

6. SQL Syntax

Don't let the SQL acronym intimidate you; it is not tough to learn the basics.

SQL background

SQL is Structured Query Language

It is a statement to get data out of one or more tables/queries. This is what Access stores for:

1. Queries (just shows the QBE grid for convenience -- choose View, SQL)
2. RowSource for Comboboxes and Listboxes (if list does not come directly from a table since a query is actually an SQL statement)
3. RecordSource for Reports and Forms (if list does not come directly from a table)

Queries can be different types, for instance:

1. Select
2. Crosstab
3. Make-Table
4. Update
5. Append
6. Delete

Select and Crosstab queries show data while the others are "action" queries" (Data Manipulation Queries) and do not *display* data -- so you cannot use them for record sources or for row sources.

To expand and reiterate:

Queries are a very powerful tool in Access -- you can do far more than simply use them to select data

... Queries can be ACTION Queries...*they DO things*, not show you things

They can add data to a table --> APPEND Query

They can make changes to a table --> UPDATE Query

They can delete records from a table --> Delete Query

They can make tables --> Make-Table Query

They can transpose data --> Crosstab Query (which is also, actually, SELECTing data from a table since there is not action)

Internally, Access stores an SQL (Structured Query Language) statement for queries*, not the visual representation you see when you design queries -- the QBE (Query-By-Example) grid is for your benefit, so you can use drag-n-drop and pick methods and visually look at things better.

*this is why it is important to get comfortable with looking at SQL statements

A great way to become more familiar with SQL is to LOOK at the SQL every time you make a query. Until you actually start to look, you never realize how easy and sensible it actually is... and it really helps to use Aliases (short abbreviations) for tablename as it makes the SQL statement shorter.

Types of Queries

~~~ SELECT ~~~

Select data from one or more tables and display the results in rows and columns.

BASIC SQL SYNTAX

SELECT fieldlist
FROM tablename
IN anotherdatabase.mdb
WHERE conditions
GROUP BY fieldlist
HAVING conditions for fields that are grouped
ORDER BY fieldlist;

~~~ APPEND ~~~

Add records to a table.

An **Append Query** is a **Select** query preceded by

INSERT INTO TableName (field1, field2, etc)

'~ for instance (and this does not use a table as it supplies actual values)

```
Dim strSQL as string
strSQL = "INSERT INTO [TableName] " _
        & " ([TextField], [NumField], [DateField] ) " _
        & " SELECT '" & strValue & "', " _
        & numValue & ", " _
        & "#" & datValue & "#" _
        & ";"
CurrentDb.Execute strSQL, dbFailOnError
CurrentDb.TableDefs.Refresh
```

~~~ UPDATE ~~~

Update records in a table.

An Update Query first identifies the tables that are used

UPDATE table1
INNER JOIN table2 **ON** table1.keyfield = table2.keyfield

Then identifies what to change

SET table1.fieldtochange = expression

Then, if you have criteria...

WHERE tablename.strField = 'somevalue'
AND tablename.numField = 99
AND tablename.dateField = #1/1/06#

~~~ MAKE TABLE ~~~

Make a table from the resulting values.

```
SELECT fieldlist
INTO tablename
IN c:\path\anotherdatabase.mdb
FROM tablename
WHERE conditions to process before recordset is made
GROUP BY fieldlist
HAVING conditions for fields that are grouped
ORDER BY fieldlist;
```

~~~ DELETE ~~~

Delete records in a table.

```
DELETE A.*
FROM tblArticles AS A
WHERE (A.somefield=2);
```

~~~ CROSSTAB ~~~

Instead of displaying values for a field in different rows, display them across the top – like display months of a year from a date field.

```
TRANSFORM Count(B.Fieldname1) AS FieldAlias
SELECT
  A.Fieldname2,
  A.Fieldname3
FROM Table2 AS B
  INNER JOIN Table1 AS A
  ON B.someID = A.someID
GROUP BY
  A.Fieldname2,
  A.Fieldname3
PIVOT B.Fieldname1;
```

you can use equations to pivot (this will be column headings). For instance, if you want the column headings to be year and month, you can do this:

```
PIVOT Format([DateField], 'yy-mm');
```

if you want month names in chronological, instead of alphabetical, order, you can do this:

```
PIVOT Format([DateField], "mmm") In
("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
```

The **PIVOT** clause can also be stated as **PIVOT BY**

If you also want a column that totals whatever you have in all the **VALUE** columns (the expression after **TRANSFORM**), repeat your transform expression after the **SELECT** keyword and the **GROUP BY** keywords. For instance:

```
TRANSFORM Count(B.Fieldname1) AS FieldAlias
SELECT
  A.Fieldname2,
  A.Fieldname,
  Count(B.Fieldname1) AS FieldAlias2
FROM Table2 AS B
  INNER JOIN Table1 AS A
  ON B.someID = A.someID
GROUP BY
  A.Fieldname2,
  A.Fieldname3,
  Count(B.Fieldname1)
PIVOT B.Fieldname1;
```

