Introduction to the theory behind QBE & SQL

Robin Beaumont Wednesday, 14 September 2011
robin@organplayers.co.uk

Contents

1. INTRODUCTION ................................................................................................................. 2
2. INTRODUCTION TO SQL ........................................................................................................ 2
3. QUERYING A SINGLE TABLE .................................................................................................. 3
   3.1 RETRIEVING SELECTED ROWS (RECORDS) ................................................................. 3
   3.2 RETRIEVING SELECTED COLUMNS (FIELDS) ............................................................ 4
   3.3 RETRIEVING ONLY UNIQUE VALUES IN A COLUMN ...................................................... 4
   3.4 RETRIEVING SELECTED COLUMNS (FIELDS) AND ROWS (RECORDS) ......................... 4
   3.5 SUMMARY INFORMATION FOR COLUMNS .................................................................. 5
   3.6 CREATING NEW FIELDS FOR RESULTS ....................................................................... 5
   3.7 SUMMARY INFORMATION FOR COLUMNS GROUPED BY SPECIFIC VALUES ..................... 6
4. WHAT SQL AND QBE WON'T ALLOW ..................................................................................... 6
5. QUERYING MORE THAN ONE TABLE - JOINS ........................................................................ 6
   5.1 FIELDS MATCH IN BOTH TABLES (EQUI-JOIN / NATURAL JOIN / INNER JOIN) ............... 7
   5.2 FIELDS MATCH IN RIGHT HAND TABLE (LEFT OUTER JOIN) ........................................ 8
   5.3 FIELDS MATCH IN LEFT HAND TABLE (RIGHT OUTER JOIN) ........................................ 10
6. UNION QUERIES .................................................................................................................. 11
7. IMPORTANCE OF PREPARATION FOR COMPLEX QUERIES .............................................. 11
8. SUMMARY ............................................................................................................................. 11

This chapter assumes you have already carried out the practical chapters up to selection queries (2) or ideally up to - queries using multiple tables.
1. Introduction

The most common criticism of databases is the difficulty of getting useful information out of them which is rather strange considering it is the principle reason why most of them are developed in the first place.

Interestingly the difficulty of obtaining most information is often because it has been saved in an inappropriate way, that is the data is unstructured i.e. not normalised. This will be the subject of the third database concepts chapter but for the time being examples will be given that consist of normalised (i.e. structured) data.

The easiest way to get information out of a database is the 'query' it. The Access practical sessions used a graphical technique (literally ‘point and click) known as QBE (Query by example). This chapter builds on this practical experience providing a introduction to what is actually going on under the bonnet.

Details of the way a single as well as multiple tables can be queried are given in this chapter along with a method of planning your queries by using a structured English approach called SQL (Structured query language). It is often pronounced ‘sequal’ or ‘ess-cue-ell’ depending if you are in the UK or not. The structured English, SQL, approach is developed for two reasons:

- It should help you formulate your ideas of what you want to achieve before trying it out at the computer.
- SQL is even more common than QBE and allows you to do more than QBE.

2. Introduction to SQL

After reading this chapter you are not expected to be able to understand the complexities of SQL. What you will be able to do is create basic SQL statements which will help formulate your ideas. This will also provide a grounding for those of you who wish to take it further and learn to use SQL in Access. Unfortunately although SQL is a recognised standard the way it works in Access, like most relational databases is slightly different. The examples that are provided in this chapter will run in Access.

SQL basically consists of a very simply sentence:

```
SELECT field(s) FROM table(s);
```

A simple example might be:

```
SELECT *
FROM doctor;
```

Which would produce all the records from the doctor table for any particular database you might have open at the time.

**Bold type** indicates what are called 'reserved words' that is the computer knows these are special and interprets them accordingly. The words are also written in capitals to help you identify them although when you type a SQL sentence into Access you do not need to do either of these things. Note the all important semi-colon at the end of the sentence.

