

**This Section is optional - if you have completed all of the work to date and wish to begin an exploration of SQL, please work through this section. Not completing this work will not affect your final mark in any way, however, if you are planning on studying database design on an undergraduate course, you will probably encounter SQL. This section does not attempt a comprehensive coverage of the subject, rather it attempts to use Access to give you a “feel” for the language.**

## Structured Query Language and Access

### *Aims of Session:*

You will be able to:

- Describe the origins and purpose of SQL
- Describe the main DML, DDL and DQL SQL statements
- Use SQL statements to select subsets of data
- Use the SELECT statement to filter columns
- Use the SELECT statement to order output
- Use the JOIN statement to create VIEWS

### *The Origins of SQL*

SQL, Structured Query Language had its origins in the development of relational databases, notably the work of EF Codd while working for IBM in 1979. (“A Relational Model of Large Data for Large Shared Data Banks”).

ANSI (The American National Standards Institute) approved SQL as THE standard relational database language. This has since been adopted by ISO (the International Standards Organisation). SQL is the international standard language that developed out of the relational model; any advanced work with relational databases will inevitably involve an understanding of the language.



SQL is a declarative language, which means that a database applications programmer only needs to specify WHAT needs to be done, rather than HOW (procedural language) to do it. The Database Manager (RDBMS - Access in our case) will execute the task; transparently as far as the database programmer is concerned.

A sound knowledge of SQL will enable you to work not only with Access but with all of the major relational database systems, Oracle, Ingres, SYBASE, and Microsoft's SQL Server, and of course with database driven web sites.

## *An Overview of SQL*

As we are using Access solely as a single user database (one user can access the data at any one time) as opposed to a multi user database, we need not at this stage concern ourselves with multi user security issues. As far as we are concerned, the important aspects of SQL are the following main elements:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)

Auditing, control and security become important when we start using multi user databases with a client/server model. The Client/Server model has a central database residing on a server or servers whilst users access the database over a network from their own desktop computers. In a real life application such as keeping student records, NHS patient records, bank account details etc. a strict control of transactions and security must be maintained. The following elements of SQL deal with these issues.

- Data Control Language (DCL)
- Data Administration Commands (DAC)
- Transactional Control Commands (TCC)

## *A worked example of table creation and population*

This section is intended to provide you with a basic overview of some of the simpler aspects of SQL.

We will work through the use of Data Definition and Data



Manipulation statements within Access. As Access is designed to shield users from the complexities of SQL it provides a graphical front end to both DDL SQL statements (**Make Table** query) and DML SQL statements (**Delete** query and **Append** query).

It is also possible to implement SQL statements within Access as an **embedded** part of Visual Basic: this is the approach taken here. By embedding the SQL commands used to create a data table, and to append, delete and edit the data held in it. Please take time to examine the SQL statements used in Appendix B: even if you have no knowledge of programming in general or Visual Basic in particular, the code should make sense to you.

Other RDBMS platforms such as Microsoft SQL server, ORACLE and SYBASE support **interactive** SQL wherein a user can simply enter the SQL commands at a command prompt (and of course write stored programs).

Embedding SQL means that we will need to write (in this case) Visual Basic code and embed the SQL statements within that code. The technique of embedding SQL into code has applications in Web Design where languages such as Java and ASP can be used to create SQL statements from user input in order to connect to a database held on the Web Server. The results of these techniques can be found in many places within The Northern College web site wherever there is any interactivity requiring user input (<http://www.northern.ac.uk>). The same techniques can of course be found on many other web sites, especially on-line shopping and banking sites.

These techniques draw upon knowledge of Database Design, Visual Basic programming and Web Design.

## Data Definition Language

The Data Definition Language is concerned with actions such as creating tables, defining fields, deleting tables and redefining tables. When you create a new table in Access and define the field structures in the design view, you are using Access as a graphical front end onto the DDL SQL commands.



Consider the following statement:

**CREATE TABLE rooms  
(Rno INTEGER PRIMARY KEY, RoomDescription  
VARCHAR(10));**

The CREATE SQL statement will create a new table called rooms. The table rooms will have two fields Rno and RoomDescription: Rno is defined as a key field and an integer, whilst RoomDescription is defined as a character field with a width of 10 characters.

Executing this statement in Access will result in the following table:

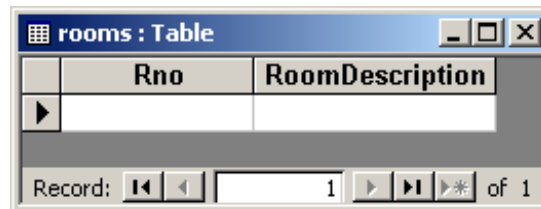


Figure 10.1: rooms - Datasheet View

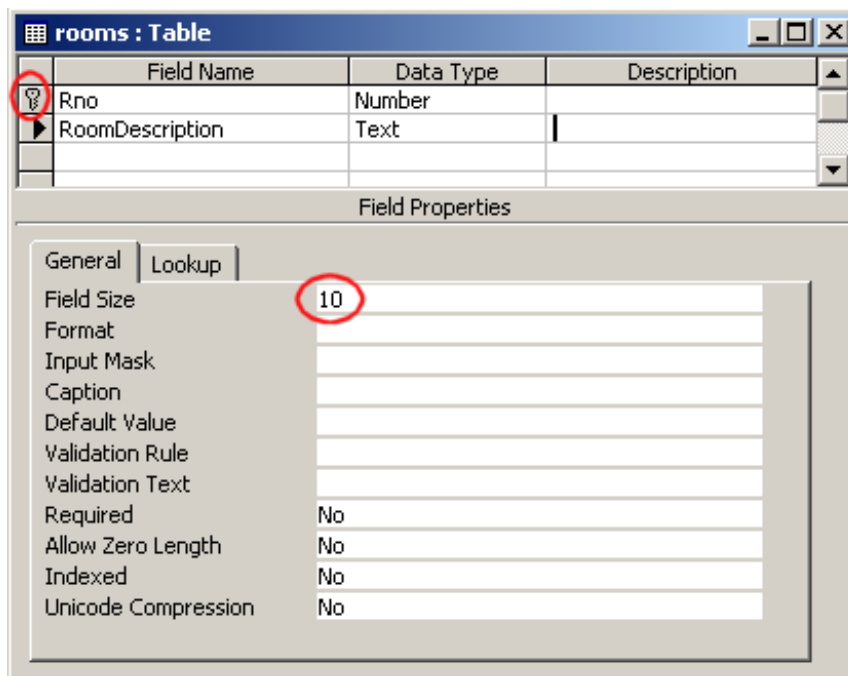


Figure 10.2: Rooms (Design View).

Notice that Rno is defined as a key field and that RoomDescription has a field size of 10 characters. Also notice that the table is empty; we've merely created the table so far. To add data to this table, we'd need to use the Data Manipulation part of SQL.



## Data Manipulation Commands

The Data Manipulation commands equate to the basic actions that we can carry out on a record in a table, namely:

<b>Insert</b>	(Add a record)
<b>Update</b>	(Edit a record)
<b>Delete</b>	(Delete a record).

The following statement will add a new record to the table Rooms (Figure 10.3).

**INSERT INTO rooms VALUES (1, 'IT Room 1');**

**Note:**

- Numerical values are not placed within quotation marks. Character values must be enclosed between single quotation marks (NOT double quotation marks)
- The order of the values MUST match the order of the fields within the table, i.e. as we have defined the table rooms to have two fields, Rno and RoomDescription in that order, we must supply the value from Rno first and the value for RoomDescription second.

