The HIMALAYAN DATABASE

The Expedition Archives of Elizabeth Hawley

Program Guide for Windows

Appendix J: SQL Searches

Himal 2.0

Richard Salisbury

The Himalayan Database

October 2017

Contents

Appendix J: SQL Searches

The **SQL Search** commands in the **Search** menu allows you to build sophisticated searches (or queries) that can extract data from one or more tables.

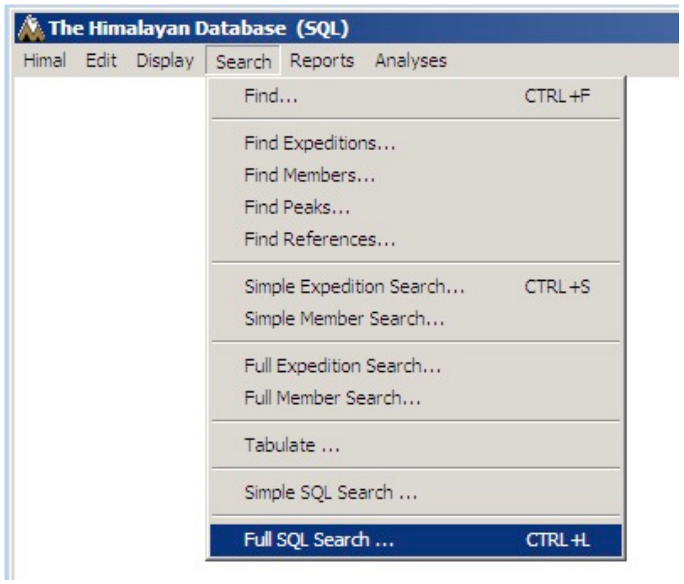
SQL (Structured Query Language and usually pronounced "sequel") is a standardized computer language that was developed in the 1970s by the IBM Corporation for accessing information stored in database tables. Relational databases produced by IBM, Oracle, Microsoft, and other software vendors support various versions or dialects of SQL. Microsoft Visual FoxPro is one of those databases.

The SQL supported by the earlier versions Visual FoxPro conforms more closely to the original version of the language often called SQL86, not the more recent dialects such as SQL92, SQL99, and SQLJ.

This section gives only a minimal introduction to the SQL language and the common features that are supported both by Visual FoxPro 6, used by the original version 1.x releases of *The Himalayan Database*. Visual FoxPro 9 expanded support to include many features of the SQL92 dialect of SQL and is now used in the current version 2.0 and later releases of *The Himalayan Database*.

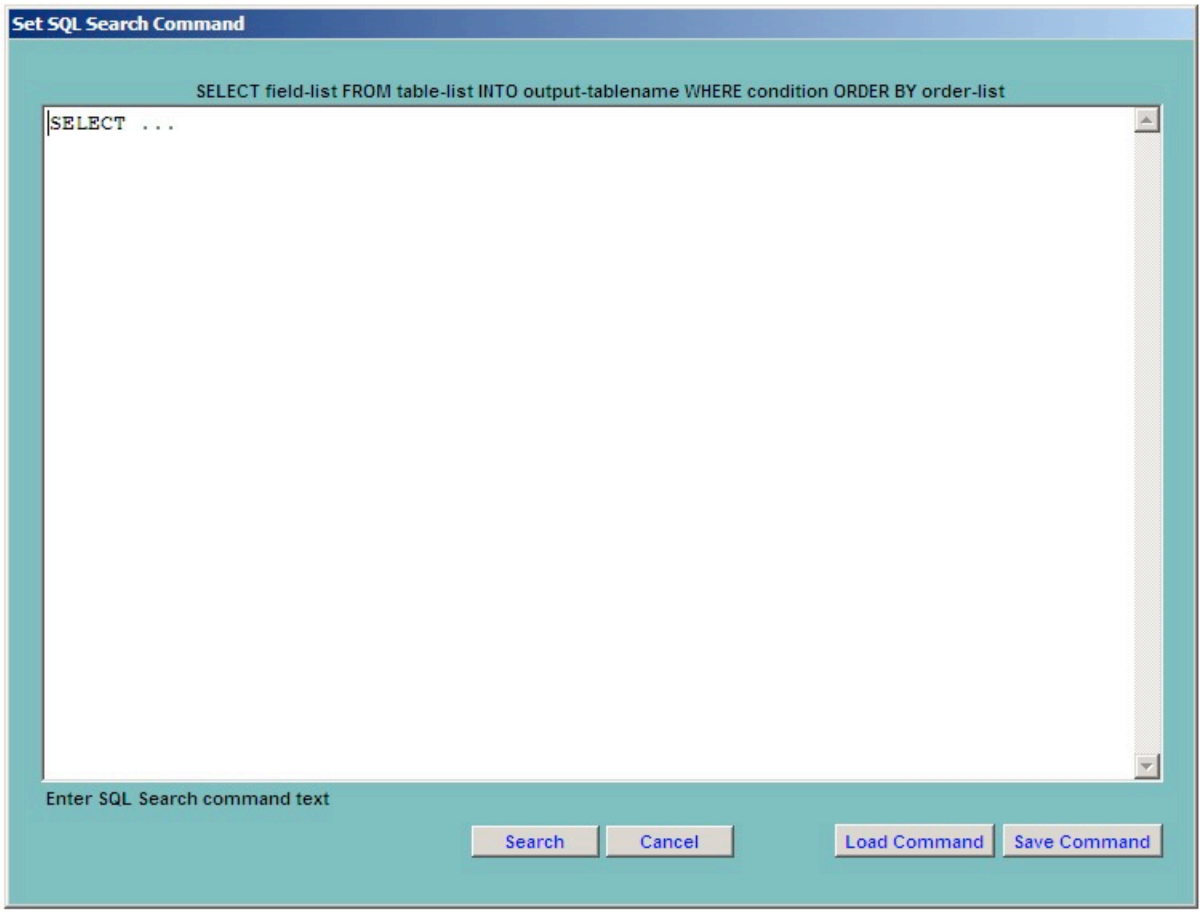
There are many computer books available in bookstores that give a more comprehensive description of SQL and the more elaborate queries that can be formulated. One such book is *Mastering SQL* by Martin Gruber (Sybex, 2000, 976 pages) that is based on his classic volume *Understanding SQL* (Sybex, 1990, 434 pages). Either edition provides a good foundation for using SQL. Of course, you can take a more lowbrow approach and use *SQL for Dummies* by Allen G. Taylor (IDG Books, 2003, 432 pages).