During this chapter we will expand this very simple sentence, technically called an SQL statement, to enable us to obtain all the information that was obtained using QBE in the Access practical chapters with the added advantage that we have specified it in an English like sentence.

**Specifying field names in Access SQL using square brackets.** As mentioned above Access's implementation of SQL differs from the international standard SQL in some ways. One important aspect is the result of Access allowing field names which have spaces in them, such as 'doctor id' To refer to such fields in an SQL statement you need to place them in square brackets e.g [doctor id].

This is one reason why I often specify field names that make more sense with a space instead with a lower dash i.e. instead of [DOCTOR İD] I would have usually given the field name DOCTOR_ID instead.
3. Querying a single table

A database table is simply a set of rows and columns filled with some data thus:

```
A table
```

Basically QBE and SQL allows us to do certain things with these rows and columns of data such as select a subset of rows or/and columns as well as obtaining summaries for columns. Each of these various operations on a table will now be described below.

3.1 Retrieving selected rows (records)

An obvious thing one might want to do is retrieve a number of rows\(^1\) (records) based upon one or more values in them.

```
A table
```

```
Selected records
```

In SQL this is:

```
SELECT *
FROM 'table name'
WHERE "\*\*\*\*\*";
```

and you replacing:

- 'table name' with the one you are interested in (e.g. doctor table in the cons4 database)
- The "\*\*\*\*\*" with criteria such as [gender] = 2 (giving you all the male GPs in this instance). Note the square brackets to indicate the field name. Optional in this instance because there are no spaces in the field name.

The * after the Select reserved word means select all fields in the table

**Exercise 1**

Describe how you would achieve the above effect using the QBE grid

---

\(^1\) In database theory this is called restriction.
3.2 Retrieving selected columns (fields)
Another thing which seems highly useful is the retrieval of just a few columns\(^2\) from a large table.

![A table with selected fields](image)

In SQL this is:

```sql
SELECT 'field names'
FROM 'table name'
;
```

Each ‘field name’ you specify should be separated by a comma e.g. [patient id], [patient surname].

**Exercise 2**
Describe how you would achieve the above effect using the QBE grid.

3.3 Retrieving only unique values in a column
This is achieved by adding the reserved word **DISTINCT** in the SQL after the 'select' :

```sql
SELECT DISTINCT [patient name]
```

This will return each unique name only once, regardless of the fact that it may occur several times in the 'patient name' field e.g. Smith.

This is equivalent to setting the UniqueValues property to Yes in the query property sheet in a query's Design view.

3.4 Retrieving selected columns (fields) and rows (records)
Frequently you want to combine the selection of rows and columns:

![A table with selected fields and columns](image)

In SQL this is:

```sql
SELECT 'field names'
FROM 'table name'
WHERE * * * *
;
```

\(^2\) In database theory this is called projection.
3.5 Summary information for columns

Wanting to know the number of specific values in a field, such as the maximum, minimum or mean value for a particular column can be specified easily in QBE.

In SQL this is:

```
SELECT 'summary function(field name).....'
FROM 'table name' . . .
```

In the ‘field names’ slot of the expression enclose the field we wish to obtain summary values for with the name of the type of summary we are interested in. Suppose we are interested in obtaining the total number of doctors in the doctors table this would become Count([doc id]). Alternatively imagine we are interested in obtaining the mean value for all the values in a column (field) called pulse, this would become Avg([pulse]) in our phrase. Notice that the words used for the summary functions are the same as those in the table in the practical chapter Selection Queries 2.

**Exercise 3**

Look at the SQL below and say how you would achieve the same effect using the QBE grid

```
SELECT Count([Doc id])
FROM doctor
;
```

You can specify any number of summary functions and fields, separating each by a comma.

