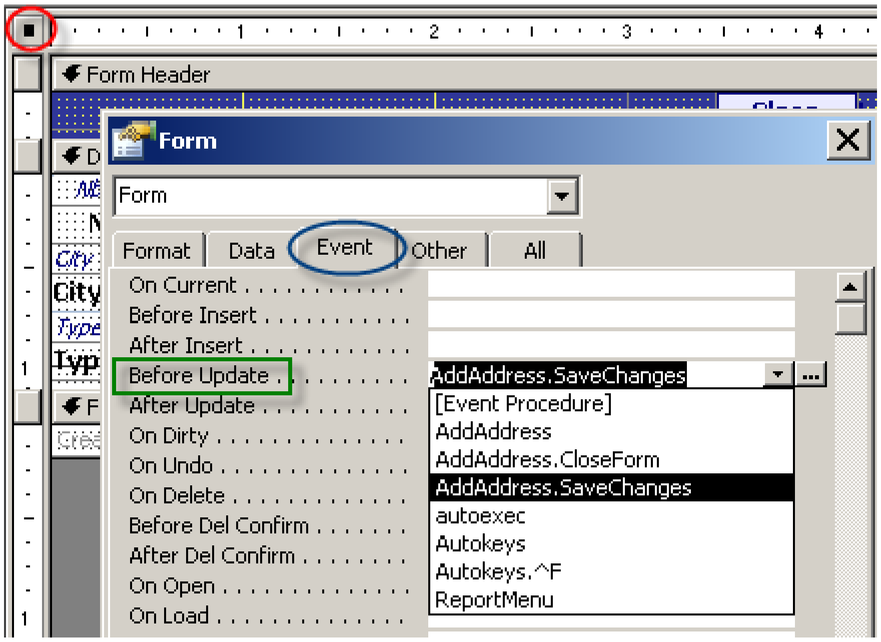
Click in the upper left where the rulers intersect to select the form.

Figure 2-20 shows a red circle around the black square in upper left indicating that the form is selected as opposed to an object on the form.

On the Property Sheet, choose the Event tab, as shown by the blue circle.

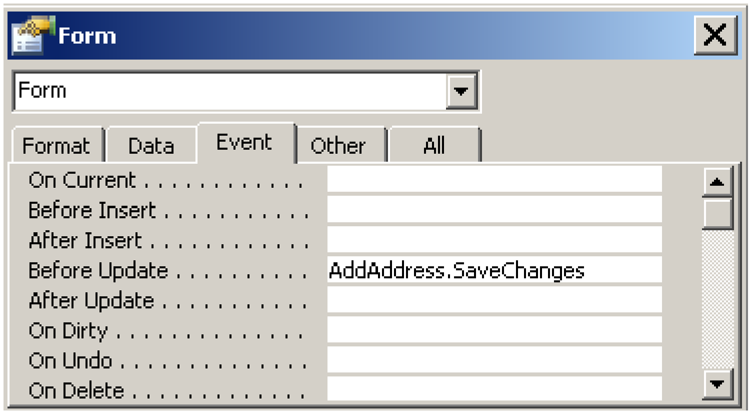
Click in the BeforeUpdate Event and choose the SaveChanges macro in the AddAddress macro group.

**Figure 2-20 Assign a Macro to the Form BeforeUpdate Event**



The **SaveChanges** macro from the **AddAddress** macro group is now assigned to the BeforeUpdate event of the form as shown in Figure 2-21.