'~~~~~'

here is an example:

Sales

DateSale, date
ItemID, long integer
Qty, integer

Items

ItemID, autonumber
Item, text

SQL:

'~~~~~'

```
TRANSFORM Sum(Qty) AS SumQty
SELECT
  Item,
  Sum(Qty) AS TotalQty
FROM Sales
  INNER JOIN Items
  ON Sales.ItemID = Items.ItemID
GROUP BY
  Item,
  Sum(Qty)
PIVOT By Format(DateSale,"yy-mm_mmm");
```

'~~~~~'

RESULTS:

'~~~~~'

Item	TotalQty	06-01_Jan	06-02_Feb	06-03_Mar
Bells	15	5	6	4
Whistles	9	2	7	

'~~~~~'

Data Definition Queries

There is another class of SQL statement, called Data Definition Queries, that can be used to create, modify or delete tables, fields, indexes and relationships. We will not go into examples here.

Aliases

Using Aliases for tablename (/queryname) makes the SQL easier to read.

For calculated fields, it is best to assign your own field alias instead of letting Access use "expr1", "expr2", etc. Calculated fields MUST have aliases.

An Alias follows the keyword AS

Table Alias:

```
FROM t_People As P INNER JOIN t_Phones As Pho ON P.PID = Pho.PID
```

Field Alias:

```
SELECT Sale-Cost As Profit, etc...
```

Joins

When you are getting information from more than one table, the default join type is INNER JOIN. This means that only records in common will be displayed. For instance, if you have a table of Companies and a table of Jobs and not every Company has done a job, but you want to see ALL the companies anyway, you would use a Left Join or a Right Join for the type of relationship between tables.

```
FROM Companies AS C
```

```
LEFT JOIN Jobs AS J ON C.CompID = J.CompID
```

The join is specified as a **LEFT JOIN** because you want all records from Companies and any matching records from Jobs. [The On clause can have the tables in either order.]

In this case, since C (Companies) is on the left side of the equal sign, the Join is a Left Join.

```
ON C.CompID = J.CompID --> C is on the LEFT side of the equal sign.
```

Parameters

If you specify data type in Query, Parameters, the SQL statement is preceded by (for example):

```
PARAMETERS [enter category] Text ( 50 );
```

while the criteria may be:

```
WHERE (B.Category=[enter category])
```

More information on SQL

For more information on SQL, one of the masters is John Viescas and he has written numerous books, all of which are fantastic. One of his books is titled, "*SQL Queries for Mere Mortals*"

Look At SQL!

Whenever you create a query, LOOK at the SQL statement and study it for a couple minutes -- it makes sense! Within a short period, the confusion will be gone...

from the menu of a query, choose:

View, SQL

The SQL statement can be selected, copied, then pasted into Word for formatting and printing (makes great wallpaper for your wall, not your computer ;) as you are learning) or into Notepad to have as a popup reference while you are working into design view, etc.

First, get comfortable with **SELECT** statements. Once you have them mastered, learn other forms.

Here is an example of the Datasheet View of a Phone List query:

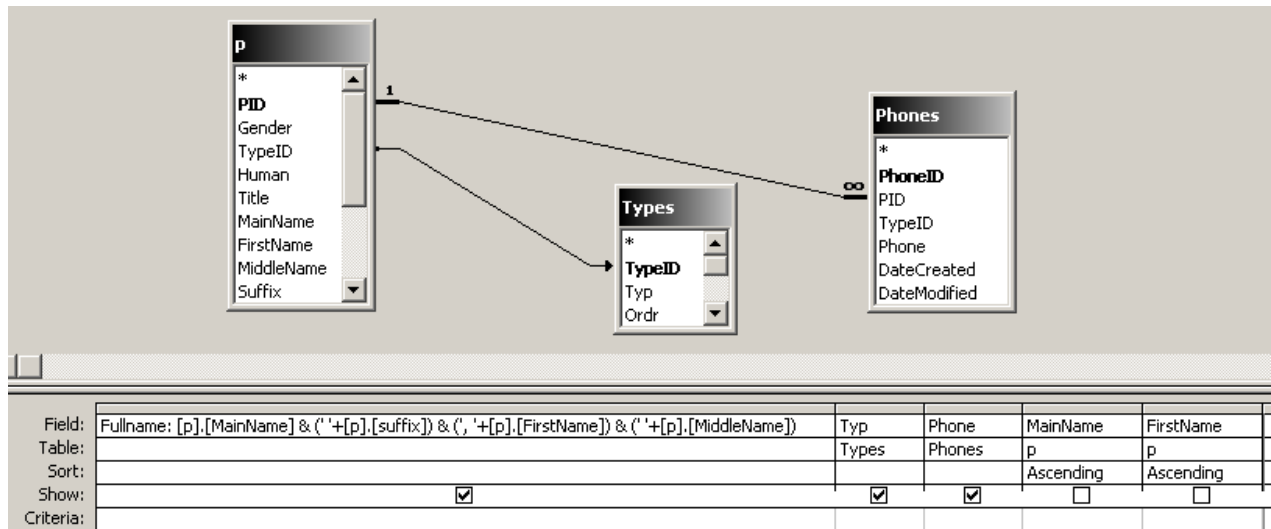
Figure 6-1 Datasheet View of a Phone List Query