The **Simple SQL Search** and **Full SQL Search** commands are located in the **Search** menu:



The remainder of this appendix will describe the use of the Full SQL Search command. The Simple SQL Search command offers an assisted method of constructing SQL searches once the basic SQL language is understood.

Clicking on the Full SQL Search commands brings up the Set SQL Search Command dialog:


Set SQL Search Command
SELECT field-list FROM table-list INTO output-tablename WHERE condition ORDER BY order-list
SELECT ...
Enter SQL Search command text
Search
Cancel
Load Command
Save Command

The simplest form of the SELECT statement that can be used with the Himal program is

SELECT field-list FROM table-list WHERE condition
    ORDER BY order-list

The FROM "table-list" clause gives the tables that are to be queried for extracting the data. For the Himalayan Database, the tables normally will be one or more of Peaks, Exped, and Members; for example:

peaks
peaks, exped
exped, members

The "field-list" describes the data that is to be extracted from the database tables by the query. This is normally a list of table fields given in the format "tablename.fieldname" such as

```
peaks.peakid, peaks.pkname, peaks.heightm
exped.expid, exped.year, exped.season, exped.route1
members.expid, members.lname, members.fname, members.citizen
peaks.pkname, exped.year, exped.season, exped.route1
```

The table names and field names used by the Himalayan Database are described in Appendix B of the Himalayan Database Program Guide.

An alias for a table name may be used to shorten the field-list. For the remainder of this appendix, we will use the aliases “p” for Peaks, “x” for Exped, and “m” for Members. So our above examples would be

```
peaks p
peaks p, exped x
exped x, members m
```

and

```
p.peakid, p.pkname, p.heightm
x.expid, x.year, x.season, x.route1
m.expid, m.lname, m.fname, m.citizen
p.pkname, x.year, x.season, x.route1
```

The WHERE clause describes how the data is be searched during the query. The syntax for the “condition” is generally the same as that used by the **Search**, **Browse** and **Expor**t commands in the Himalayan Database and is described in Appendix C of the Himalayan Database Program Guide. The WHERE clause is optional, but is almost always used.

The ORDER BY clause gives the order in which the results are to be sorted. The “order-list” is usually one or two of the field names (the second being a secondary sort order). The ORDER BY clause is optional.

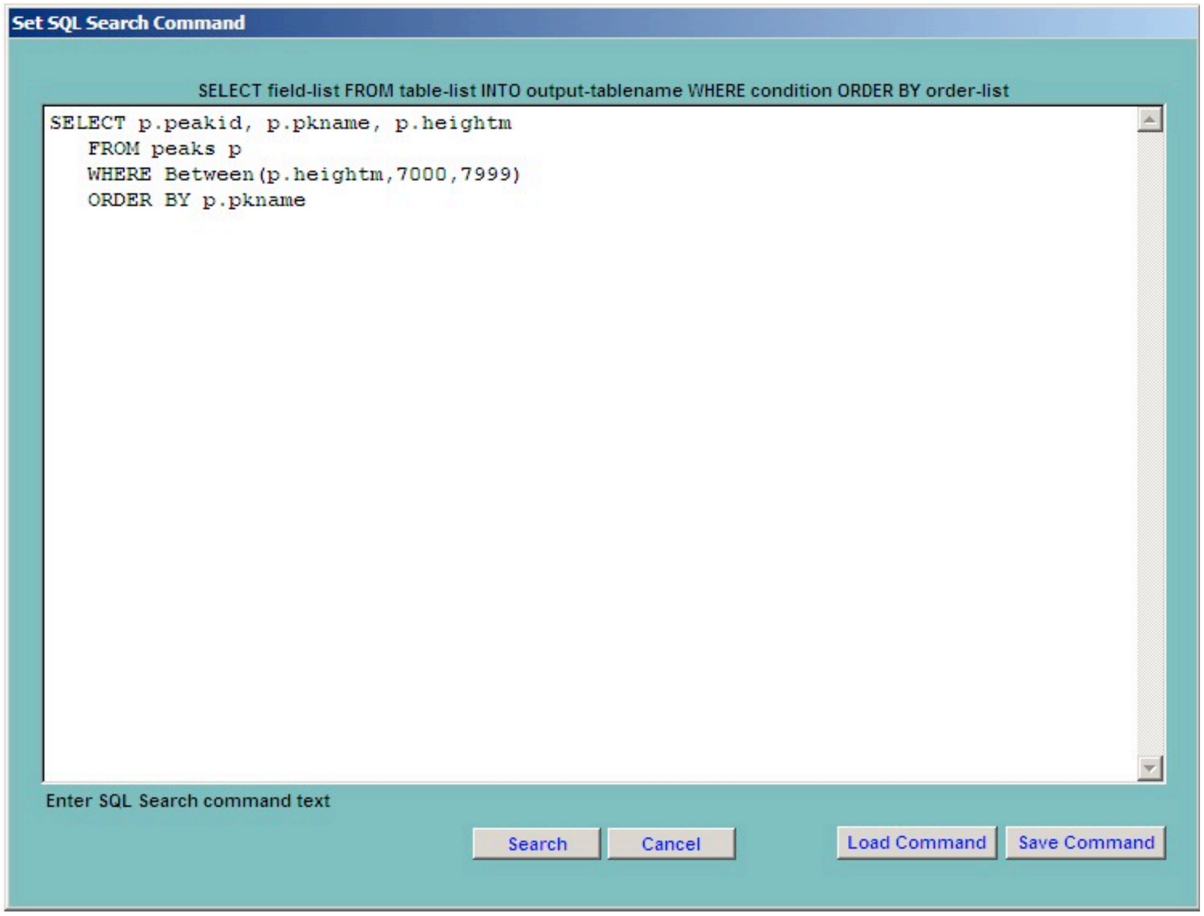
For example, to search for all of the 7000m peaks in the Himalayan Database, you can construct a SQL statement of the form

```
SELECT p.peakid, p.pkname, p.heightm
    FROM peaks p
    WHERE Between(p.heightm,7000,7999)
    ORDER BY p.pkname
```