**Figure 2-21 Property Sheet for Form showing Macro Assignment**



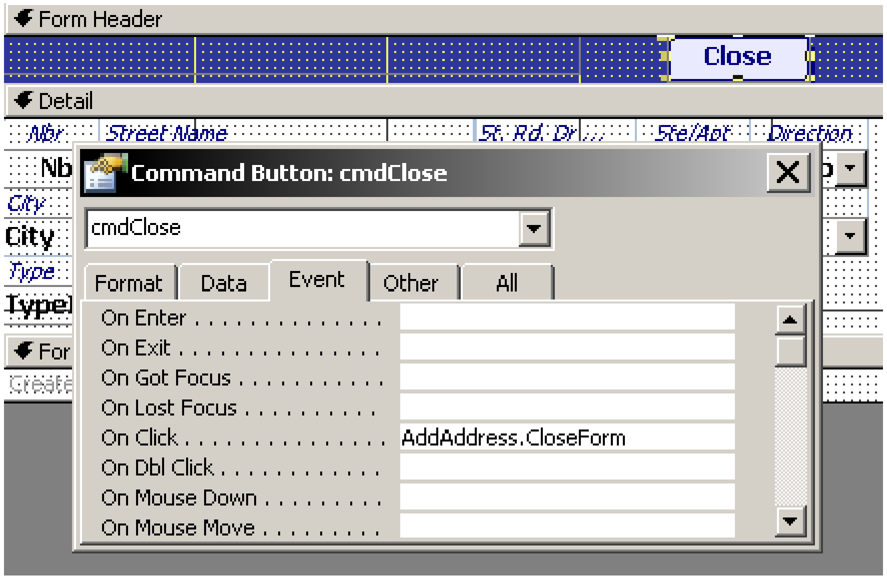
| | |
|---|---|
| On Current | |
| Before Insert | |
| After Insert | |
| Before Update | AddAddress.SaveChanges |
| After Update | |
| On Dirty | |
| On Undo | |
| On Delete | |

Next, we need to assign a macro to the Click event of our Close Command Button.

Click the command button to select it and then assign the CloseForm macro from the AddAddress macro group

The **CloseForm** macro from the **AddAddress** macro group will be assigned to the On Click event of our Close command button as shown in Figure 2-22.

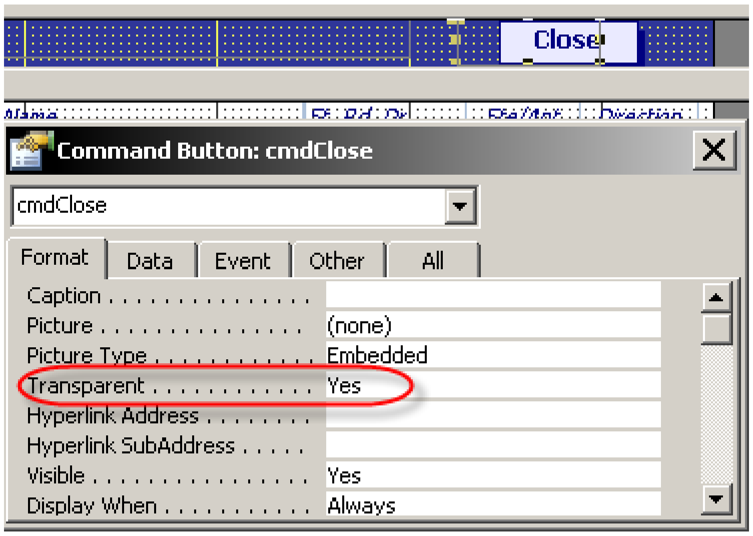
**Figure 2-22 Macro Assigned to Click Event of Command Button**



| | |
|---|---|
| On Enter | |
| On Exit | |
| On Got Focus | |
| On Lost Focus | |
| On Click | AddAddress.CloseForm |
| On Dbl Click | |
| On Mouse Down | |
| On Mouse Move | |

In case you are wondering how we got a command button colored blue … we didn't. A label control was used to make the blue 'Close' rectangle. Then a command button was laid on the top and its Transparent property was set to Yes. To demonstrate this better, the transparent command button has been moved slightly to the left as shown in Figure 2-23.

**Figure 2-23 Transparent Command Button is drawn on top of a Label Control**



| | |
|---|---|
| Caption | |
| Picture | (none) |
| Picture Type | Embedded |
| Transparent | Yes |
| Hyperlink Address | |
| Hyperlink SubAddress | |
| Visible | Yes |
| Display When | Always |

## *AutoKeys Macro*

An AutoKeys macro gives you a way to set up global shortcut keys for your application. In an AutoKeys macro, the Macro Name parameter is used to define the keystroke.