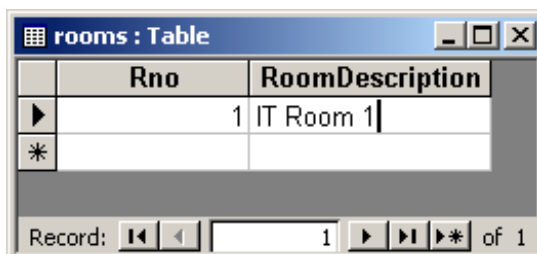


Figure 10.3: Inserting a record

Assume that we've executed the following SQL statements:

```

INSERT INTO rooms VALUES (2, 'IT Room 2');
INSERT INTO rooms VALUES (3, 'IT Room 3');
INSERT INTO rooms VALUES (4, 'MBB');
INSERT INTO rooms VALUES (5, 'IT Room 1');
INSERT INTO rooms VALUES (6, 'IT Room 1');
INSERT INTO rooms VALUES (7, 'Owen 3');
INSERT INTO rooms VALUES (8, 'Owen 4');
    
```

We now have a data table rooms as shown in Figure 10.4



Notes

Rno	RoomDescription
1	IT Room 1
2	IT Room 2
3	IT Room 3
4	MBB
5	IT Room 1
6	IT Room 1
7	Owen 3
8	Owen 4

Figure 10.4: rooms with new records

We can see from Figure 10.4 that records 5 and 6 refer to the same room (IT Room 1); this is inconsistent data.

We can use the UPDATE command to change the value of the RoomDescription field for record 5 as shown in the following SQL command.

```
UPDATE rooms  
SET RoomDescription= 'Room 11' WHERE Rno=5;
```

Note the WHERE clause, this identifies which record is to be changed (updated).

The result of this command is shown in figure10.5

Rno	RoomDescription
1	IT Room 1
2	IT Room 2
3	IT Room 3
4	MBB
5	Room 11
6	IT Room 1
7	Owen 3
8	Owen 4

Figure 10.5: Update on record 5



We still have an inconsistency with record 6; we'll resolve this by deleting the record. The SQL statement to do this is shown below.

**DELETE FROM rooms WHERE Rno=6;**

The results of this command are shown in figure 10.6.

Rno	RoomDescription
1	IT Room 1
2	IT Room 2
3	IT Room 3
4	MBB
5	Room 11
7	Owen 3
8	Owen 4

Figure 10.6: Delete record 6

## Data Query Language

The SELECT statement is the most versatile command in SQL: it is used to:

- Select Subsets of data from a table
- Order Output
- Group Output
- Select subsets of data from joined tables (views)

It is possible to use Access to implement the SELECT statement interactively by using the SQL view of a query: the following examples are implemented in this way.

This treatment of the SELECT statement is NOT definitive. We shall confine the examples to the aspects of the SELECT statement that we can effect using the SQL view of an Access query. This is not to say that the full SQL SELECT statement is not implemented in Access, rather it is implemented as an embedded SQL statement in Visual Basic. Most notably, we will not consider the GROUP BY clause which groups data into discrete sections (which can be totalled, averaged, counted etc.). Access does of course provide a way of grouping in its Report definition interface.



## The syntax of the SELECT statement

The following definition is not complete; rather it gives a simplified view of the more common aspects of the SELECT statement.

**SELECT <columnlist> | \* FROM <table | query>;**

Where <columnlist> = Column1, Column2, Column3, ... etc.

And

<table | query> is replaced with either the name of an existing table OR the name of an existing query.

**Note:** the | symbol means OR in this context.

## Examples of the SELECT statement

**SELECT \* FROM studrec;**

The \* is a wildcard character and indicates that we wish to list all columns (fields) the results of this query are shown in figure 160.

As can be seen, all columns (fields) and all records have been selected.

Let's try this again, this time listing all columns and records from a different table (CARS). The SQL statement would be:

**SELECT \* FROM cars;**

This gives the result shown in Figure 10.7.



Query2 : Select Query								
	SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	towns
▶	1	TJ	OSMAN	29-Sep-53	M	No	0	Millhouse Green
	2	S	LANGLEY	21-Aug-57	F	No	0	HUDDERSFIELD
	3	H	WILSON	07-Jul-62	M	Yes	1	HUDDERSFIELD
	4	J	CARTER	21-Mar-54	F	Yes	2	BARNESLEY
	5	A	JONES	10-Nov-48	F	Yes	2	SHEFFIELD
	6	S	ISHOMO	05-Dec-50	M	No	0	LEEDS
	7	K	ARNOTT	01-Aug-60	F	Yes	2	SHEFFIELD
	8	B	ARNOTT	23-May-62	F	Yes	1	Leeds
	9	N	GREEN	30-Sep-58	M	Yes	1	SHEFFIELD
	10	H	JACKSON	21-Apr-41	M	Yes	0	SHEFFIELD
	11	A	ARNOTT	23-Aug-54	M	Yes	2	BARNESLEY
	12	N	HEY	10-Oct-55	F	No	0	SILKSTONE
	13	K	WILSON	13-Mar-65	M	No	2	BARNESLEY
	14	J	BROWN	29-Sep-53	F	Yes	1	BARNESLEY
	15	A	ARNOTT	23-Aug-54	F	No	1	BARNESLEY
	16	G	WHITE	03-Mar-65	M	Yes	3	BARNESLEY
	17	J	GREEN	06-Aug-69	F	No	0	BARNESLEY
	18	J	GREEN	09-Aug-45	M	No	0	BARNESLEY
	19	F	WATSON	03-Mar-58	M	Yes	1	THURLSTONE
	20	L	HARVEY	03-Aug-54	F	No	2	THURLSTONE
	21	T	MOSLEY	31-Oct-75	M	Yes	2	Silkstone
	22	J	POWERS	30-Aug-45	M	Yes	0	HENDON
	23	J	CHESTER	15-Mar-60	M	No	0	BARNESLEY
	24	J	MARSHALL	03-Jul-50	F	No	2	THURLSTONE
	25	P	ARNOTT	30-Jul-65	F	No	1	SHEFFIELD
	26	W	PURDUM	01-Sep-69	M	Yes	0	SHEFFIELD
	27	B	NADIR	24-Sep-58	M	No	0	SHEFFIELD
	28	A	OLIVER	24-Feb-58	F	Yes	1	SHEFFIELD
	29	T	JAMISON	23-Jan-29	F	Yes	0	SILKSTONE
	30	C	ARNOTT	03-Apr-53	F	Yes	0	HUDDERSFIELD
	31	D	LEE	01-Jun-53	F	Yes	0	HUDDERSFIELD
	32	E	DAVIS	02-Nov-53	M	Yes	0	LEEDS
	33	F	NADIR	12-Dec-53	F	Yes	0	SHEFFIELD
	34	G	HOLLAND	27-Dec-48	M	No	0	ROTHERHAM
	35	H	KNIGHT	12-Jun-70	F	Yes	1	DONCASTER
	36	I	ARNOTT	19-Jun-72	F	Yes	2	ROTHERHAM
	37	A	WHITE	20-Feb-53	M	Yes	1	SILKSTONE
	38	G	DAVIDSON	20-Feb-53	F	Yes	1	THURLSTONE
	39	U	CAPRA	12-Feb-77	F	Yes	2	HUDDERSFIELD
	40	R	AKBAR	30-Mar-78	M	Yes	0	HUDDERSFIELD
	41	S	OSMAN	29-Aug-53	M	No	3	HUDDERSFIELD
	42	J	MARSHALL	03-Jul-50	F	No	2	Thurlstone
*						No		

Record: 1 of 42

Figure 10.7: Select \* From studrec;

Notes

	REGNO	OID	COLOUR	MODEL
	ABC 123	3	WHITE	ESCORT
	CUA 579 C	1	WHITE	CAVALIER
	D 111 ABC	4	RED	GOLF
	F 28 CWE	1	WHITE	HONDA ACCORD
	G 123 NC	5	BLACK	FIESTA
	GXN 732 Y	2	BLUE	ESCORT
	H 123 WW	4	YELLOW	2CV
	N 609 UWJ	1	MERCURY	CARINA
	SAD 1		BLACK	TRABANT
	TJO 1	1	RED	FERRARI
	TJO 2	1	GREEN	RANGE ROVER
▶	UWJ 189 Y	10	RED	MAZDA
*		0		

Record: 12 of 12

Figure 10.8: *SELECT \* FROM cars;*

It is possible to define which columns are to be selected by listing them, separating each by a comma. Consider the following example that will list all records in cars but will only list the colour, model and regno of each car (in that order): this is shown in figure 10.8.