The second, third and fourth lines are indented to force a blank between the end of the previous line and the next line. The entire command is passed to Visual FoxPro as one long command line and the FROM, WHERE and ORDER BY clauses each must be preceded by a blank. Since you cannot tell by looking if a line ends with a blank, it is a good habit to indent the next line.

The result of this query will contain one record for each peak in the 7000m range giving the peak ID, the peak name, and the peak height.

To execute this query, enter the SQL command text into the dialog box:


Set SQL Search Command
SELECT field-list FROM table-list INTO output-tablename WHERE condition ORDER BY order-list
Enter SQL Search command text
Search
Cancel
Load Command
Save Command

```
SELECT p.peakid, p.pkname, p.heightm
   FROM peaks p
   WHERE Between(p.heightm,7000,7999)
   ORDER BY p.pkname
```

In the above example and all of the following examples, the SQL keywords are given in uppercase for clarity; however, they may be used either in upper, lower, or mixed case.

If you want all of the fields from all of the tables in your search, you can use "*" for the field-list; for the above example, this would be given as:

```
SELECT * FROM peaks p WHERE Between(p.heightm,7000,7999)
    ORDER BY p.pkname
```

In another example, to search for all American and Canadian women that attempted Everest:

```
SELECT m.peakid, m.fname, m.lname, m.citizen, m.myear, m.mseason
    FROM members m
    WHERE m.peakid="EVER" And m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA")
    ORDER BY m.myear, m.mseason
```

Note the use of Upper function that matches all upper and lowercase spellings of USA and Canada in the database.

The result of this query may contain duplicate records where the peak ID, the first and last names, and the citizenship are the same when one woman has made multiple attempts on Everest. To eliminate the duplicates, the DISTINCT keyword is used to force the result to have only one copy of each record:

```
SELECT DISTINCT m.peakid, m.fname, m.lname, m.citizen,
        m.myear, m.mseason
    FROM members m WHERE m.peakid="EVER" And m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA")
    ORDER BY m.myear, m.mseason
```

To search for all those (both men and women) age 60+ summiting Everest:

```
SELECT m.peakid, m.lname, m.fname, m.citizen, m.calcage,
        m.myear, m.mseason FROM members m
    WHERE m.peakid="EVER" And m.msuccess And m.calcage>=60
    ORDER BY m.calcage
```

Note the insertion of "And m.msuccess" to indicate that only successful attempts are wanted and the use of Calcage field instead of the Age field from the Members table since only the calculated age is available in the published version of the database.

Using the DISTINCT keyword in the above example:

```
SELECT DISTINCT m.peakid, m.lname, m.fname, m.citizen, m.calcage,
        m.myear, m.mseason FROM members m
    WHERE m.peakid="EVER" And m.msuccess And m.calcage>=60
    ORDER BY m.calcage
```

would eliminate very few (if any) duplicate records since most duplicate members found in the query would have attempted Everest at a different age each time, thus the resulting records would be different in the Calcage field.

The above examples have only queried a single table to produce the result. The real power of the SQL Select command is to produce results by searching multiple tables using join-conditions.

We now will expand our example to search for all those age 60+ that have summited any Nepalese 8000m peak, ordered by peak ID, then by age:

```
SELECT p.peakid, p.pkname, m.lname, m.fname, m.citizen, m.calcage,
        m.myear, m.mseason FROM members m, peaks p
    WHERE m.peakid=p.peakid And m.msuccess And
        m.calcage>=60 And p.heightm>=8000
    ORDER BY p.peakid, m.calcage
```

Note that this query includes some non-traditional 8000m peaks such as Annapurna East & Central, Kangchenjunga Central & South and Yalung Kang.

The query for all American and Canadian women that attempted Everest from only the Nepal side requires the searching of both the Members and Exped tables by joining both tables:

```
SELECT m.peakid, x.host, m.fname, m.lname, m.citizen, m.myear,
        m.mseason
    FROM members m, exped x
    WHERE m.expid=x.expid And m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA") And x.host=1 And
        m.peakid="EVER"
    ORDER BY m.myear, m.mseason
```

In the above example, to substitute the peak name into the result of the query instead of the peak ID, the Peaks, Members and Exped tables would need to be joined to extract the peak name from the Peaks table:

```
SELECT p.pkname, x.host, m.fname, m.lname, m.citizen, m.myear,
        m.mseason
    FROM peaks p, members m, exped x
    WHERE m.peakid=p.peakid And m.expid=x.expid And
        m.sex="F" And Inlist(Upper(m.citizen),"USA","CANADA") And
        x.host=1 And m.peakid="EVER"
    ORDER BY m.myear, m.mseason
```

These examples illustrate the use of “join-conditions” that are required to show how the tables are linked. The normal join-conditions between the Himalayan Database tables are:

Exped with Members	exped.expid = members.expid
Exped with Peaks	exped.peakid = peaks.peakid
Exped with Refer	exped.expid = refer.expid
Members with Peaks	members.peakid = peaks.peakid

If the join-conditions are omitted, then meaningless and potentially disastrous results could occur.