Key names are surrounded with curly braces.

```
{ESC}
{F12}
{ENTER}
```

`The carrot symbol ^ is used to mean` Ctrl

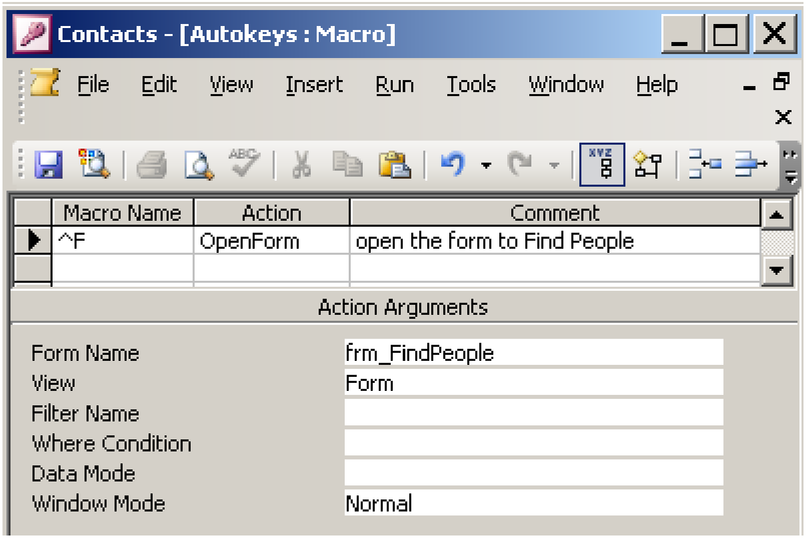
The Plus symbol + is used to mean Shift

You can look at the Help for SendKeys to find the other codes that can be used for AutoKeys -- except for the Alt key, which is not defined for use in an AutoKeys macro.

### *AutoKeys Example*

Figure 2-24 shows that we have defined Ctrl F to open the form to Find People.

**Figure 2-24 AutoKeys Macro showing how to set a Global Shortcut key**


Contacts - [Autokeys : Macro]
File Edit View Insert Run Tools Window Help

| Macro Name | Action | Comment |
|---|---|---|
| ^F | OpenForm | open the form to Find People |

Action Arguments

| | |
|---|---|
| Form Name | frm_FindPeople |
| View | Form |
| Filter Name | |
| Where Condition | |
| Data Mode | |
| Window Mode | Normal |

## Modules

Modules are used for storing Visual Basic for Applications (VBA) code (programs). Module code can do anything a macro can do (with the exception of AutoKeys) … and multitudes more! The ability to control Access using logic and loops is where the awesome power truly gets unleashed.

Unlike sequential programming, where a task is coded from input to report and cannot collect user input during this time, Access is *event*-driven … because the user clicked a button, or changed a record, or opened a form.

Code procedures are often short and respond to something that the user has requested or done such as clicked on a button or modified data in a control. Code can also be set on a timer or launched by other applications.

Code that is stored with a form or report is called a Class Module. Code that is stored in an independent module sheet is a Standard (or General) Module.

### *What is Me?*

When you are in the code behind a form or report, Me refers to the form or report itself. Me is not necessary unless there is a conflict with names – using it makes things more clear. If you have Me.Total in code, you know it is referring to a control or a field named Total. If you just use Total, it could also mean a variable.

Another advantage of using Me is this: when you type **Me.** in code, Access *IntelliSense* will prompt you with a list of choices so it makes coding easier and faster. Let us assume we have a command button to close the form when the user clicks on it. The name of the command button is 'cmdClose' and this would be the Click event.

```
Private Sub cmdClose_Click()

  '--------------- ignore Errors

  On Error Resume Next

  '--------------- save record before closing form
  ' Add Note: Do this test for bound forms (forms with a record source) only.

  'test to see if record needs to be saved

  If Me.Dirty = True Then

       Me.Dirty = False

  End If

  '--------------- if record is still 'dirty' then it was not saved
  ' so don't close

  If Me.Dirty = True Then

       Exit Sub

  End If

  '--------------- close the form and do not save design changes
  DoCmd.Close acForm, Me.name, acSaveNo

End Sub
```

After you choose Close from the list of actions, you are prompted to specify other information, called arguments. Arguments are separated by commas, and can be positional or named arguments. Positional arguments must be specified in the correct order; named arguments can be given in any order, and without having to include more than one comma in a row. This example used positional arguments.

The first argument is the type of object. acForm is a built-in constant that evaluates to 2, which means the object is a form.