**SELECT colour, model, regno FROM cars;**



Notes

	colour	model	regno
▶	WHITE	ESCORT	ABC 123
	WHITE	CAVALIER	CUA 579 C
	RED	GOLF	D 111 ABC
	WHITE	HONDA ACCOI	F 28 CWE
	BLACK	FIESTA	G 123 NC
	BLUE	ESCORT	GXN 732 Y
	YELLOW	2CV	H 123 WW
	MERCURY	CARINA	N 609 UWJ
	BLACK	TRABANT	SAD 1
	RED	FERRARI	TJO 1
	GREEN	RANGE ROVEI	TJO 2
	RED	MAZDA	UWJ 189 Y
*			

Record: 1 of 12

Figure 10.9: selecting specific columns from a table

### The syntax of the SELECT statement (with the WHERE clause)

The following definition is an enhancement of the previous definition given: however it remains a simplification.

**SELECT <columnlist> | \* FROM <table|query>  
WHERE <testable condition>;**

Where <testable condition> = Any valid test or expression e.g. tname='Huddersfield', kids > 2, sname <> 'ARNOTT' etc.

Let us now consider how we can filter the records we wish to select. The following statement would list all male students.

**SELECT init, sname, gender FROM studrec  
WHERE gender='M';**

This will give the following output (figure 10.10).



Notes

	init	sname	gender
▶	S	OSMAN	M
	TJ	OSMAN	M
	H	WILSON	M
	S	ISHAMO	M
	N	GREEN	M
	H	JACKSON	M
	A	ARNOTT	M
	K	WILSON	M
	G	WHITE	M
	J	GREEN	M
	F	WATSON	M
	T	MOSLEY	M
	J	POWERS	M
	J	CHESTER	M
	W	PURDUM	M
	B	NADIR	M
	E	DAVIS	M
	G	HOLLAND	M
	A	WHITE	M
	R	AKBAR	M
*			

Figure 10.10: listing all men from studrec

### The structure of studrec

Remember that earlier in this course we defined the field tname as a lookup field and linked it to the table called towns. The design view of studrec shown in figure 10.11 shows that tname derives its values from the table towns: this is why Access displays the column head as towns rather than tname.



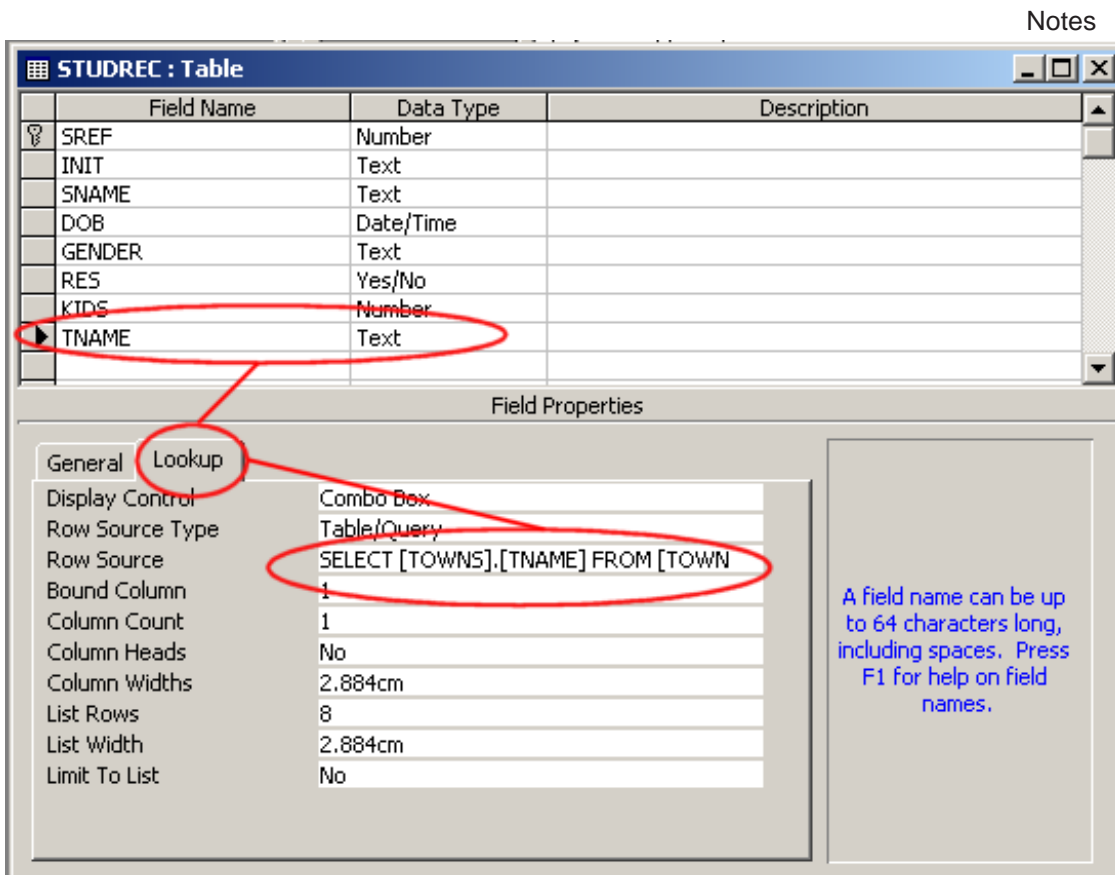


Figure 10.11: Studrec showing tname defined as a lookup field

### Complex selection criteria

A more complex selection criteria would involve more than one test. Consider the following example that selects all men who live in Sheffield (note that we'll list the tname column in this example)

```
SELECT init, sname, gender, tname FROM studrec  
WHERE gender='M' and tname='sheffield';
```

This will give the following result (figure 10.12).

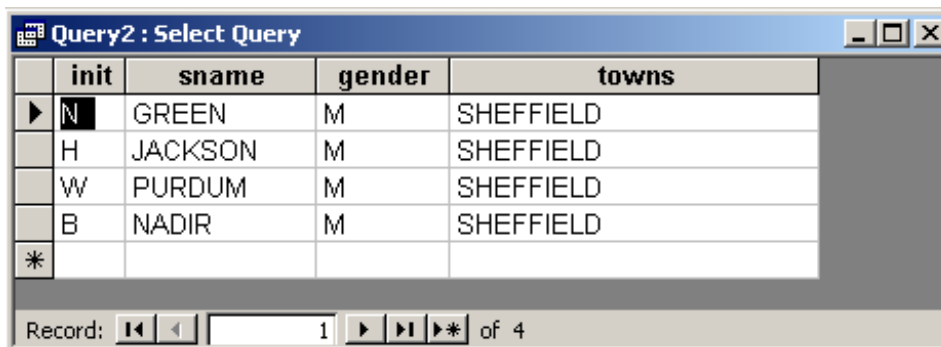


Figure 10.12: Men who live in Sheffield



Notes

Contrast the previous complex selection (using the AND logical operator) with the following statement that is identical except that the AND logical operator is replaced with the OR logical operator.

**SELECT init, sname, gender, tname FROM studrec  
WHERE gender='M' or tname='sheffield';**

As you can see from figure 10.13, the scope of the selection has been widened.

init	sname	gender	towns
S	OSMAN	M	HUDDERSFIELD
TJ	OSMAN	M	Millhouse Green
H	WILSON	M	HUDDERSFIELD
A	JONES	F	SHEFFIELD
S	ISHEMO	M	LEEDS
K	ARNOTT	F	SHEFFIELD
N	GREEN	M	SHEFFIELD
H	JACKSON	M	SHEFFIELD
A	ARNOTT	M	BARNSLEY
K	WILSON	M	BARNSLEY
G	WHITE	M	BARNSLEY
J	GREEN	M	BARNSLEY
F	WATSON	M	THURLSTONE
T	MOSLEY	M	Silkstone
J	POWERS	M	HENDON
J	CHESTER	M	BARNSLEY
P	ARNOTT	F	SHEFFIELD
W	PURDUM	M	SHEFFIELD
B	NADIR	M	SHEFFIELD
A	OLIVER	F	SHEFFIELD
E	DAVIS	M	LEEDS
F	NADIR	F	SHEFFIELD
G	HOLLAND	M	ROTHERHAM
A	WHITE	M	SILKSTONE
R	AKBAR	M	HUDDERSFIELD
*			

Figure 10.13: all men, together with those women who live in Sheffield



Finally, in our exploration of the WHERE clause of the SELECT statement let us revisit an earlier example where we worked through the development of a query to list all men who lived in either Leeds or Sheffield (Boolean Logic, sub-section 'Query Files').