In the example of the age 60+ climbers summiting 8000m peaks, if there were 15 8000m peaks in the Peaks table and 100 climbers had summited some of these 15 peaks, then the result should be 100 records. But by omitting the join-condition, 1500 (10 x 50) records would be the result due to runaway cross-linking between the tables. This is sometimes described as a “Cartesian join.”

If an unjoined query were done between the Exped table (9000+ records) and the Member table (65,000+ records) without any other conditions on the query, the result would exceed 585,000,000 (9000 x 65,000) records and would likely freeze the computer due to insufficient memory to complete the query. Imagine the potential results of an unjoined query between the three Himalayan Database tables (450 x 9000 x 65,000)!

The results of the query are normally displayed in a Browse grid window on your computer screen. Instead of displaying the result, you can redirect the output to a permanent Visual FoxPro dbf-type table by inserting the INTO clause into your SELECT statement:

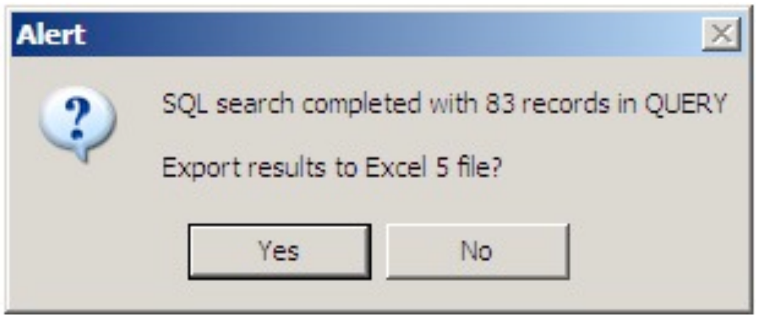
```
SELECT field-list FROM table-list INTO TABLE output-table-name
    WHERE condition ORDER BY order-list
```

The output table name must begin with an alphabetic character. For example,

```
SELECT p.peakid, p.pkname, p.heightm FROM peaks p
        INTO TABLE peaks7000
        WHERE Between(p.heightm,7000,7999)
        ORDER BY p.pkname
```

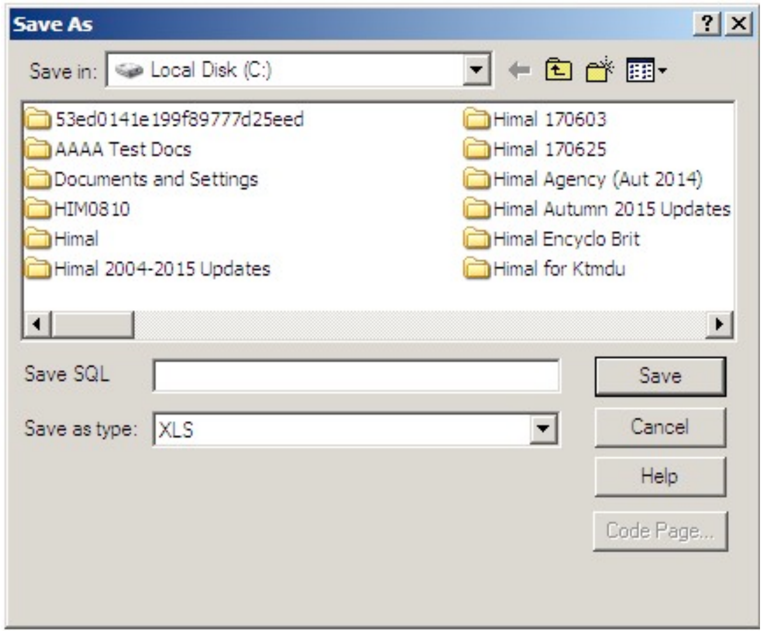
Output tables from SQL queries may be used in subsequent SQL queries as is discussed later in this section.

After you have executed your query and have finished viewing the output on your screen, you can also save the output to an Excel file, which in most cases is more useful than saving the output to a Visual FoxPro table.


Alert

SQL search completed with 83 records in QUERY

Export results to Excel 5 file?


Save As

Save in: Local Disk (C:)

53ed0141e199f89777d25eed
AAAA Test Docs
Documents and Settings
HIM0810
Himal
Himal 2004-2015 Updates
Himal 170603
Himal 170625
Himal Agency (Aut 2014)
Himal Autumn 2015 Updates
Himal Encyclo Brit
Himal for Ktmdu

Save SQL:

Save as type: XLS

You can save and retrieve your SQL commands in the same manner as you save **Search** and **Export** command conditions by using the **Save Command** and **Load Command** buttons on the Set SQL Search Command dialog. See the end of Appendix C for further details.

Special SQL Operators

The above examples used the Visual FoxPro Between and Inlist functions in the condition expressions for the WHERE clause in the SELECT command.

SQL also has similar functions in its own language that may be used instead of the Visual FoxPro functions:

SQL Function	Visual FoxPro Function
field-name BETWEEN value1 AND value2	Between(field-name,value1,value2)
field-name IN (value1,value2,…)	Inlist(field-name,value1,value2,…)

The Visual FoxPro Inlist function also can be reversed to the form

```
Inlist(value,field-name1,field-name2,…)
```

whereas the SQL IN function cannot be reversed.