3.6 Creating new fields for results

We can also put the result in a new filed as we did in the QBE grid. To obtain the same result in SQL all we do is add another reserved word to the sentence **AS**:

```
SELECT Count(doctor.[Doc id])
AS [CountOfDoc id]
FROM doctor
;
```

The **AS** reserved word indicates the name of the new field to put the result in. You can repeat the ‘**AS**’ reserved word as many times as you like for pairs of field names. Each pair is separated, as usual, by a comma.
3.7 Summary information for columns grouped by specific values

Considering the above example which provides the total number of doctors, a logical question following on from that is to find the number of males and females. That is:

A table

Totals, counts, mean etc.
for each value

In SQL this is:

```sql
SELECT 'field names'
FROM 'table name'
GROUP BY 'field names' . . . .
```

In the 'field names' slot following the 'group by' part of the phrase add the names of the field(s) we wish to group by. Therefore to obtain a count of GPs grouped by sex we would add the name of the field which held the gender value for each record, the 'gender' field in this case.

A final extra little piece of information for the high fliers. You can subsequently filter out records that do not match a criteria after you have grouped them by adding a 'having' term in the above phrase:

```sql
select Count([doctor id])
from doctor
group by [gender]
having Count([gender]) > 5;
```

This reports the total number of doctors for either sex if there are more than 5. A typical answer might be 7 females and 6 males. If there where 5 or less they would not be reported. This is very useful for finding out the most frequent or rare values in a particular field, such as diagnostic codes. You can also extend this to find a particular number or percentage of records e.g. TOP 25 PERCENT in which case you must also specify the ordering of the records e.g. ORDER BY [diastolic BP] DESC. Different database implementation of SQL provide slightly different ways of achieving this.

4. What SQL and QBE won't allow

From the above it appears that most of the things you can do with tables are concerned with columns. In particular the summary functions and 'group by' only work for columns. This is deliberate, having to do with the mathematical theory upon which relational databases are based. Basically if you want to do something like add up along rows or possibly add the totals for three columns it usually means you have not structured the table correctly. All these can be done but it usually means more fuss than you feel it's worth at the end of the day.

You have been warned!

5. Querying more than one table - Joins

This again can be considered by way of extending the table diagram used above. Because we are now dealing with more than one table when specifying fields they will be proceeded by the table name and a full stop (i.e. doctor.gender refers to the gender field in the doctor table). Incidentally Access automatically adds the table name as a prefix like this when it generates the SQL from and QBE you have produced.
5.1 Fields match in both tables (equi-join / natural join / inner join)

In this situation only those rows are chosen that match in both tables. This is the default for Libreoffice bases and Access when you run a QBE query on more than one table.

To specify a join it is necessary to state:

- Which tables are involved.
- Which are the fields that form the link between the tables.

In SQL:

```
SELECT 'fields'
FROM 'first table name'
INNER JOIN 'second table name'
ON 'first table name.field name' = 'second table name.field name'
```

Taking as an example the first exercise in the practical chapter Queries using multiple tables where you retrieve the total number of patients for the male and female partners.

```
SELECT doctor.[Gender], Count(patient.[patient id])
FROM doctor
INNER JOIN patient
on doctor.[Doc id] = patient.[Doc id]
group by doctor.[Gender];
```
Patients episodes example
As a second example consider the link between patients and episodes in the cons4 database. Suppose we want to find all patients that have visited the doctor for a consultation. We would therefore run a join query on patients and episodes:

```
FROM patient
INNER JOIN episode
ON episode.[patient id] = patient.[patient id]
```

The result is given below. Fifteen records have been retrieved, the entire episodes table. This is as would be expected as all the patient id values found in the episode table also exist in the patient table.