The SQL statement to select these records is shown below:

```
SELECT init, sname, gender, tname
FROM studrec
WHERE gender='M' and tname='sheffield' or
gender='M' and tname='leeds';
```

	init	sname	gender	towns
▶	S	ISHEMO	M	LEEDS
	N	GREEN	M	SHEFFIELD
	H	JACKSON	M	SHEFFIELD
	W	PURDUM	M	SHEFFIELD
	B	NADIR	M	SHEFFIELD
	E	DAVIS	M	LEEDS
*				

Figure 10.14: Men who live in either Sheffield or Leeds

### The effect of precedence

All complex tests are evaluated according to strict conventions of precedence. As outlined earlier in this course (Boolean Logic, sub section - Logical Operators and Venn Diagrams) the order of precedence is:

1. ( )
2. NOT
3. AND
4. OR

Consider the WHERE clause in the previous SELECT statement. If we take the last record (*E DAVIS, male from Leeds*) as an example, we can see that the evaluation takes place as follows:

- (1) **gender='m' and tname='sheffield' or gender='m' and tname='leeds'**



Notes

In the case of E DAVIES who is male and lives in Leeds, i.e. gender='M' & tname='Leeds'. Therefore the test is 'seen' by Access as being:

(2) 'm'='m' and 'leeds'='sheffield' or 'm'='m' and 'leeds'='leeds'

which in turn evaluates to:

(3) T and F or T and T

Because  
 m=m is true (T)  
 leeds=sheffield is false (F)  
 leeds=leeds is true (T)

The **AND** conjunctive logical operator must be evaluated first as it has precedence over **OR**. This is done by using the rules defined in the **AND** truth table (below).

**AND**

A	B	OUT
0	0	0
1	0	0
0	1	0
1	1	1

**Note:** 0 = False whilst 1 = True.

There are two tests that use the AND logical operator in stage 3:

- T and F (which evaluates to F, i.e. A=1, B=0)
- T and T (which evaluates to T, i.e. A=1, B=1)

Using the AND truth table we get (from 3)

(4) F OR T

**OR**

A	B	OUT
0	0	0
1	0	1
0	1	1
1	1	1

Evaluating (4) against the OR truth table (A=0, B=1), we arrive at the ultimate logical state of the original expression (5), which is a state of True.



(5) **T**

The TRUE result means that this record will be selected.

### The effect of parentheses

Consider the following SQL SELECT statement.

```
SELECT init, sname, gender, tname
FROM studrec
WHERE sname='arnott' AND gender='M' OR
tname='sheffield';
```

This will list everyone who lives in Sheffield, together with all men who are called Arnott. This is the case because the test sname='arnott' and gender='M' will be evaluated first. The result of this test will then be OR'd with the test tname='sheffield' (figure 10.15).

init	sname	gender	towns
A	JONES	F	SHEFFIELD
K	ARNOTT	F	SHEFFIELD
N	GREEN	M	SHEFFIELD
H	JACKSON	M	SHEFFIELD
A	ARNOTT	M	BARNSELEY
P	ARNOTT	F	SHEFFIELD
W	PURDUM	M	SHEFFIELD
B	NADIR	M	SHEFFIELD
A	OLIVER	F	SHEFFIELD
F	NADIR	F	SHEFFIELD

Figure 10.15: *sname='arnott' AND gender='M' OR tname='sheffield'*

By introducing parenthesis we can effect a change in the resulting subset of records.

```
SELECT init, sname, gender, tname
FROM studrec
WHERE sname='arnott' AND (gender='M' OR tname='sheffield');
```



Notes

	init	sname	gender	towns
▶	◀	ARNOTT	F	SHEFFIELD
	A	ARNOTT	M	BARNLSLEY
	P	ARNOTT	F	SHEFFIELD
	*			

Record: 1 of 3

Figure 10.16: *sname='arnott' AND (gender='M' OR tname='sheffield)*

The introduction of parentheses changes the order in which the testable condition is evaluated. Everything within the parentheses is evaluated first: the result (in this example) is then AND'd with the test *sname='arnott'*. This is shown in figure 10.16.

### *Developing the WHERE clause.*

SQL provides a number of subsidiary operators to the WHERE clause of the SELECT statement. This section will demonstrate some of the simpler operators, namely:

- LIKE
- IN
- BETWEEN
- IS NULL

#### **LIKE**

SQL uses the LIKE operator to compare values using the wildcard operator. In many MS-DOS and Windows based operations you maybe familiar with the use of the \* character to substitute for any other character string. It is important to note that whilst SQL in most implementations will use the % sign as the wildcard character: Access uses the \* character.

For example,

T\* would match Thomas  
Tom

\*T would match Internet  
This string because it ends with the letter T

\*T\* would match Active  
Bantam



Notes

The SQL command

```
SELECT init, sname, gender, tname FROM studrec  
WHERE tname LIKE 'S*';
```

Will list all students whose home town begins with the letter S (Figure 10.17)

init	sname	gender	towns
A	JONES	F	SHEFFIELD
K	ARNOTT	F	SHEFFIELD
N	GREEN	M	SHEFFIELD
H	JACKSON	M	SHEFFIELD
P	ARNOTT	F	SHEFFIELD
W	PURDUM	M	SHEFFIELD
B	NADIR	M	SHEFFIELD
A	OLIVER	F	SHEFFIELD
F	NADIR	F	SHEFFIELD
N	HEY	F	SILKSTONE
T	MOSLEY	M	Silkstone
T	JAMISON	F	SILKSTONE
A	WHITE	M	SILKSTONE

Figure 10.17: *tname LIKE 'S\*'*

Note: this technique is useful when used in conjunction with Parameter Queries (Boolean Logic, sub-section Parameter Queries, page 44)

## IN

The IN operator enables us to select records according to whether a value is matched to a particular value in a set of given values. Consider the following clause;

```
... WHERE kids IN (1, 2, 3);
```

This will select each record where the value of kids is 1, 2 or 3. This is shown in the following statement:

```
SELECT init, sname, gender, kids FROM studrec  
WHERE kids IN (1,2,3);
```

This gives the following output (Figure 10.18)



Notes

init	sname	gender	kids
S	OSMAN	M	3
J	MARSHALL	F	2
H	WILSON	M	1
J	CARTER	F	2
A	JONES	F	2
K	ARNOTT	F	2
B	ARNOTT	F	1
N	GREEN	M	1
A	ARNOTT	M	2
K	WILSON	M	2
J	BROWN	F	1
A	ARNOTT	F	1
G	WHITE	M	3
F	WATSON	M	1
L	HARVEY	F	2
T	MOSLEY	M	2
J	MARSHALL	F	2
P	ARNOTT	F	1
A	OLIVER	F	1
H	KNIGHT	F	1
I	ARNOTT	F	2
A	WHITE	M	1
G	DAVIDSON	F	1
U	CAPRA	F	2
*			

Figure 10.18: ... WHERE kids IN (1,2,3);

## BETWEEN

SQL provides an efficient way of selecting records as a test of a range of values, i.e. all values that are part of a given set where the user defines the lower and upper bounds.

The lower and upper bounds are included in the set. Consider:

... WHERE kids between 1 and 3;

This clause will list all students who have 1, 2 or 3 children as the following example shows.

**SELECT init, sname, gender, kids FROM studrec WHERE kids BETWEEN 1 AND 3;**



Notes

The result of this query is shown in Figure 10.19 and is of course identical to the previous IN query shown in figure 10.18.

init	sname	gender	kids
S	OSMAN	M	3
J	MARSHALL	F	2
H	WILSON	M	1
J	CARTER	F	2
A	JONES	F	2
K	ARNOTT	F	2
B	ARNOTT	F	1
N	GREEN	M	1
A	ARNOTT	M	2
K	WILSON	M	2
J	BROWN	F	1
A	ARNOTT	F	1
G	WHITE	M	3
F	WATSON	M	1
L	HARVEY	F	2
T	MOSLEY	M	2
J	MARSHALL	F	2
P	ARNOTT	F	1
A	OLIVER	F	1
H	KNIGHT	F	1
I	ARNOTT	F	2
A	WHITE	M	1
G	DAVIDSON	F	1
U	CAPRA	F	2
*			

Figure 10.19: ... WHERE kids between 1 and 3;

The BETWEEN operator is particularly useful when working with DATE data types. Suppose we wished to list everyone who was born on 29<sup>th</sup> September 1953 thought to 29<sup>th</sup> September 1963: the following statement achieves this.

```
SELECT init, sname, gender, dob FROM studrec  
WHERE dob Between #9/29/1953# And #9/29/1963#;
```

Notice that the dates are represented in US format (mm/dd/yyyy); whether this is the case on your computer will depend on your windows settings.