For example, using the SQL functions

```
SELECT p.peakid, p.pkname, p.heightm FROM peaks p
    WHERE Between(p.heightm,7000,7999)
    ORDER BY p.pkname
```

could be rewritten as

```
SELECT p.peakid, p.pkname, p.heightm FROM peaks p
    WHERE p.heightm BETWEEN 7000 AND 7999
    ORDER BY p.pkname
```

and

```
SELECT m.peakid, m.fname, m.lname, m.citizen FROM members m
    WHERE m.peakid="EVER" And m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA")
```

could be rewritten as

```
SELECT m.peakid, m.fname, m.lname, m.citizen FROM members m
    WHERE m.peakid="EVER" And m.sex="F" And
        Upper(m.citizen) IN ("USA","CANADA")
```

Other publications that directly describe the SQL language will use the SQL functions in their examples.

Aggregate Operators and Grouping Results

The SQL language provides several special functions for aggregating data from several records in a table:

COUNT(*)	counts the number of records in the query result
SUM(field-name)	produces the arithmetic sum of all the values in "field-name"
AVG(field-name)	produces the average (mean) of all the values in "field-name"
MIN(field-name)	produces minimum value of all the values in "field-name"
MAX(field-name)	produces maximum value of all the values in "field-name"

These SQL function may be used in the field-list and in the GROUP BY and HAVING clauses of the SELECT command (but not in the condition phrase of the WHERE clause). Of these, the COUNT and SUM functions are probably the most useful with the Himalayan Database.

The GROUP BY clause enables you to group values in a query based on the values of one or more fields and is specified in the form

```
GROUP BY group-list
```

where "group-list" is normally a list of table field names.

To illustrate these concepts, we will perform a sequence of queries that search for the number of deaths on Everest between 1995 and 1996.

The first query is used to collect the raw data

```
SELECT x.year, x.season, x.mdeaths, x.hdeaths FROM exped x
    WHERE x.peakid="EVER" And x.mdeaths+x.hdeaths>0 And
        Between(x.year,"1995","1996")
```

and generates a list of death counts where each record represents one expedition that had either a member or hired death. From the table below, we see that there was one expedition in Spring 1995 with one hired death, three expeditions in Autumn 1995 with either a hired death or a member death, and multiple expeditions in Spring 1996 with numerous deaths, etc.

Year	Season	Mdeaths	Hdeaths
1995	1	0	1
1995	3	0	1
1995	3	0	1
1995	3	1	0
1996	1	4	0
1996	1	1	1
1996	1	1	0
1996	1	1	0
1996	1	1	0
1996	1	3	0
1996	3	1	0
1996	3	0	1
1996	3	0	1

The second query is used to total the number of deaths

```
SELECT SUM(x.mdeaths), SUM(x.hdeaths) FROM exped x
    WHERE x.peakid="EVER" And x.mdeaths+x.hdeaths>0 And
        Between(x.year,"1995","1996")
```

and produces the total death counts for 1995 and 1996

Sum_mdeaths	Sum_hdeaths
13	6

This may or may not be an interesting result, but it certainly is not the most useful result that can be obtained.

The aggregate functions produce only one row in the output table for each of the aggregated results. Hence if other field-names are in output field-list, the result may or may not be meaningful and in some cases not even correct. The third query adds “x.year, x.season” to the output list

```
SELECT x.year, x.season, SUM(x.mdeaths), SUM(x.hdeaths)
    FROM exped x WHERE x.peakid="EVER" And
        x.mdeaths+x.hdeaths>0 And Between(x.year,"1995","1996")
```

with the result

Year	Season	Sum_mdeaths	Sum_hdeaths
1996	3	13	6

Since the year and season were not aggregated, only the last value found appears in the result that is not particularly useful and also very misleading. Note: In Visual Foxpro 9, the above statement would be invalid.

To avoid this problem and still produce a meaningful result, we can use the GROUP BY clause that is specified as

```
GROUP BY group-list
```

where “group-list” is normally a list of table field names. This enables you to group the results of a query based on the values of one or more fields:

In our fourth query, we will group the totals by year and season:

```
SELECT x.year, x.season, Sum(x.mdeaths), Sum(x.hdeaths)
    FROM exped x WHERE x.peakid="EVER" And
        x.mdeaths+x.hdeaths>0 And Between(x.year,"1995","1996")
    GROUP BY x.year, x.season
```

This produces a much more meaningful and desirable result:

Year	Season	Sum_mdeaths	Sum_hdeaths
1995	1	0	1
1995	3	1	2
1996	1	11	1
1996	3	1	2

You will note that the column titles are normally the field name or a variation thereof. You can explicitly specify your own column titles by using the

```
AS column-name
```

clause in the SELECT statement. The column-name cannot contain blanks. Thus our previous query can be written as

```
SELECT x.year, x.season, Sum(x.mdeaths) AS member_deaths,
        Sum(x.hdeaths) AS hired_deaths
    FROM exped x WHERE x.peakid="EVER" And
        x.mdeaths+x.hdeaths>0 And Between(x.year,"1995","1996")
    GROUP BY x.year, x.season
```

which produces

Year	Season	Member_deaths	Hired_deaths
1995	1	0	1
1995	3	1	2
1996	1	11	1
1996	3	1	2

Aggregate functions can be used with multiple fields. Thus we can add another column for total deaths by specifying

```
SELECT x.year, x.season, Sum(x.mdeaths) AS member_deaths,
        Sum(x.hdeaths) AS hired_deaths,
        Sum(x.mdeaths+x.hdeaths) AS total_deaths
    FROM exped x WHERE x.peakid="EVER" And
        x.mdeaths+x.hdeaths>0 And Between(x.year,"1995","1996")
    GROUP BY x.year, x.season
```

which produces

Year	Season	Member_deaths	Hired_deaths	Total_deaths
1995	1	0	1	1
1995	3	1	2	3
1996	1	11	1	12
1996	3	1	2	3

When using the GROUP BY clause, all field values within each group must have the same value except for those being summed; otherwise incorrect results may occur. For example in the above query, if you add "x.host" to the SELECT field list, the results will be incorrect for each group with more than one value in the host field, unless "x.host" is also added to the GROUP BY list.