<table>
<thead>
<tr>
<th>episode id</th>
<th>episode.patient id</th>
<th>episode.Doc id</th>
<th>patient.patient id</th>
<th>Title</th>
<th>first name</th>
<th>Surname</th>
<th>patient.Doc id</th>
<th>Date seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>23</td>
<td>1</td>
<td>mr</td>
<td>john</td>
<td>smith</td>
<td>23</td>
<td>23/04/94</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>mr</td>
<td>john</td>
<td>smith</td>
<td>23</td>
<td>06/05/94</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>miss</td>
<td>shella</td>
<td>jones</td>
<td>23</td>
<td>13/04/73</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>prof</td>
<td>richard</td>
<td>farmer</td>
<td>23</td>
<td>02/06/90</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>03/02/96</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>23/02/96</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>03/03/96</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>01/01/95</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>05/01/95</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>11/01/95</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>17/01/95</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>28/01/95</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>10/02/95</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>27/02/95</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>20/03/95</td>
</tr>
</tbody>
</table>

5.2 Fields match in right hand table (left outer join)
Suppose we wanted all the records from the left hand table including those that did not have matching records in the other table.

```
Table A
Selected records = all in table A + matching records in table B
Table B
004
004
004
004
```

In SQL:
```
SELECT 'fields'
FROM 'left table name'
LEFT JOIN 'right table name'
ON 'left table name.field name' = 'right table name.field name'
```

Patients episodes example

Consider again the link between patients and episodes in the cons4 database. In the above example we discovered all the patient episodes, however it would be more use to discover the visits for all registered patients (i.e. all patient ids in the patient table). This can be done with a left inner join.

. . . .

FROM patient

LEFT JOIN episode

ON patient.[patient id] = episode.[Patient ID];

The result is given below showing that several registered patients have not had any consultations (episodes).

<table>
<thead>
<tr>
<th>episode id</th>
<th>episode_id</th>
<th>episode_doc_id</th>
<th>patient_id</th>
<th>Title</th>
<th>first name</th>
<th>Surname</th>
<th>patient_doc_id</th>
<th>date_seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>23</td>
<td>1</td>
<td>Mr</td>
<td>John</td>
<td>Smith</td>
<td>23</td>
<td>23/04/94</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Mr</td>
<td>John</td>
<td>Smith</td>
<td>23</td>
<td>06/05/94</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Miss</td>
<td>Sheila</td>
<td>Jones</td>
<td>23</td>
<td>13/04/73</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>Prof</td>
<td>Richard</td>
<td>Farmer</td>
<td>23</td>
<td>02/06/90</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>Mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>03/02/96</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>Mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>23/02/96</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>Mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>03/03/96</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>01/01/95</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>05/01/95</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>11/01/95</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>17/01/95</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>28/01/95</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>10/02/95</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>27/02/95</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Shirley</td>
<td>Anderson</td>
<td>1</td>
<td>20/03/95</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Mr</td>
<td>Chris</td>
<td>Bull</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Mrs</td>
<td>Ann</td>
<td>Cookson</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Mr</td>
<td>Gordon</td>
<td>Brown</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Mr</td>
<td>Alan</td>
<td>Bull</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Ms</td>
<td>Christina</td>
<td>Verdi</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Karen</td>
<td>Wedge</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>Miss</td>
<td>Sarah</td>
<td>Williams</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
5.3 Fields match in left hand table (right outer join)

Suppose we wanted all the records from the right hand table including those that did not have matching records in the other table.

Table A

<table>
<thead>
<tr>
<th>004</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
</tr>
<tr>
<td>004</td>
</tr>
</tbody>
</table>

Table B

<table>
<thead>
<tr>
<th>004</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Selected records = all in table B + matching records in table A

In SQL:

```
SELECT 'fields'
FROM 'left table name'
RIGHT JOIN 'right table name'
ON 'left table name.field name' = 'right table name.field name'
```

Patients episodes example

Consider again the link between patients and episodes in the cons4 database. Suppose we wanted to find out if unregistered patients had had consultations (episodes). This question could be answered using a right join.

```
FROM patient
RIGHT JOIN episode
ON patient.[patient id] = episode.[Patient ID]
```

The result is given below showing that all patient episodes were for patients which had a ID in the patient table.