The dates are enclosed with # symbols, this is required by Access but it is important to note that in the major SQL implantations this is not the case and single quotation marks (') are used.

The preceding select statement gives the following output (figure 10.20).

init	sname	gender	dob
TJ	OSMAN	M	29 September 1953
S	LANGLEY	F	21 August 1957
H	WILSON	M	07 July 1962
J	CARTER	F	21 March 1954
K	ARNOTT	F	01 August 1960
B	ARNOTT	F	23 May 1962
N	GREEN	M	30 September 1958
A	ARNOTT	M	23 August 1954
N	HEY	F	10 October 1955
J	BROWN	F	29 September 1953
A	ARNOTT	F	23 August 1954
F	WATSON	M	03 March 1958
L	HARVEY	F	03 August 1954
J	CHESTER	M	15 March 1960
B	NADIR	M	24 September 1958
A	OLIVER	F	24 February 1958
E	DAVIS	M	02 November 1953
F	NADIR	F	12 December 1953
*			

Figure 10.20: ... WHERE dob Between #9/29/1953# And #9/29/1963#;

### IS NULL

The IS NULL operator tests for the existence of a NULL value: remember that NULL is not the same as zero. A field with a NULL value is empty.

We could use 'IS NULL', if for example, we wished to check for any cars on our database that do not have identified owners: our test data holds such a car, the black Trabant).

The following select statement will identify the ownerless car: the resulting output is shown in figure 10.21.



**SELECT oid, regno, colour, model FROM cars  
WHERE oid IS NULL;**

oid	regno	colour	model
	SAD 1	BLACK	TRABANT
*			

Figure 10.21: ... WHERE oid IS NULL;

## SORTING OUTPUT

One of the major benefits of a computerised database system is that we can re-order the data to suit our purposes. Our use of the SELECT statement can be extended to accommodate sorting of the output by including the ORDER BY clause.

### Definition of the SELECT statement (extended to include ORDER BY)

The syntax of the SELECT statement can now be represented as:

```
SELECT <columnlist> | * FROM <table|query>
WHERE <testable condition>
ORDER BY <columnlist> [Asc | Desc];
```

Where

<columnlist> = column1, column2, ...

<testable condition> = Any valid test or expression e.g.

tname='Huddersfield', kids > 2, sname <> 'ARNOTT' etc.

Asc will indicate that the output is to be sorted in ascending order

Desc indicates that the output will be sorted in descending order

[ ] anything in square brackets is optional

**Note:** this definition of the SELECT statement remains a simplification.



## Simple Sorting

Let us consider the data in studrec: assume that we wish to list all men by order of their ages (oldest first through to youngest at the end of the list).

The following SELECT statement will achieve this.

```
SELECT studrec.INIT, studrec.SNAME, studrec.GENDER, studrec.DOB
FROM studrec
WHERE gender='M'
ORDER BY dob;
```

The output generated by this statement is shown in figure 10.22.

INIT	SNAME	GENDER	DOB
J	JACKSON	M	21 April 1941
J	GREEN	M	09 August 1945
J	POWERS	M	30 August 1945
G	HOLLAND	M	27 December 1948
S	ISHEMO	M	05 December 1950
A	WHITE	M	20 February 1953
S	OSMAN	M	29 August 1953
TJ	OSMAN	M	29 September 1953
E	DAVIS	M	02 November 1953
A	ARNOTT	M	23 August 1954
F	WATSON	M	03 March 1958
B	NADIR	M	24 September 1958
N	GREEN	M	30 September 1958
J	CHESTER	M	15 March 1960
H	WILSON	M	07 July 1962
G	WHITE	M	03 March 1965
K	WILSON	M	13 March 1965
W	PURDUM	M	01 September 1969
T	MOSLEY	M	31 October 1975
R	AKBAR	M	30 March 1978
*			

Figure 10.22: ...ORDER BY dob;

The default sort order is ascending (ASC) so we do not need to include specify this. Note that the dates are stored as Julian dates so the older the person, the smaller the value of the Julian date, hence the oldest to youngest ordering.



Notes

*“Julian dates (abbreviated JD) are simply a continuous count of days and fractions since noon Universal Time on January 1, 4713 BCE (on the Julian calendar).”*

<http://aa.usno.navy.mil/data/docs/JulianDate.html>  
 [Accessed 13/12/2002]

**Note:** BCE stands for Before Common Era where 1BCE = 1AD

Contrast this with the following statement that lists the same output in “reverse order”, i.e. youngest to oldest.

```
SELECT studrec.INIT, studrec.SNAME, studrec.GENDER, studrec.DOB
FROM studrec
WHERE gender='M'
ORDER BY dob DESC;
```

The resulting output is shown in figure 10.23.

	INIT	SNAME	GENDER	DOB
▶	R	AKBAR	M	30 March 1978
	T	MOSLEY	M	31 October 1975
	W	PURDUM	M	01 September 1969
	K	WILSON	M	13 March 1965
	G	WHITE	M	03 March 1965
	H	WILSON	M	07 July 1962
	J	CHESTER	M	15 March 1960
	N	GREEN	M	30 September 1958
	B	NADIR	M	24 September 1958
	F	WATSON	M	03 March 1958
	A	ARNOTT	M	23 August 1954
	E	DAVIS	M	02 November 1953
	TJ	OSMAN	M	29 September 1953
	S	OSMAN	M	29 August 1953
	A	WHITE	M	20 February 1953
	S	ISHMO	M	05 December 1950
	G	HOLLAND	M	27 December 1948
	J	POWERS	M	30 August 1945
	J	GREEN	M	09 August 1945
	H	JACKSON	M	21 April 1941
*				

Record: 1 of 20

Figure 10.23: ...ORDER BY dob DESC;



## Ordering output by more than one column

Often we need to order our output by more than one column, e.g. if we wished to list students alphabetically by name, in which case we would need to order the output first of all by sname, then by init. The following example demonstrates the use of more than one column to order output.

```
SELECT studrec.INIT, studrec.SNAME, studrec.GENDER, studrec.DOB
FROM studrec
WHERE studrec.GENDER='M'
ORDER BY sname, init;
```

The resulting output is shown in figure 10.24:

INIT	SNAME	GENDER	DOB
R	AKBAR	M	30 March 1978
A	ARNOTT	M	23 August 1954
J	CHESTER	M	15 March 1960
E	DAVIS	M	02 November 1953
J	GREEN	M	09 August 1945
N	GREEN	M	30 September 1958
G	HOLLAND	M	27 December 1948
S	ISHEMO	M	05 December 1950
H	JACKSON	M	21 April 1941
T	MOSLEY	M	31 October 1975
B	NADIR	M	24 September 1958
S	OSMAN	M	29 August 1953
TJ	OSMAN	M	29 September 1953
J	POWERS	M	30 August 1945
W	PURDUM	M	01 September 1969
F	WATSON	M	03 March 1958
A	WHITE	M	20 February 1953
G	WHITE	M	03 March 1965
H	WILSON	M	07 July 1962
K	WILSON	M	13 March 1965
*			

Figure 10.24: ...ORDER BY sname, init;



## JOINS

A join is a relational database term that is used to combine two or more tables (e.g. STUDREC and CARS). As with most other aspects of SQL, ACCESS provides a very efficient user interface to enable us to JOIN tables: we used this earlier in the course when we looked at Inner Joins, Left Joins and Inner Joins (Essential Relational Technique & Theory, sub-section Joins, Figures 121, 122 & 123, pages 167-168).