The HAVING clause which is specified by

```
HAVING condition
```

may be used further refine the output of queries that use the GROUP BY clause. For example, if in the above example, we only wanted the results for seasons that had multiple deaths, we could specify

```
SELECT x.year, x.season, Sum(x.mdeaths) AS member_deaths,
        Sum(x.hdeaths) AS hired_deaths,
        Sum(x.mdeaths+x.hdeaths) AS total_deaths
    FROM exped x WHERE x.peakid="EVER" And
        x.mdeaths+x.hdeaths>0 And Between(x.year,"1995","1996")
    GROUP BY x.year, x.season
    HAVING Sum(x.mdeaths+x.hdeaths) > 1
```

which produces

Year	Season	Member_deaths	Hired_deaths	Total_deaths
1995	3	1	2	3
1996	1	11	1	12
1996	3	1	2	3

If we were to use the SELECT statement

```
SELECT x.year, x.season, Sum(x.mdeaths) AS member_deaths,
        Sum(x.hdeaths) AS hired_deaths,
        Sum(x.mdeaths+x.hdeaths) AS total_deaths
    FROM exped x WHERE x.peakid="EVER" And
        x.mdeaths+x.hdeaths>0 And Between(x.year,"1995","1996")
        And x.mdeaths+x.hdeaths > 1
    GROUP BY x.year, x.season
```

the query potentially could give a different result since the "x.mdeaths+

x.hdeaths > 1” phrase would apply to individual expeditions in the database, not to the expeditions grouped by year and season.

The HAVING clause can be used without the GROUP BY clause to refine the output from a query, in which case it really acts like a WHERE clause.

The COUNT function can be used to give a quick total of the number of records in a query that match a specified condition. For example, to count the number of successful expeditions for all peaks in the 1990s, use the SQL statement

```
SELECT COUNT(*) FROM exped x
    WHERE (x.success1 Or x.success2 Or x.success3 Or x.success4) And
        Between(x.year,"1990","1999")
```

which gives the result

Cnt
915

Note the use of the parentheses around the “success” portion of the WHERE clause to force the order of expression evaluation. If the parentheses were omitted as in

```
SELECT COUNT(*) FROM exped x
    WHERE x.success1 Or x.success2 Or x.success3 Or x.success4 And
        Between(x.year,"1990","1999")
```

the query would incorrectly yield the result greater than 4800 since the SQL statement would be the equivalent of

```
SELECT COUNT(*) FROM exped x
    WHERE x.success1 Or x.success2 Or x.success3 Or
        (x.success4 And Between(x.year,"1990","1999"))
```

since the AND operator is normally evaluated before the OR operator.

The order of precedence for the logical operators is NOT, AND, and lastly OR, and within each of these three categories, the order of precedence is left to right. To override the order of precedence, parentheses must be used. To ensure that you are always getting the result you want, use parentheses liberally when in doubt.

To expand the above example to count all successful Everest expeditions in the 1990s, use the SELECT statement

```
SELECT COUNT(*) FROM exped x WHERE x.peakid="EVER" And
    ((x.success1 Or x.success2 Or x.success3 Or x.success4) And
    Between(x.year,"1990","1999"))
```

which gives a count of 168. Note the use of parentheses to force the expression evaluation to yield the correct result.

You could circumvent this precedence problem by using the following query:

```
SELECT COUNT(*) FROM exped x WHERE x.peakid="EVER" And
    Inlist(.T., x.success1, x.success2, x.success3, x.success4) And
        Between(x.year,"1990","1999")
```

The above discussion is only a brief introduction to SQL's special function and the GROUP BY and HAVING clauses. More complete discussions are given in other publications.

SQL Queries with Sub-Queries

An SQL query may be used to control the results of another query. Normally this is done by using a sub-query in the WHERE clause that is given in the form

```
SELECT field-list FROM table-list
    WHERE value =
        (SELECT field-list FROM table-list
            WHERE condition)
```

In this form, the SQL language specifies that the sub-query produce a single value that is passed to the primary query for evaluating the query condition.

For example, to search for the Annapurna I expeditions on which a person named Reinhold Messner summited you can use the query

```
SELECT x.peakid, x.year, x.season FROM exped x
    WHERE x.expid=
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess And m.peakid="ANN1")
```

which produces the result

Peakid	Year	Season
ANN1	1985	1

If the query were changed to

```
SELECT x.peakid, x.year, x.season FROM exped x
    WHERE x.expid=
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess)
```

the query would be erroneous and would generate an "Invalid SQL search command" error message since the logic of our expression requires a single value for "x.expid=" clause, while the sub-query produces more than one value because there are several expeditions on which Reinhold Messner summited. The query also would have been erroneous if Reinhold Messner had summited on more than one Annapurna I expedition.

To get around the one-value restriction, we can use the IN operator handle sub-queries that produce multiple values. Thus, we have

```
SELECT x.peakid, x.year, x.season FROM exped x
    WHERE x.expid IN
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess)
```

which produces the (partial) result

Peakid	Year	Season
ANN1	1985	1
DHA1	1985	1
LHOT	1986	3
MAKA	1986	3
CHOY	1983	1
KANG	1982	1
EVER	1980	2
MANA	1972	1
…	…	…

that gives all of the expeditions on which Reinhold Messner summited.