Next, Access wants to know the name of the object. Me.name is used to indicate the name of the form that the code is attached to.

*Note:* Instead of using Me, you could specify the name of the form as a string, however, doing so means that your code is less portable. If your form is a bound form (i.e. includes a RecordSource), you should save the record before closing the form. If the 'Dirty' property of the form is true, that means the current record has changes that are not saved. The reason that 'Dirty' is tested twice is because, after the first test, if the record still has unsaved changes then it did not pass validation and the user still needs to take action so the closing of the form should not happen. This will help prevent "losing" data, as discussed in this excellent tip by Access MVP Allen Browne:

**Losing data when you close a form**
http://allenbrowne.com/bug-01.html

Whenever you close a form that has been opened for the user to edit data, it is best to instruct Access not to save the form (you do not want to replace the version of the form that you, as the developer, created and saved), which is why the constant, acSaveNo, is used and specifies that the form's DESIGN should not be saved. This (acSaveNo) should always be specified to avoid problems; otherwise Access automatically saves property settings such as the Filter property.

### *Testing for Conditions in Code*

Take a breath … we are going to look at more code ….

Many of you have probably heard the term "If-Statement". Life is full of choices. If one thing happens, it will cause another to happen. Simply put: **If** this, **then** that.

In terms of VBA, an IF statement tests a condition and, depending on the outcome, does one thing or another – or nothing. A condition is a numeric or string expression that evaluates to **TRUE** or **FALSE**.

The order that words have to be written is called *syntax*. Optional parts are shown in brackets.

```
If condition Then
      [statements]
[ElseIf condition-n Then
      [elseif_statements]]
[Else
      [else_statements]]
End If
```

*Example:*

```
If Me.ST = 'TX' Then
      Msgbox "Texas is a great state … but it sure gets hot!"
Else
      Msgbox "If you move to Texas, make sure you have a swimming pool!"
End If
```

WHERE:
ST is the Name of a control containing the State Abbreviation on the form this code is behind
`Msgbox` is a function built-in to Access to display a message.

### *Design View of a Module*

Figure 2-25 shows the Visual Basic Editor (VBE) and is where you create and edit module code. The name of the module we are looking at, bas_crystal_code_general_071020, is highlighted in gray in the Project Explorer Window shown in the upper left. When you Double-Click on a module in the Project Explorer Window, its code will be shown on the right.

In the lower left, the Properties window is displayed and shows the Name property of the module.