SQL recognises the following JOINS

EQUIJOIN	(an INNER JOIN in Access)
NATURAL JOIN	(same as an INNER JOIN except that columns are not duplicated e.g. when joining studrec and cars we would not select both studrec.sref and cars.oid because both columns refer to the same value)
OUTER JOIN	(when we select all rows in one table and matching rows in another. Access sub divides the OUTER JOIN into a LEFT JOIN and a RIGHT JOIN)
NON-EQUIJOIN	(we do not have to join tables where the Key and Foreign Keys are equal, other operators such as >, < <> can be used. We could, for example use this to obtain a list of students who do not own cars.
SELF JOIN	(Where a table is joined to itself.)

Notes



## Practical Examples

The following examples will show how SQL statements are implemented via an Access query. These examples are necessarily limited in scope and anything more complex needs to be implemented either as embedded SQL within Visual Basic (which Access is equipped with) or via a more specialised SQL database Management System such as SQL Server or Oracle.

### Car Parking

Consider our earlier example of listing all cars and their owners (the right join example, Figure 123 - page 168): we listed all cars, together with all students who owned a car.

CARS was joined to STUDREC on the Foreign Key OID and Key SREF (Figure 10.25)



right join : Select Query

STUDREC

\* SREF  
INIT  
SNAME  
DOB

1 ← ∞

CARS

\* REGNO  
OID  
COLOUR  
MODEL

Field:	Table:	Sort:	Show:	Criteria:	or:
SREF	STUDREC		<input checked="" type="checkbox"/>		
INIT	STUDREC		<input checked="" type="checkbox"/>		
SNAME	STUDREC		<input checked="" type="checkbox"/>		
REGNO	CARS		<input checked="" type="checkbox"/>		
OID	CARS		<input checked="" type="checkbox"/>		
COLOUR	CARS		<input checked="" type="checkbox"/>		
MODEL	CARS		<input checked="" type="checkbox"/>		

Notes

Figure 10.25: Design View of CARS Right JOINED to STUDREC



Notes

The SQL statement that lies behind Figure 10.25 is shown below:

```
SELECT studrec.sref, studrec.init, studrec.sname,
cars.regno, cars.oid, cars.colour, cars.model
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid;
```

The result of this statement is shown in figure 10.26.

SREF	INIT	SNAME	REGNO	OID	COLOUR	MODEL
1	TJ	OSMAN	CUA 579 C	1	WHITE	CAVALIER
1	TJ	OSMAN	F 28 CWE	1	WHITE	HONDA ACCORD
1	TJ	OSMAN	TJO 1	1	RED	FERRARI
1	TJ	OSMAN	TJO 2	1	GREEN	RANGE ROVER
2	S	LANGLEY	GXN 732 Y	2	BLUE	ESCORT
3	H	WILSON	ABC 123	3	WHITE	ESCORT
4	J	CARTER	D 111 ABC	4	RED	GOLF
4	J	CARTER	H 123 WW	4	YELLOW	2CV
5	A	JONES	G 123 NC	5	BLACK	FIESTA
10	H	JACKSON	UWJ 189 Y	10	RED	MAZDA
1	TJ	OSMAN	N 609 UWJ	1	MERCURY	CARINA
			SAD 1		BLACK	TRABANT

Figure 10.26: List all cars together with their owners

The Access implementation of the JOIN clause differs from the ISO SQL standard implementation. Once again, you should take note that other DBMS such as SQL Server and Oracle will use a slightly different notation.

The JOIN clause is a sub clause of the FROM clause of the SELECT statement. We could substitute the terms LEFT JOIN or INNER JOIN if we wished; this would give the output as shown in Figures 6.14 and 6.13. respectively).



Notes

### Refining the output

The list shown in figure 10.26 might be a little more readable if we omitted to list sref and oid. Access allows us to simply omit these columns from the <columnlist > part of the SELECT statement. The amended statement is shown below:

```
SELECT studrec.init, studrec.sname, cars.regno, cars.colour, cars.model
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid;
```

The resulting output is shown in figure 10.27.

INIT	SNAME	REGNO	COLOUR	MODEL
TJ	OSMAN	CUA 579 C	WHITE	CAVALIER
TJ	OSMAN	F 28 CWE	WHITE	HONDA ACCORD
TJ	OSMAN	TJO 1	RED	FERRARI
TJ	OSMAN	TJO 2	GREEN	RANGE ROVER
S	LANGLEY	GXN 732 Y	BLUE	ESCORT
H	WILSON	ABC 123	WHITE	ESCORT
J	CARTER	D 111 ABC	RED	GOLF
J	CARTER	H 123 WW	YELLOW	2CV
A	JONES	G 123 NC	BLACK	FIESTA
H	JACKSON	UWJ 189 Y	RED	MAZDA
TJ	OSMAN	N 609 UWJ	MERCURY	CARINA
		SAD 1	BLACK	TRABANT
*				

Figure 10.27: omitting studrec.sref and cars.oid from the output.

### Sorting the output

A further improvement to the readability of the output would be to list the rows by the student's name: there are of course many ways in which we could list the data, which we choose depends upon the use that is to be made of the output.

The following version of the SELECT statement shows the JOIN clause being used together with the ORDER BY clause.

```
SELECT studrec.init, studrec.sname, cars.regno, cars.colour, cars.model
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid
ORDER BY studrec.sname, studrec.init;
```

The resulting output is shown in figure 10.28.



Notes

	INIT	SNAME	REGNO	COLOUR	MODEL
▶			SAD 1	BLACK	TRABANT
	J	CARTER	H 123 WW	YELLOW	2CV
	J	CARTER	D 111 ABC	RED	GOLF
	H	JACKSON	UWJ 189 Y	RED	MAZDA
	A	JONES	G 123 NC	BLACK	FIESTA
	S	LANGLEY	GXN 732 Y	BLUE	ESCORT
	TJ	OSMAN	N 609 UWJ	MERCURY	CARINA
	TJ	OSMAN	TJO 2	GREEN	RANGE ROVER
	TJ	OSMAN	TJO 1	RED	FERRARI
	TJ	OSMAN	F 28 CWE	WHITE	HONDA ACCORD
	TJ	OSMAN	CUA 579 C	WHITE	CAVALIER
	H	WILSON	ABC 123	WHITE	ESCORT
*					

Record: 1 of 12

Figure 10.28: Output sorted by student's name

### Filtering rows

Finally in our example, let us filter out any cars that do not have owners (the abandoned Trabant). We can do this by incorporating the WHERE clause into the SELECT statement, this is shown below:

```
SELECT studrec.init, studrec.sname, cars.regno,
cars.colour, cars.model
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid
WHERE NOT cars.oid IS NULL
ORDER BY studrec.sname, studrec.init;
```

The resulting output is shown in figure 10.29.



Notes

	INIT	SNAME	REGNO	COLOUR	MODEL
▶	J	CARTER	H 123 WW	YELLOW	2CV
	J	CARTER	D 111 ABC	RED	GOLF
	H	JACKSON	UWJ 189 Y	RED	MAZDA
	A	JONES	G 123 NC	BLACK	FIESTA
	S	LANGLEY	GXN 732 Y	BLUE	ESCORT
	TJ	OSMAN	N 609 UWJ	MERCURY	CARINA
	TJ	OSMAN	TJO 2	GREEN	RANGE ROVER
	TJ	OSMAN	TJO 1	RED	FERRARI
	TJ	OSMAN	F 28 CWE	WHITE	HONDA ACCORD
	TJ	OSMAN	CUA 579 C	WHITE	CAVALIER
	H	WILSON	ABC 123	WHITE	ESCORT
*					

Figure 10.29: ... WHERE NOT cars.oid IS NULL

### What Next?

This practical and ACCESS orientated introduction to SQL is only the starting point. For very large data sets, perhaps 50,000 records or more, we cannot rely on a 'desktop' RDBMS. Microsoft market SQL Server and other companies also produce their own products, perhaps the most widely used is ORACLE: these 'industrial strength' RDBMS's provide developers with the full range of SQL commands and are optimised for very large data sets.

The SQL examples we have been examining are a good grounding on some of the basics of SQL. It is important to realise that there are many other aspects to SQL in general and to the DQL (Data Query Language) in particular.