However, these results are not as useful as they could be. So we will add Messner's name to the results with a table-join using the query

```
SELECT x.peakid, x.year, x.season, m.fname, m.lname
    FROM exped x, members m
    WHERE x.expid=m.expid And x.expid IN
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess)
```

which produces the (partial) result

But this still is not quite yet what we want since the results include all other climbers on the same expeditions as Messner. To eliminate these unwanted climbers, we can change the query to

```
SELECT x.peakid, x.year, x.season, m.fname, m.lname
    FROM exped x, members m
    WHERE x.expid=m.expid And m.lname="Messner"
        And x.expid IN
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess)
```

which produces the (partial) result

Peakid	Year	Season	Fname	Lname
ANN1	1985	1	Reinhold	Messner
CHOY	1983	1	Reinhold	Messner
DHA1	1985	1	Reinhold	Messner
EVER	1980	2	Reinhold	Messner
KANG	1982	1	Reinhold	Messner
LHOT	1986	3	Reinhold	Messner
MAKA	1986	3	Reinhold	Messner
MANA	1972	1	Reinhold	Messner
…	…	…	…	…

We can dress up the result in a couple of ways. First, we will combine the first and last name columns, and second, order the result:

```
SELECT x.peakid, x.year, x.season,
        Trim(m.fname)+" "+ m.lname AS climber
    FROM exped x, members m
    WHERE x.expid=m.expid And m.lname="Messner"
        And x.expid IN
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess)
    ORDER BY x.year, x.season
```

The Visual FoxPro Trim function, removes the trailing blanks from the first name. Now the (partial) query result is

Peakid	Year	Season	Climber
TILI	1971	3	Reinhold Messner
MANA	1972	1	Reinhold Messner
MNPW	1977	1	Reinhold Messner
EVER	1978	1	Reinhold Messner
EVER	1980	2	Reinhold Messner
KANG	1982	1	Reinhold Messner
CHOY	1983	1	Reinhold Messner
ANN1	1985	1	Reinhold Messner
…	…	…	…

Now you may question why we went to all this trouble when a much simpler query will produce the same result:

```
SELECT x.peakid, x.year, x.season,
        Trim(m.fname)+" "+ m.lname AS climber
    FROM exped x, members m
    WHERE x.expid=m.expid And m.fname="Reinhold" And
        m.lname="Messner" And m.msuccess
    ORDER BY x.year, x.season
```

The answer will be apparent when we expand the last query to search for all expeditions on which both Reinhold Messner and Hans Kammerlander summited:

```
SELECT x.peakid, x.year, x.season,
        Trim(m.fname)+" "+ m.lname AS climber
    FROM exped x, members m
    WHERE x.expid=m.expid And m.fname="Reinhold" And
        m.lname="Messner" And x.expid IN
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess)
        And x.expid IN
        (SELECT m.expid FROM members m
            WHERE "Hans" $ m.fname And m.lname="Kammerlander"
                And m.msuccess)
    ORDER BY x.year, x.season
```

which produces the result

Peakid	Year	Season	Climber
CHOY	1983	1	Reinhold Messner
ANN1	1985	1	Reinhold Messner
DHA1	1985	1	Reinhold Messner
LHOT	1986	3	Reinhold Messner
MAKA	1986	3	Reinhold Messner

The above example also introduces the “$” operator that searches for an imbedded character string. In this case, since Hans Kammerlander’s complete name as given in the database is Johann (Hans) Kammerlander, we can search for “Hans” as the first name by using the expression

```
"Hans" $ m.fname
```

In order to place both climber’s names in the result, we need to alter the query again by using two sub-queries:

```
SELECT x.peakid, x.year, x.season,
        "Reinhold Messner & Hans Kammerlander" AS climbers
    FROM exped x, members m
    WHERE x.expid=m.expid And m.fname="Reinhold" And
        m.lname="Messner" And x.expid IN
        (SELECT m.expid FROM members m
            WHERE m.fname="Reinhold" And m.lname="Messner"
                And m.msuccess) And x.expid IN
        (SELECT m.expid FROM members m
            WHERE "Hans" $ m.fname And m.lname="Kammerlander"
                And m.msuccess)
        ORDER BY x.year, x.season
```

which produces the result

Peakid	Year	Season	Climbers
CHOY	1983	1	Reinhold Messner & Hans Kammerlander
ANNA1	1985	1	Reinhold Messner & Hans Kammerlander
DHA1	1985	1	Reinhold Messner & Hans Kammerlander
LHOT	1986	3	Reinhold Messner & Hans Kammerlander
MAKA	1986	3	Reinhold Messner & Hans Kammerlander

The above query illustrates the technique of placing the character string “Reinhold Messner & Hans Kammerlander” into the result.

This is also an example where a simple join between two tables would not produce the desired result since we are requiring another relationship in the Members table, that is, both Messner and Kammerlander summited the same peak on the same expedition. Using a simple query such as

```
SELECT x.peakid, x.year, x.season,
        Trim(m.fname)+" "+ m.lname AS climber
    FROM exped x, members m
    WHERE x.expid=m.expid And m.fname="Reinhold" And
        m.lname="Messner" And "Hans" $ m.fname And
        m.lname="Kammerlander" And m.msuccess
    ORDER BY x.year, x.season
```

would always produce an empty result since no single member record has the name of both Reinhold Messner and Hans Kammerlander.