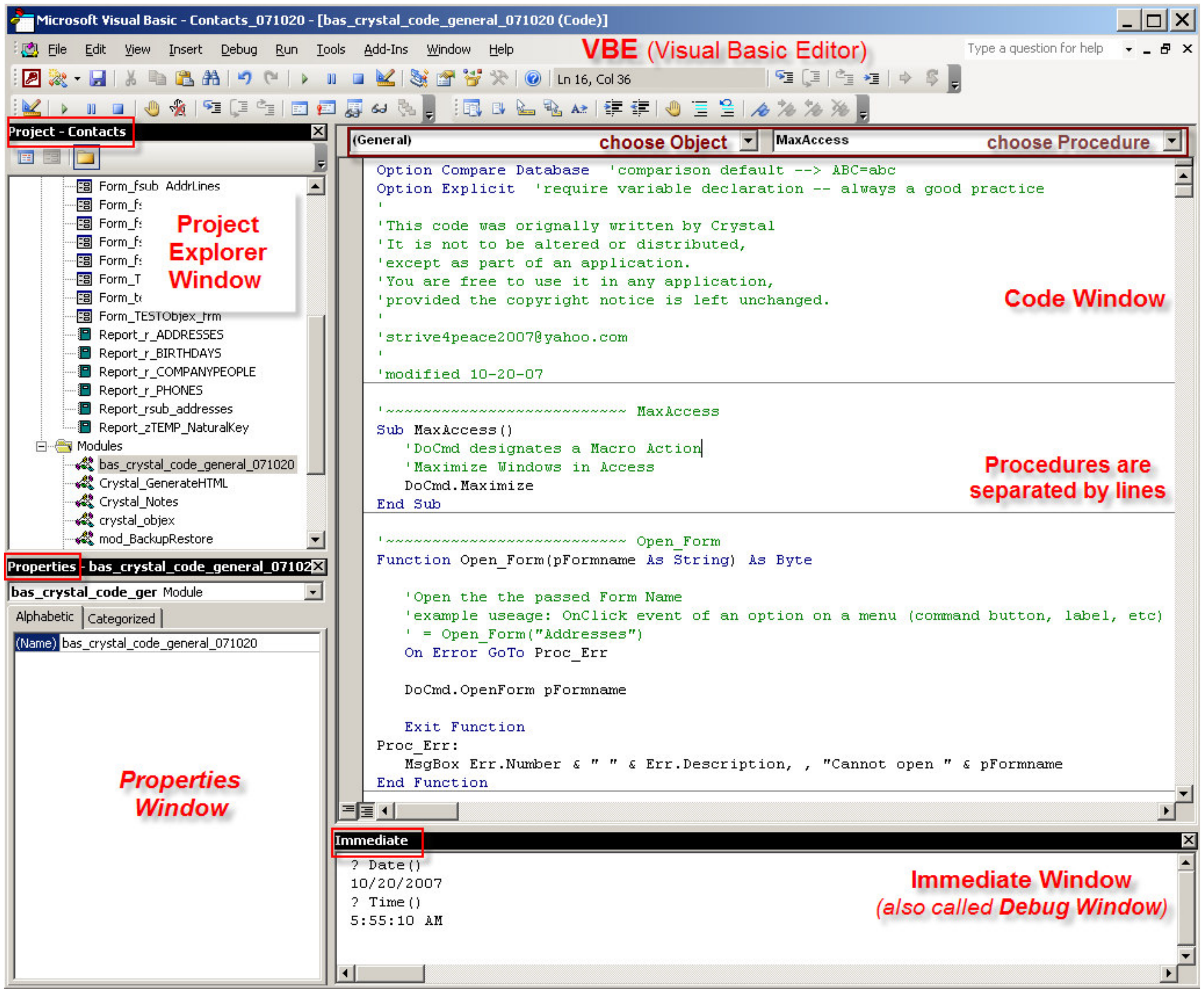
On the bottom right is the Immediate Window, formerly known as, and often still called, the Debug window. You can type a question mark followed by a function or sub and Access will execute it.

On the right is the code window. Remember the macro we just showed you to Maximize Access? The top routine does the same thing using VBA code – so don't be intimidated by code! Its fun! (geek bell is ringing loudly).

In code, comments are shown in green. These lines are added to describe what you are doing and are not part of the executable program code. Some of the keywords are shown in navy, and the rest of the program code is displayed in black.

The Object and Procedure comboboxes at the top give you a quick way to jump to specific places in code without scrolling through the module sheet.

**Figure 2-25 VBE (Visual Basic Editor) – Design View of a Module**


Microsoft Visual Basic - Contacts_071020 - [bas_crystal_code_general_071020 (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
VBE (Visual Basic Editor)
Type a question for help
Ln 16, Col 36
Project - Contacts
Project Explorer Window
Form_fsub AddrLines
Form_TESTObjex_frm
Report_r_ADDRESSES
Report_r_BIRTHDAYS
Report_r_COMPANYPEOPLE
Report_r_PHONES
Report_rsub_addresses
Report_zTEMP_NaturalKey
Modules
bas_crystal_code_general_071020
Crystal_GenerateHTML
Crystal_Notes
crystal_objex
mod_BackupRestore
Properties - bas_crystal_code_general_07102
bas_crystal_code_ger Module
Alphabetic Categorized
(Name) bas_crystal_code_general_071020
Properties Window
(General) choose Object
MaxAccess choose Procedure
Code Window
Procedures are separated by lines
Immediate
Immediate Window (also called Debug Window)

```
Option Compare Database  'comparison default --> ABC=abc
Option Explicit  'require variable declaration -- always a good practice
'
'This code was orignally written by Crystal
'It is not to be altered or distributed,
'except as part of an application.
'You are free to use it in any application,
'provided the copyright notice is left unchanged.
'
'strive4peace2007@yahoo.com
'
'modified 10-20-07

'~~~~~~~~~~~~~~~~~~~~~~~~~~ MaxAccess
Sub MaxAccess()
   'DoCmd designates a Macro Action
   'Maximize Windows in Access
   DoCmd.Maximize
End Sub

'~~~~~~~~~~~~~~~~~~~~~~~~~~ Open_Form
Function Open_Form(pFormname As String) As Byte

   'Open the the passed Form Name
   'example useage: OnClick event of an option on a menu (command button, label, etc)
   ' = Open_Form("Addresses")
   On Error GoTo Proc_Err

   DoCmd.OpenForm pFormname

   Exit Function
Proc_Err:
   MsgBox Err.Number & " " & Err.Description, , "Cannot open " & pFormname
End Function
```

```
? Date()
10/20/2007
? Time()
5:55:10 AM
```

