NEW!

Question time

Which database querying tool is text-based and reactionary, yet immensely adaptable and even a boon in some social circles? Why, SQL of course. In the first part of our new tutorial, Mark Whitehorn introduces the basics.

SQL stands for Structured Query Language, which is referred to either as its individual letters or is called "Sequel". It appears as if the former reference is more common in the UK and the latter in the US, but as the two are interchangeable don't let it be a cause of anxiety.

Despite many similarities to C, Pascal, BASIC *et al*, SQL is not a programming language. It is a data access language or data sub-language. As such, it is a very restricted language which deals only with how tables of data can be manipulated. It lacks many of the other features (such as the ability to write information to a particular place on the screen) which characterise a full programming language.

Using SQL

SQL is often described as a standard, but when you actually start using it you find that, like many standards, it's not as standard as all that.

The examples given here are in a generic form of SQL: you may well find discrepancies depending on the actual version used. For example, the generic DISTINCT becomes DISTINCTROW in Microsoft's Access. Having said that, the differences are not great, and should not pose serious problems.

The name itself ("SQL") is somewhat misleading as it implies that this sub-language is concerned exclusively with querying. In fact, the language is sufficiently rich to allow the user to perform many other operations such as creating tables, but it remains true that the most common usage of the language is to ask questions of a database. This part of the language comprises the Data Manipulation Language (DML) statements of SQL.

DML statements are, by convention, written in UPPERCASE. The first ones we'll look at are SELECT, FROM, DISTINCT and WHERE. The sample tables shown in Fig 1 will be used for the examples.

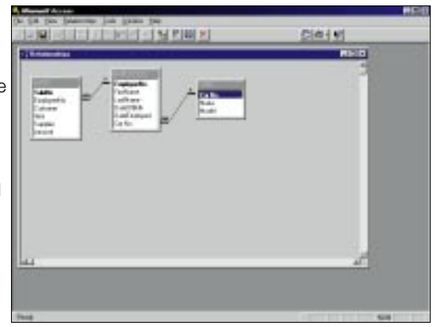

Fig 1 **The sample files used in my examples**

SELECT & FROM

The first statement, SELECT, is used to extract a collection of fields from a given table. FROM simply directs attention to the table in question. Therefore, the statement

```
SELECT SaleNo, Item, Amount
FROM SALES;
```

will yield the following;

SaleNo	Item	Amount
1	Sofa	£235.67
2	Chair	£453.78
3	Stool	£82.78
4	Suite	£3,421.00
5	Sofa	£235.67
6	Sofa	£235.67
7	Bed	£453.00

SQL doesn't eliminate duplicates by default, so:

```
SELECT Item, Amount
FROM SALES;
```

will yield

Item	Amount
Sofa	£235.67
Chair	£453.78
Stool	£82.78
Suite	£3,421.00
Sofa	£235.67
Sofa	£235.67
Bed	£453.00

DISTINCT

You can force SQL to remove the duplicates by using the statement DISTINCT, which dictates that all rows in the answer table must be unique. The query

```
SELECT DISTINCT Item, Amount
FROM SALES;
```

produces:

Item	Amount
Bed	£453.00
Chair	£453.78
Sofa	£235.67
Stool	£82.78
Suite	£3,421.00

WHERE

SELECT lets you choose the fields with which to work, and WHERE lets you choose the records.

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa';
```

produces

Item	Amount
Sofa	£235.67
Sofa	£235.67
Sofa	£235.67

while

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa' AND Customer =
'Smith';
```

yields

Item	Amount
Sofa	£235.67

All sorts of variations are already possible, combining SELECT and WHERE statements: as you can see from the last example, WHERE clauses can contain conditions.

Fig 2 One record found

EmployeeNo	FirstName	LastName	Date of Birth	DateEmployed
2	John	Greeves	21 March 1967	01 January 1990

Conditions

We'll digress here to cover the range of Conditions that are acceptable within a WHERE clause. Conditions typically consist of logical expressions which can be evaluated for truth; in other words, they are checked to discover whether they are true or false.

Thus if we use the SQL statement

```
SELECT EmployeeNo, FirstName,
LastName, DateOfBirth, DateEmployed
FROM EMPLOYEES
WHERE EmployeeNo = 2;
```

then we can expect the RDBMS to examine every record in the EMPLOYEE table, and place in the answer table only those records for which the condition

```
WHERE EmployeeNo = 2
```

is true. As you'd hope, this is only true for one record (Fig 2).

A condition is constructed from operators such as those shown in Fig 3.

The logical operators in Fig 4 have a lower priority than those above and are therefore processed after them, unless brackets are used to alter precedence.