The SELECT statement can be developed so that it makes use of sub queries, which is a query that is contained within the WHERE clause of a query, i.e. a nested query. This enhancement makes it possible to provide the main query with previously unknown, or at least undefined, data.

Other aspects of SQL we have not touched upon include optimising the performance of databases by the use of indexes, summarising data and of course managing security constraints for multi user databases.



If you are considering a university course that covers the design and development of databases, you will almost certainly be introduced to the relational model; this in turn means that you will almost certainly be required to develop an understanding of SQL. I hope that this relatively practical approach will stand you in good stead for your future studies and development.

## *How to Use the SQL View in Access*

Whilst all of the preceding SQL examples have been thoroughly tested with Access 2000, all of the statements were pasted into this document from Access; I took care to edit them slightly by removing some of the superfluous parentheses that Access automatically inserts. I also took care to use the return key to display the SQL commands over a number of lines rather than as one long string of characters. This was done to enhance (human) readability. Do not be concerned if your SQL statements differ slightly from the examples in this chapter, what matters is that you are able to use SQL to effect the desired outcomes and to understand how the commands work.

This section will guide you through a practical example of using the SQL view of an Access query.

### **Developing an SQL statement to list all car owners**

This example will take you through using the SQL view of an Access query using the example of listing car owners as shown in figure 176. Your ability to work through this tutorial depends on you having completed the earlier work on relational databases and in particular, it is important that you have both the cars table fully populated, together with studrec.



Ensure that you have Access running and that you have the Student Records database running.

You should see something like the screen shown in figure 10.30.



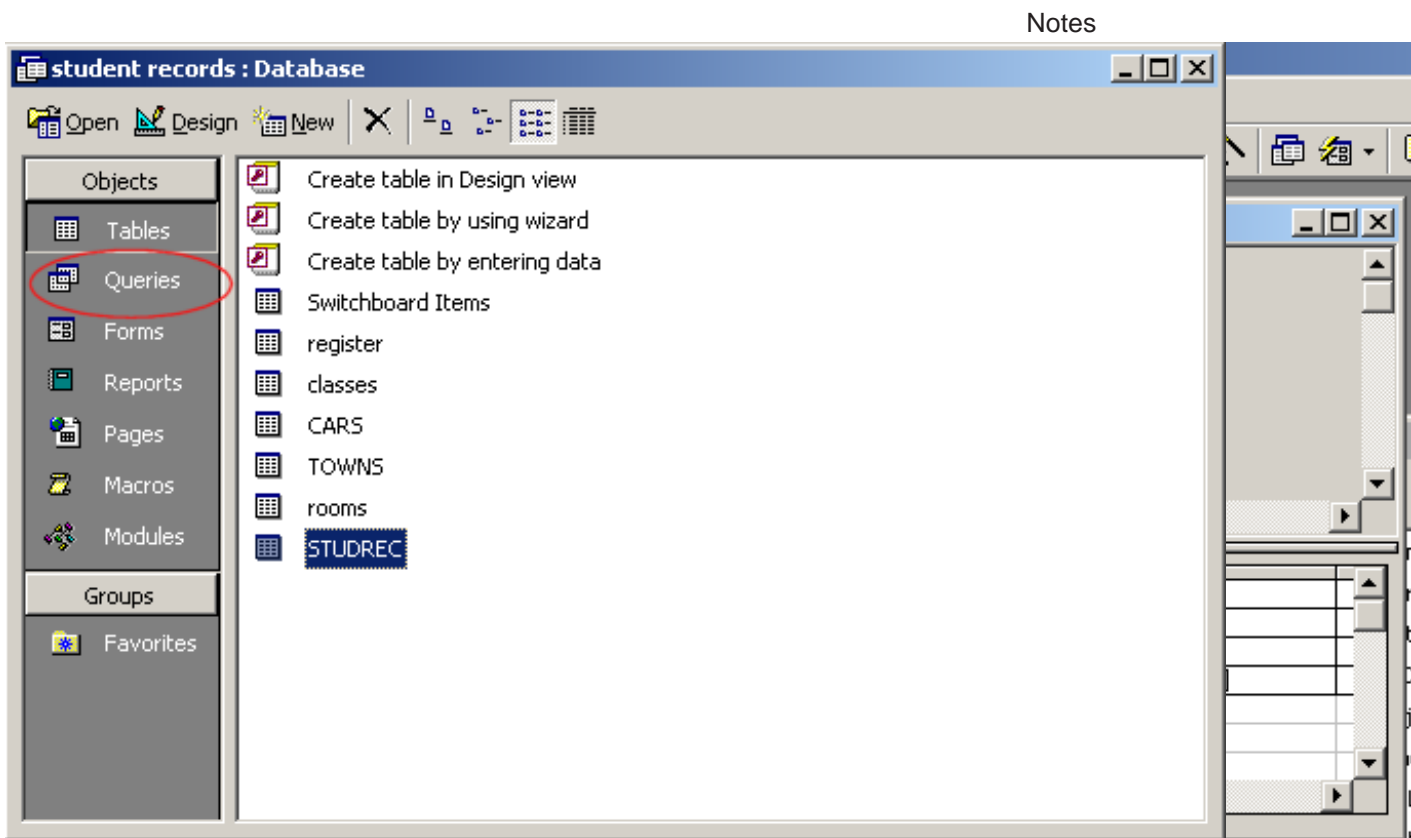


Figure 10.30: The Student Records database

- ☛ Select the Queries object
- ☛ Create a new query in design view
- ☛ Click on the CLOSE button of the SHOW Tables window (DO NOT add any tables) – Figure 10.31

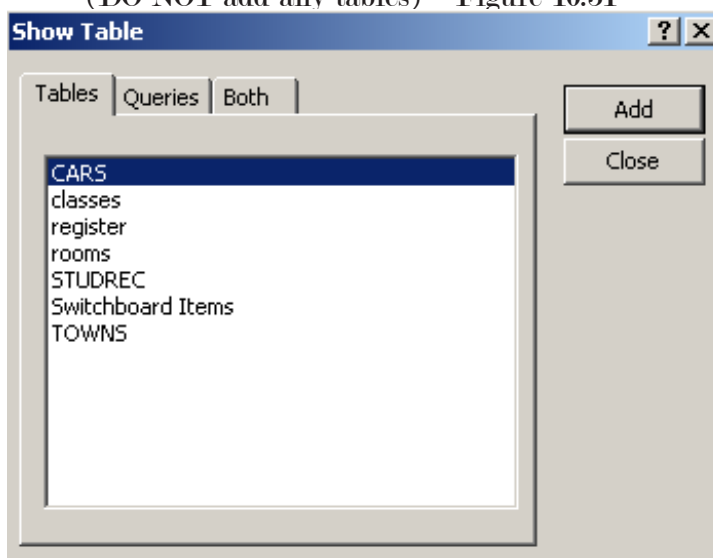


Figure 10.31: Show Table window



Notes



Select the SQL view of the Query as shown in figure 10.32.