You can use the Page Up and Page Down buttons to move up or down one screen at a time

You can press Ctrl Page Up or Ctrl ↑ to move up one procedure at a time

You can press Ctrl Page Down or Ctrl ↓ to move down one procedure at a time

Ctrl Home moves the cursor to the top of the module

Ctrl End moves the cursor to the end of the module

Home moves the cursor to the beginning of a line

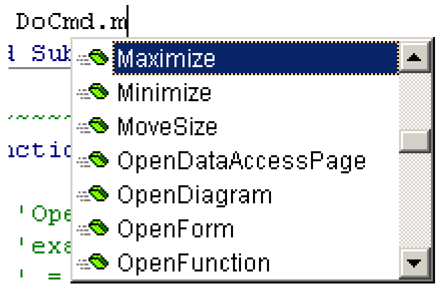
End moves the cursor to the end of a line

Indenting code is always a good idea. Press Tab to increase the indent or Shift Tab to decrease the indent. If you select one or more lines and press Tab or Shift Tab, you will increase or decrese the indent for all the lines selected.

## IntelliSense

When you are typing code, starting with version 97, Access has really nice feature called *IntelliSense* that helps you write code by prompting you with choices. In this example, once DoCmd. is typed, Access shows you a list of macro actions (choices for DoCmd). As you type letters, the list index moves. So it IS easy to write code ☺

**Figure 2-26 IntelliSense helps Write Code**



```
DoCmd.m
```

## *Objects, Containers, Collections*

Everything tangible in Access is an *object* whether it is a table, query, form, field, or control. Objects are stored in *containers* and many objects of the same type are called a *collection*.

### Properties and Methods

- Properties are like adjectives that describe an object.
- Methods are like verbs and define actions an object can do

For humans, properties include height, weight, eye color, hair color, and gender.

Human methods would be actions such as run, jump, eat, and sleep.

Table properties include name, date created, and a collection of fields. Field properties include the name, data type, size, and description.

Table actions include: add, delete, rename, and open.

### Events

Events occur when the user does something such as Click on a form control, or change data in a form. When events happen, code can be launched.

Events that can be trapped at the form level include Open, Load (populate with data), Current (change which record is displayed), BeforeInsert (just before a record is added), AfterInsert (after a record is added), BeforeUpdate (before changes to a record are written), and AfterUpdate (after changes are written). There is a Timer event and you can specify the Time Interval to launch code as long as a form is open – be careful about using a timer, though, it can lead to strange problems.

Control events include Click, DoubleClick, BeforeUpdate, AfterUpdate, GotFocus, LostFocus, and MouseUp.

## *VBA*

VBA (Visual Basic for Applications) is based on a simple set of logic control. Essential statements include testing conditions to determine which code to execute (IF, THEN, ELSE) and using loops to repeat tasks (DO WHILE, LOOP). Think in terms of discrete tasks that can be launched at the click of a button.

Code acts on objects and is triggered by events. Understanding the types of objects that are available and knowing the properties (describe an object, like adjectives describe nouns) and methods (actions that object can do, like verbs are for nouns) is paramount to writing VBA code.

Most of what you will learn as you dive into the exciting world of programming is that VBA is *logical* – it makes sense!… just as you would drive a car but not a piece of fruit; and eat fruit but not a car.

## *Reference Libraries*

Much as a set of encyclopedias is a resource to look up information, reference libraries provide program code with additional information.