	Fullname	Typ	Phone
▶	A & M Surplus	Store	(123) 456-7890
	Ackerman, Ian T.	Friend	(123) 456-7890
	Ackley, Cynthia T.	Friend	(678) 969-0845
	Ackley, Cynthia T.	Friend	(678) 732-6511
	Ackley, Steven	Friend	(42) 514-9794
	All Natural	Health Store	(555) 545-4545
	American School of Beauty	Beauty	(333) 489-7845
	Anderson, Heather E.	Friend	(770) 637-4055
	Anderson, Heather E.	Friend	(678) 937-3575
	Anderson, Walt	Friend	(770) 128-7468
	Anderson, Walt	Friend	(678) 994-3611
	Andy's Alarm systems	Home Suppl/Serv	(456) 871-6512
	Angelo's Floor Covering	Home Suppl/Serv	(456) 074-2131
	Applegate Ace Hardware	Bldg Supplies	(123) 456-7892
	Astrid's Upholstry	Services	(325) 635-4877
	Backseat Painting	Services	(456) 056-7232
	Baker, Debra T.	Friend	(678) 103-0103
	Baker, Debra T.	Friend	(770) 851-9069
	Baker, Richard	Friend	(404) 710-4585
	Baker, Richard	Friend	(404) 594-3515
	Barker Tires	Auto	(130) 564-7974
	Barnett Auto Body	Auto	(415) 610-6543
	Barricks, Cecilla F.	Friend	(90) 667-7332
	Barricks, Cecilla F.	Friend	(803) 106-8148
	Barricks, Jonathan	Friend	(912) 646-3705

Record: 1 of 209

... and the Design View of the same query:

Figure 6-2 Design View of a Phone List Query



... and the SQL View:

Figure 6-3 SQL View of a Phone List Query

```

SELECT [p].[MainName] & ('+[p].[suffix]) & ('+[p].[FirstName]) & ('+[p].[MiddleName]) AS Fullname
, Types.Typ
, Phones.Phone
FROM (t_PEOPLE AS p LEFT JOIN t_Types AS Types ON p.TypeID = Types.TypeID)
INNER JOIN t_Phones AS Phones ON p.PID = Phones.PID
ORDER BY p.MainName, p.FirstName;

```

Boldfacing the keywords and adding a few more line breaks gives this:

```

SELECT
    p.MainName & (' '+p.suffix) & (' '+p.FirstName) & (' '+p.MiddleName) AS Fullname
    , Types.Typ
    , Phones.Phone
FROM (t_PEOPLE AS p
    LEFT JOIN t_Types AS Types
        ON p.TypeID = Types.TypeID)
    INNER JOIN t_Phones AS Phones
        ON p.PID = Phones.PID
ORDER BY p.MainName
    , p.FirstName;

```

An Alias of 'p' has been used for the People table, which makes the SQL easier to read. The Alias is in this phrase:

t_PEOPLE AS p

Can you spot another Alias?

There are actually 4 Aliases altogether, 3 for tables, and one for a calculated field.

Using VBA to execute SQL statements

You can execute action queries using VBA

```
'~~~~~
dim strSQL as string
strSQL = "UPDATE tablename " _
& " SET fieldname = value " _
& " WHERE conditions;"

Debug.Print strSQL
CurrentDb.Execute strSQL

'~~~~~
```

Debug.Print


```
Debug.Print strSQL
```

--> this prints a copy of the SQL statement to the Debug (Immediate) window (**Ctrl G**)

After you execute your code, open the Debug window

Ctrl G to Goto the debuG window -- look at the SQL statement

If the SQL statement has an error

1. Make a new query (design view)
2. choose View, SQL from the menu
(or SQL from the toolbar, first icon)
3. cut the SQL statement from the debug window -- select the statement then **Ctrl X**
4. paste into the SQL window of the Query **Ctrl V**
5. run ! from the SQL window 

-- Access will tell you where the problem is in the SQL

Subqueries

```
SELECT field1..., (subquery) AS alias FROM datasource
SELECT fieldlist FROM datasource WHERE field comparison operator (subquery)
SELECT fieldlist FROM datasource WHERE field ANY|SOME|ALL (subquery)
WHERE expression [NOT] EXISTS (subquery)
```