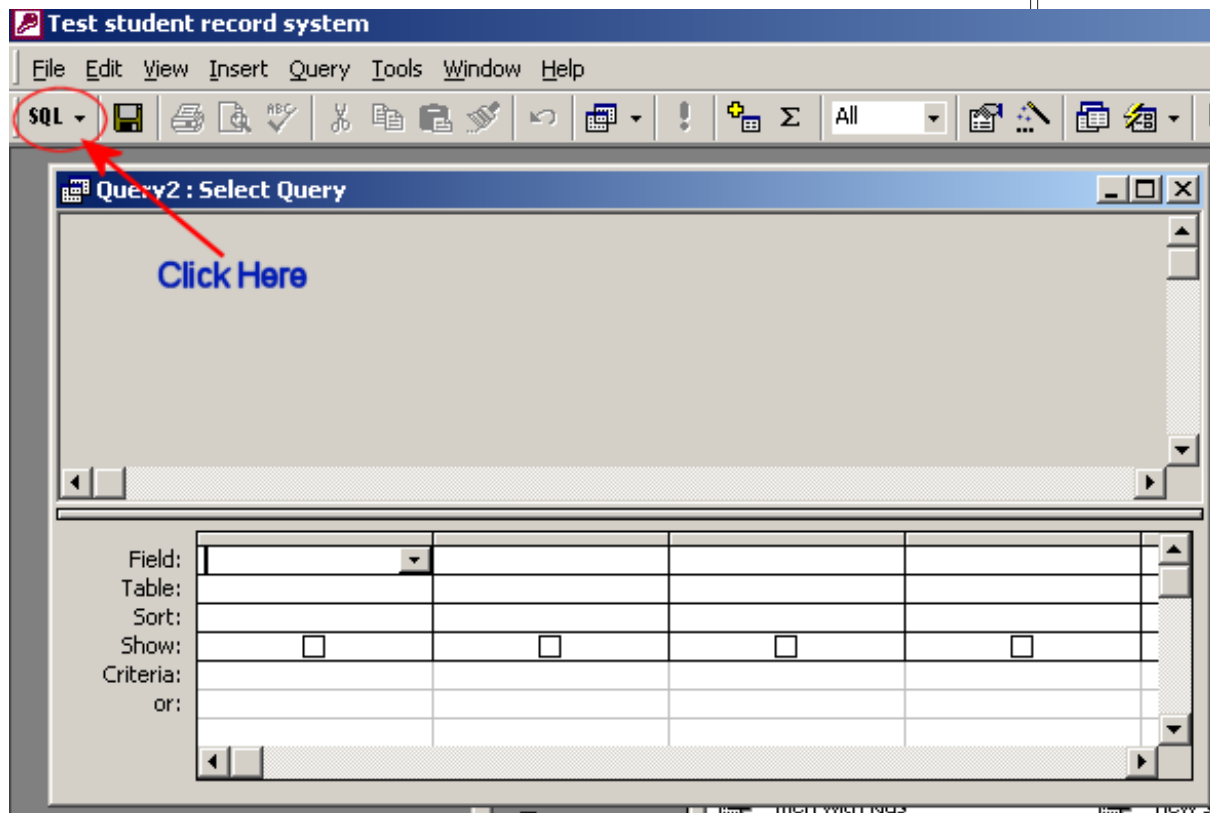


Figure 10.32: Switching to the SQL View



Click on the SQL View Icon

We are now ready to begin entering the SQL statement. The complete statement is shown below, however it is good practice to build it up stage by stage (at least when we are learning about SQL).

```
SELECT studrec.init, studrec.sname, cars.regno,
cars.colour, cars.model
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid
WHERE NOT cars.oid IS NULL
ORDER BY studrec.sname, studrec.init;
```



Notes



Type in the statement shown below (don't forget to include the semi colon (;) at the end of the statement).

```
SELECT studrec.init, studrec.sname, cars.regno, cars.colour, cars.model  
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid;
```

Your screen should be something like the one shown in figure 10.33.

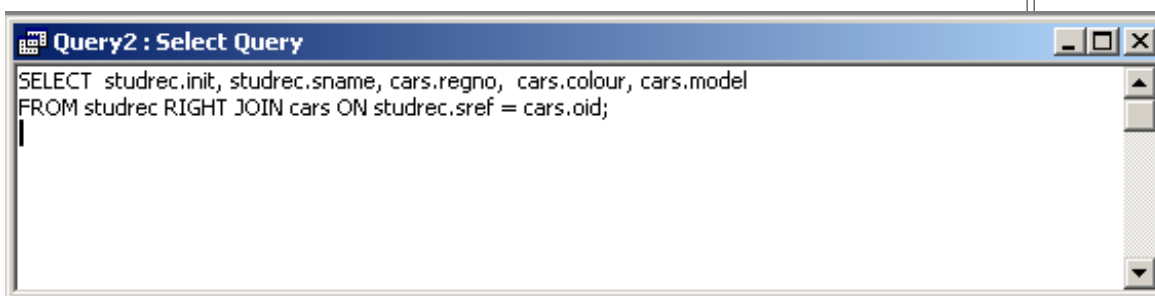


Figure 10.33: *SELECT statement version 1*



Check the output by switching to DATASHEET view at this stage.



Check all is well (compare with Figure 10.27) & switch back to SQL view



Edit the **SELECT** statement to include the **ORDER BY** clause (Figure 10.28), i.e. add **ORDER BY studrec.sname, studrec.init**

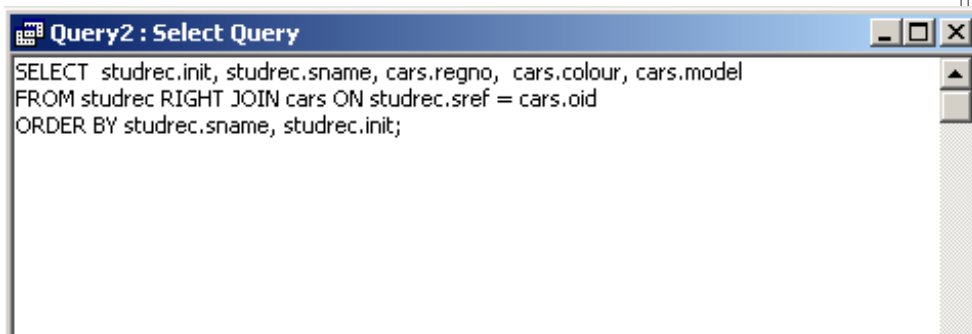


Figure 10.34: *SELECT statement version 2*



Check that your output compares with that shown in figure 10.35.



Notes



Edit the **SELECT** statement once more to filter out the ownerless Trabant (Figure 10.35), i.e. add **WHERE NOT cars.oid IS NULL**

```
SELECT studrec.init, studrec.sname, cars.regno, cars.colour, cars.model
FROM studrec RIGHT JOIN cars ON studrec.sref = cars.oid
WHERE NOT cars.oid IS NULL
ORDER BY studrec.sname, studrec.init;
```

Figure 10.35: *SELECT* statement version 3 (final version)



Check that your output compares with that shown in figure 10.29.



Save your query as **SQL-CAR OWNERS**

It is interesting to see how Access has used a graphical interface to interpret this SQL statement.



Open the query **SQL-CAR OWNERS** and switch to design view, you should see something like figure 10.36.



**SQL - CAR OWNERS : Select Query**

studrec

\* SREF  
INIT  
SNAME  
DOB

cars

\* REGNO  
OID  
COLOUR  
MODEL

**Right Join of cars ON studrec**

**Sort by SNAME then by INIT**

**do NOT include cars where OID is null**

Field:	INIT	SNAME	REGNO	COLOUR	MODEL	INIT	OID
Table:	studrec	studrec	cars	cars	cars	studrec	cars
Sort:		Ascending				Ascending	
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:							Not Is Null
or:							

Notes

Figure 10.36: The SELECT statement represented graphically



Notes

Finally, let us see if Access has amended our SELECT statement in any way.



Switch to SQL view, you should see the following  
(Figure 10.37)

```
SQL-CAR OWNERS : Select Query
SELECT studrec.INIT, studrec.SNAME, cars.REGNO, cars.COLOUR, cars.MODEL
FROM studrec RIGHT JOIN cars ON studrec.SREF = cars.OID
WHERE ((Not (cars.OID) Is Null))
ORDER BY studrec.SNAME, studrec.INIT;
```

Figure 10.37: Access has added parentheses

Note that the parentheses that Access has automatically added do not change the way in which the SELECT statement is executed.



## Exercise 10

You should submit your answers in word processed form ONLY. Ensure that each answer is clearly numbered and labelled (e.g. 10.1: List all female students).

You should also take care to check your answers by implementing them in Access. Do NOT submit Access screenshots or printouts for marking.

The tables used in these questions are STUDREC and CARS.

Ensure that you always include sufficient columns to ensure that your output is meaningful and in any case always include studrec.init and studrec.sname.

What SQL statements would you use to achieve the following output?

- 1: List all female students
- 2: List all students whose family name begins with 'M'.
- 3: List all students who have between 1 and 2 children(inclusive).
- 4: List all students in order of their home town
- 5: List all students from Barnsley who have children
- 6: List all students who were born in the 1960's
- 7: List only those students who own cars
- 8: List all Huddersfield students who own cars
- 9: List all students who own yellow, red, black or blue cars
- 10: List all students and all of the cars, showing who owns which cars