### VBA

The VBA reference library includes generic functions (not specific to an application) such as:

*math*

**Round** numbers. Round(23.473,1) → 23.5
**Abs** – get absolute value of a number. Abs(-5) → 5
**Sgn** – get sign of a number. Sgn(-5) → -1
**Rnd** – Random Number. CLng(Rnd(Now())*100) → {Some random integer between 0 to 100}

*manipulate strings*

**Left** # letters. Left("Access Database", 6) → "Access"
**InStr** – determine position of a character or phrase. Instr("Access Database", " ") → 7
**Mid** – extract a sub string. Mid("Smith, Mary",8, 4) → "Mary"
**Len** – get length of a string. Len("Smith, Mary") → 11

*access the file system*

**Dir** – retrieve first filename from a particular location. FileNam = DIR("c:\data\*.*)
**Name** – change filename and/or path. Name "c:\data\MyFile.doc" AS "c:\data\Myfiles\NewName.doc"
**FileCopy** – make a copy of a file. FileCopy "c:\data\MyFile.xls", "c:\data\CopyOfMyFile.xls"
**GetAttr**ibutes of a spec such as Normal, Read-Only, Hidden, Directory. GetAttr("c:\data") → 16
16 is the constant indicating a directory
**FileDateTime** – date file last changed. FileDateTime("c:\data\AccessBasics.doc") → 8/26/2007

*convert (between data types – these are very important)* *--commonly known as "Type Conversion" functions*

**CSng**, **CDbl**, **CCur**, **CInt**, **CLng**, **CDate**, **CStr**, etc.
to long integer: CLng(1005.672) → 1006
to date: CDate(#8/26/07#) + 10 → 9/5/2007
**Format** to change numbers* into strings. Format(Date(), "dd-mmm-yy") → 26-Aug-07
to change strings. Format(" I want to be capital", ">") → I WANT TO BE CAPITAL
** Dates are stores internally as numbers*

*Get and Set Dates and Times*

get current **Date** and **Time** from system clock. Date() → 11/21/2007, Time() → 4:48:32 PM, Now() → 11/21/2007 4:49:04 PM
extract the **Day**, **Month**, or **Year** from a date. Year( Date() ) → 2007
**DateSerial** to create a date from year, month, day. DateSerial(2007, 8+1, 26) → 9/26/2007
extract the **Hour**, **Minute**, or **Second** from a date/time. Minute(#8:30#) → 30
**TimeSerial** to create a time from hour, minute, second. TimeSerial(16, 30, 0) → 4:30:00 PM
**DatePart** to extract intervals such as quarter, weekday, day of month. DatePart("q",#1/1/2010#) → 1
**DateAdd** to add/subtract specified intervals to/from date. DateAdd("m", 3, #8/26/07#) → 11/26/2007

**DateDiff** for difference between dates. DateDiff("q",#1/1/07#, #12/5/07#) → 3
**DateValue** to extract a date. DateValue("Apr 3, 2007 8:30") → 4/3/2007
**TimeValue** to extract a time. TimeValue("Apr 3, 2007 8:30") → 8:30:00 AM

There are functions for Financial calculations such as amortizing payments, Constants, Informational functions to check data type or errors, and much more. The VBA library is listed at the top of the order and cannot be moved because it is the first external library accessed to interpret code.

## Access

The objects in Access (tables, queries, forms, reports, macros, modules, etc) are defined in the Microsoft Access #.0 Object Library, where #.0 indicates the version number. The Access library is listed second and, like the VBA library, cannot be removed.

## ActiveX Data Objects

Provides the programmatic interface (i.e.: the VBA object types, methods and functions) which allow you to manipulate data using the ADO protocol. ADO database objects appear in the object browser as the ADODB library and is defined in the Microsoft ActiveX Data Objects #.# Library. ADO communicates with a variety of data sources through "OLE DB Providers". OLE DB Providers goes beyond the capabilities of ODBC (Open DataBase Connectivity). In simpler terms, ADO makes data stored in other formats outside of Access accessible, similar to DAO, but a broader range of sources can be communicated with.

## OLE

The ability to do things such as enable conversations with other applications is defined in the OLE (Object Linking and Embedding) Automation reference library, which uses COM (Component Object Model) technology.

## DAO

DAO stands for Data Access Objects. Using a Microsoft DAO library allows you to do things such as find, view and manipulate data and definitions in Databases, Tables (TableDefs), Queries (QueryDefs); to loop through objects a container holds (such as records of a table); and retrieve/put data from/into external files such as other databases. You can also do many of the same operations using ADO. There is some overlap between DAO and ADO but the methods they use are different.