The following SQL statement asks for a table of the items and amounts from the Sales table for sale numbers greater than six:

```
SELECT Item, Amount
FROM SALES
WHERE SaleNo > 6;
```

Item	Amount
Bed	£453.00

while this one only wants to see records for sofas for sale numbers greater than six;

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa' AND SaleNo > 6;
```

There are none.

This next statement asks for all records for sofas, suites and beds, regardless of sale number:

```
SELECT Item, Amount
FROM SALES
WHERE Item IN ('Sofa', 'Suite',
'Bed');
```

Item	Amount
Sofa	£235.67
Suite	£3,421.00
Sofa	£235.67
Sofa	£235.67
Bed	£453.00

and this one adds a condition which specifies records for the same three pieces of furniture with sale numbers greater than six:

```
SELECT Item, Amount
FROM SALES
WHERE Item IN ('Sofa', 'Suite',
'Bed') AND SaleNo > 6;
```

Item	Amount
Bed	£453.00

Conditions are nothing if not logical, and rendering a series of conditions into plain English is a good way of understanding what it will do in practice.

ORDER BY

Another useful command is ORDER BY. It gives you control over the order in which records appear in the answer table generated by the query. You specify the field by which you want records ordered, as in the following statement:

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa'
ORDER BY SaleNo;
```

where the records are ordered by the number of each sale, with the default being in ascending order. If you feel you want to specify this, the command is ASC, as shown below:

```
SELECT Item, Amount
FROM SALES
WHERE SaleNo > 6
ORDER BY Item ASC;
```

It's a perfectly acceptable statement, but it's tautological. The next statement:

```
SELECT Item, Amount
FROM SALES
WHERE SaleNo > 6
ORDER BY Item DESC;
```

will produce exactly the same data but will be sorted differently, as DESC, as you'll have gathered, sorts records in descending order. You can use sorts in both directions:

```
SELECT Item, Customer, SaleNo,
Amount
FROM SALES
WHERE SaleNo > 0
ORDER BY Customer ASC, Amount DESC;
```

Note that Amount doesn't *have* to be in the SELECT statement to be used for sorting the records in the answer table, although this would often be the case.

This will sort the customer records in ascending order, with the amounts each customer has spent shown in descending order.

Item	Customer	SaleNo	Amount
Chair	Johnson	2	£453.78

Fig 3 Operators

Symbol	Meaning	Example	Notes	Records returned from Employee table
=	Equal to	EmployeeNo = 2		1
>	Greater than	EmployeeNo > 2		2
<	Less than	EmployeeNo < 2		1
<>	Not equal to	EmployeeNo <> 2		3
>=	Greater than or Equal to	EmployeeNo >= 2		3
<=	Less than or Equal to	EmployeeNo <= 2		2
IN	Equal to a value within a collection of values	EmployeeNo IN (2, 3, 4)		3
LIKE	Similar to	LastName LIKE "Gr*"	Finds Greeves and Groves. Uses wildcards. Wild cards vary between SQL implementations.	2
BETWEEN…AND	Within a range of values, including the two values which define the limits	EmployeeNo BETWEEN 2 AND 4	Equivalent to: EmployeeNo IN (2, 3, 4)	3
IS NULL	Field does not contain a value	DateEmployed IS NULL		0

p254 ➢

Fig 4 Logical operators

Symbol	Meaning	Example	Notes	Records returned from Sales table
AND	Both expressions must be true in order for the entire expression to be judged true	SaleNo > 3 AND Customer = "Smith"	AND is evaluated before OR	1
OR	If either or both expressions are true, the entire expression is judged to be true	SaleNo > 3 OR Customer = "Smith"	AND is evaluated before OR	5
NOT	Inverts Truth	SaleNo NOT IN (2, 3, 4)	(just as well it isn't available for the real world!)	4

Suite	Jones	4	£3,421.00
Bed	Jones	7	£453.00
Sofa	Simpson	6	£235.67
Sofa	Simpson	1	£235.67
Sofa	Smith	5	£235.67
Stool	Smith	3	£82.78

Wild cards

Wild cards are used in SQL much as they are used elsewhere, for occasions where you want a range of data that fits a certain pattern. The variation below is not uncommon:

```
SELECT *
FROM SALES
WHERE SaleNo > 1;
```

In this case, the * symbol is used as a wild card, meaning "all Fields".

Fig 5 Records from SALES table

Sale No.	Employee No.	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	£3,421.00
5	3	Smith	Sofa	Harrison	£235.67

Sub-queries

The use of conditions can be expanded into sub-queries to add further refinement to queries. In the following example:

```
SELECT Customer
FROM SALES
WHERE EmployeeNo IN
   (SELECT EmployeeNo
    FROM EMPLOYEES
    WHERE DateEmployed > 12/5/89);
```

the statement inside brackets is known as a sub-query and would work perfectly happily as a query all on its own. (Incidentally, this is a good case where dialects of SQL differ. Access requires that the date be wrapped up in # symbols, thus the last line would read as

```
WHERE DateEmployed > #12/5/89#)
```

Any operation performed on a table (or tables) results in another table — one containing the answer. This is termed "closure" and it is an invariable rule. The aforementioned sub-query produces an answer table, shown here:

EmployeeNo
2
3
4

By looking at the answer table generated by the sub-query, we can see that the original statement in its full form can be simplified to:

```
SELECT Customer
FROM SALES
WHERE EmployeeNo IN (2,3,4)
```

and the records from the SALES table for which this is true are shown in Fig 5.

So the query actually yields:

Customer
Smith
Jones
Smith

We can eliminate the duplicate records by adding the word Distinct to the first line of the SQL command.

■ There will be more on honing your SQL skills in part 2 of this workshop next month.