<table>
<thead>
<tr>
<th>episode.id</th>
<th>episode.patient.id</th>
<th>episode.Doc.id</th>
<th>patient.patient.id</th>
<th>Title</th>
<th>first name</th>
<th>Surname</th>
<th>patient.Doc.id</th>
<th>Date seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>23</td>
<td>1</td>
<td>mr</td>
<td>john</td>
<td>smith</td>
<td>23</td>
<td>23/04/94</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>mr</td>
<td>john</td>
<td>smith</td>
<td>23</td>
<td>06/05/94</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>miss</td>
<td>shella</td>
<td>jones</td>
<td>23</td>
<td>13/04/73</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>prof</td>
<td>richard</td>
<td>farmer</td>
<td>23</td>
<td>02/06/90</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>03/02/96</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>23/02/96</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>mr</td>
<td>John</td>
<td>Hewitt</td>
<td>1</td>
<td>03/03/96</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>01/01/95</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>05/01/95</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>11/01/95</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>17/01/95</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>28/01/95</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>10/02/95</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>27/02/95</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>miss</td>
<td>shirley</td>
<td>anderson</td>
<td>1</td>
<td>20/03/95</td>
</tr>
</tbody>
</table>
6. Union Queries

These type of queries allow you to select all specified fields from one or more tables. They can only be developed by using SQL. You can't create them in QBE. They are very useful if you wish to combine several fields together from different tables.

As an example suppose you had a table called 'union test', with data in three fields (f1, f2, f3) which you wish to find the combined sum of. The data is given below:

<table>
<thead>
<tr>
<th>ID1</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>2</td>
<td>162</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>3</td>
<td>456</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>4</td>
<td>765</td>
</tr>
</tbody>
</table>

The SQL is given below:

```sql
SELECT SUM([f1])
FROM [union test]
UNION ALL SELECT SUM([f2])
FROM [union test]
UNION ALL SELECT SUM([f3])
FROM [union test];
```

This creates a single field with three records in it, each being the total for the three above fields. 

7. Importance of preparation for complex queries

If you intend to carry out a complex query. Such as one involving several tables or several criteria on a single large table it is sensible to work with dummy data first and manually check your results. For example you would copy a few records to a temporary database which you would run the query on, knowing the result having worked it out by hand before on the few records. Alternatively you can create purpose build records yourself.

This may seem like unnecessary preparation but all seasoned database managers do this religiously.

8. Summary

This chapter has provided an introduction into the standard method available for getting information out of relational DBMSs' namely SQL. Libreoffice Base SQL and Access SQL, which is slightly different from the international standard, was introduced as a method primarily to help formulate queries on paper before going to the computer.

In this chapter I have introduced you to a number of SQL reserved words. The knowledge gained from this chapter is sufficient to provide the basics for someone wishing to begin to write SQL in either Libreoffice Base or Access.

The importance of having the data in the correct structure (i.e. fields in the correct table in other words normalised) to facilitate the easy use of SQL was stressed. and just how do you decide which fields go in which table is a core aspect of Modelling which is a complete topic in itself.
A better way of obtaining the totals is to use the following SQL:

```sql
SELECT Sum([f1]) + Sum([f2]) + Sum([f3]) AS result INTO unionresult
FROM [union test];
```

This creates a new table called 'unionresult'. With one field and one record in it consisting of the total for all three fields in the original table.

The same result can also be obtained in QBE:

<table>
<thead>
<tr>
<th>Field</th>
<th>result: Sum([f1]) + Sum([f2]) + Sum([f3])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>Expression</td>
</tr>
<tr>
<td>Sort:</td>
<td></td>
</tr>
<tr>
<td>Show</td>
<td></td>
</tr>
<tr>
<td>Criteria</td>
<td></td>
</tr>
<tr>
<td>Or:</td>
<td></td>
</tr>
</tbody>
</table>