The DAO Library was built-in to Access 97. Then, in versions 2000 and 2002, it had to be added by the user. Microsoft realized it was best to include it automatically and in version 2003 and above, DAO is again a default library.

## Excel

To get data from/ put data into Excel, and control the Excel objects, you can specify the Microsoft Excel #.# Object Library, which is not included by default. This is an example of using Early Binding. However, it is best to convert early bound automation code to use Late Binding (no checked library reference required), after you have finished writing and debugging your code.

For more information on binding:

**Late Binding in Microsoft Access, by Tony Toews**
http://www.granite.ab.ca/access/latebinding.htm

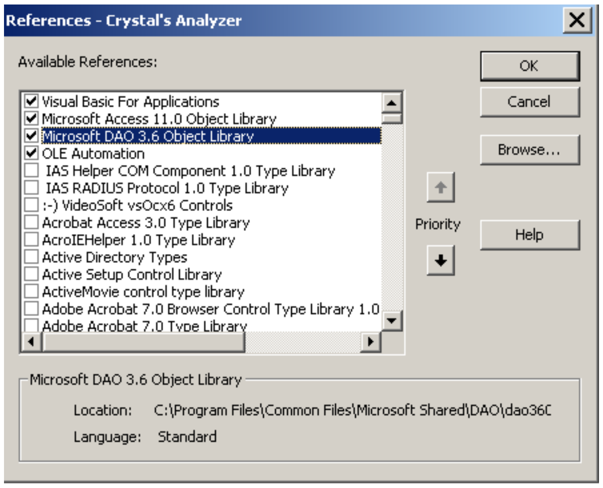
**Dick's Clicks, Early Binding vs. Late Binding, by Dick Kusleika**
http://www.dicks-clicks.com/excel/olBinding.htm

**Using early binding and late binding in Automation, Microsoft**
http://support.microsoft.com/kb/245115

## Other libraries

There is a multitude of other libraries available in case you want to write code to include use of them such as MS Graph, HTML, Outlook, PowerPoint, Word, XML, and Scripting (the file system). If you are using the Object Browser to explore capabilities, you might *temporarily* reference another library to learn more about it. Be careful about keeping references to libraries that are not necessary. It is okay to reference one so that you can use the Object Browser to see what is in it, but then remove it if your code does not need it. The Object Browser is discussed in the **Properties and Methods** section of this document.

A broken reference will cause strange errors when you run an application. To "fix" this, you can often remove one of the libraries, then put it back. If any references are flagged as MISSING, uncheck it, click on the OK button to dismiss the References dialog. Compile your code. If it compiles okay without the reference, then it is not needed. If you do need the reference, re-open this dialog and add the removed reference back in. If it does not appear in the list, use the Browse button to locate the file.


References - Crystal's Analyzer

Available References:

- [✓] Visual Basic For Applications
- [✓] Microsoft Access 11.0 Object Library
- [✓] Microsoft DAO 3.6 Object Library
- [✓] OLE Automation
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] :-) VideoSoft vsOcx6 Controls
- [ ] Acrobat Access 3.0 Type Library
- [ ] AcroIEHelper 1.0 Type Library
- [ ] Active Directory Types
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library
- [ ] Adobe Acrobat 7.0 Browser Control Type Library 1.0
- [ ] Adobe Acrobat 7.0 Type Library

Microsoft DAO 3.6 Object Library

Location: C:\Program Files\Common Files\Microsoft Shared\DAO\dao360

Language: Standard

*Note: You should not need to exit the database to fix a broken reference – but sometimes that is what it takes.*

To see library references, from a module window, use this menu option → Tools, ReferencesHere are some articles on library references:

**Solving Problems with Library References, by Allen Browne**
http://allenbrowne.com/ser-38.html

**Access Reference Problems, by Doug Steele**
http://www.accessmvp.com/djsteele/AccessReferenceErrors.html

Another important consideration when referencing libraries is their order, or priority. Here is an article on Reference Priority.

**ADO and DAO Library References in Access Databases, by Tom Wickerath**
http://www.access.qbuilt.com/html/ado_and_dao.html