To make the simpler form of the query work properly, we could join a duplicate copy of the Members table set the proper relationship for the second climber:

```
SELECT x.peakid, x.year, x.season,
        Trim(m1.fname)+" "+ m1.lname AS climber_1,
        Trim(m2.fname)+" "+ m2.lname AS climber_2
    FROM exped x, members m1, members m2
    WHERE x.expid=m1.expid And m1.fname="Reinhold" And
        m1.lname="Messner" And m1.msuccess And
        x.expid=m2.expid And "Hans" $ m2.fname And
        m2.lname="Kammerlander" And m2.msuccess
    ORDER BY x.year, x.season
```

This is an example where alias names must be used for the Member table in order to distinguish which copy is to be used for each part of the condition statement in the WHERE clause. The above query produces the result

Peakid	Year	Season	Climber_1	Climber_2
CHOY	1983	1	Reinhold Messner	Johann (Hans) Kammerlander
ANN1	1985	1	Reinhold Messner	Johann (Hans) Kammerlander
DHA1	1985	1	Reinhold Messner	Johann (Hans) Kammerlander
LHOT	1986	3	Reinhold Messner	Johann (Hans) Kammerlander
MAKA	1986	3	Reinhold Messner	Johann (Hans) Kammerlander

As you can see, there often is more than one way to construct a SQL query for a particular search. To gain a further understanding of these examples, you should consult one of the many books devoted to the SQL language.

Visual FoxPro 6 does have some restrictions on the use of sub-queries:

(1) only two sub-queries can be included in the WHERE clause of the primary query;
(2) sub-queries cannot be nested within other sub-queries;
(3) aggregate functions (such as COUNT, SUM, AVG, etc.) cannot be used in sub-queries; they are only allowed in the primary query.

The last restriction on the use of aggregate functions with sub-queries is particularly unfortunate as it greatly restricts the usefulness of correlated sub-queries (queries that refer to a table in the primary query).

For example in standard SQL, to search for all Americans that summited Everest more than once, the following SELECT statement could be used:

```
SELECT m1.fname, m1.lname, m1.citizen, m1.msmtdate1
    FROM members m1
    WHERE 1 <
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.peakid=m2.peakid And m1.citizen=m2.citizen And
                m1.msuccess=m2.msuccess And m2.peakid="EVER" And
                m2.citizen="USA" And m2.msuccess)
    ORDER BY m1.lname, m1.fname, m1.msmtdate1
```

But the version of SQL supported by Visual FoxPro 6 does not allow the use of the aggregate function COUNT(*) in the sub-query (this statement would be valid in Visual FoxPro 9). Instead, we must use an uncorrelated SELECT statement:

```
SELECT DISTINCT m1.fname, m1.lname, m1.citizen, m1.msmtdate1
    FROM members m1, members m2
    WHERE m1.expid <> m2.expid And
        m1.peakid = m2.peakid And
        m1.peakid ="EVER" And
        m1.fname = m2.fname And
        m1.lname = m2.lname And
        m1.citizen = m2.citizen And
        m1.citizen = "USA" And
        m1.msuccess And m2.msuccess
    ORDER BY m1.lname, m1.fname, m1.msmtdate1
```

which produces the (partial) result

Fname	Lname	Citizen	Msmtdate1
William Barkley (Bill)	Allen	USA	23/05/2010
William Barkley (Bill)	Allen	USA	20/05/2011
William Barkley (Bill)	Allen	USA	21/05/2016
Robert Mads	Anderson	USA	26/05/2003
Robert Mads	Anderson	USA	23/05/2010
Conrad Daniel	Anker	USA	17/05/1999
Conrad Daniel	Anker	USA	14/06/2007
Conrad Daniel	Anker	USA	26/05/2012
Melissa Sue	Arnot	USA	22/05/2008
Melissa Sue	Arnot	USA	23/05/2009
Melissa Sue	Arnot	USA	23/05/2010
...	…	…	…

To search for Americans that summited Everest more than twice, it would be a simple change to the correlated sub-query (changing "WHERE 1 <" to "WHERE < 2"), but the necessary modification to the uncorrelated query would be

```
SELECT DISTINCT m1.fname, m1.lname, m1.citizen, m1.msmtdate1
    FROM members m1, members m2, members m3
    WHERE m1.expid <> m2.expid And m1.expid <> m3.expid And
        m2.expid <> m3.expid And
        m1.peakid = m2.peakid And m1.peakid = m3.peakid And
        m1.peakid ="EVER" And
        m1.fname = m2.fname And m1.fname = m3.fname And
        m1.lname = m2.lname And m1.lname = m3.lname And
        m1.citizen = m2.citizen And m1.citizen = m3.citizen And
        m1.citizen = "USA" And
        m1.msuccess And m2.msuccess And m3.msuccess
    ORDER BY m1.lname, m1.fname, m1.msmtdate1
```

which is a more complex statement than the correlated version. Even more complex would be the SELECT statement to search for Americans that summited Everest more than three times, etc.

To obtain only a list of the Americans that summited Everest more than once without their summit dates or a count number of times each one summited, a simple query using the GROUP BY and HAVING clauses may be used:

```
SELECT m.fname, m.lname, m.citizen
    FROM members m
    WHERE m.peakid = "EVER" And
         m.citizen = "USA" And m.msuccess
    GROUP BY m.lname, m.fname, m.citizen
    HAVING COUNT(*) > 1
    ORDER BY m.lname, m.fname
```

The GROUP BY clause groups together the summit records for each climber and the HAVING clause selects out those climbers with more than one summit success. The COUNT(*) function in this context counts the records in each group. The (partial) output for the above query is

Fname	Lname	Citizen
William Barkley (Bill)	Allen	USA
Robert Mads	Anderson	USA
Conrad Daniel	Anker	USA
Melissa Sue	Arnot	USA
Peter George (Pete)	Athans	USA
Neal Jay	Beidleman	USA
Damian	Benagas	USA
Guillermo (Willie)	Benegas	USA
Luis Guillermo	Benitez	USA
Wallace Wayne (Wally)	Berg	USA
Julio J.	Bird	USA
Brent Russell	Bishop	USA
Christine Joyce Feld	Boskoff	USA
…	…	…

Only one record per group is included in the output of the SELECT statement, so this statement cannot list all of the summit dates or the summit count for each climber as was done in the preceding examples.

In a later section, we will illustrate how to use the results of this query to obtain the results that we really want.

Special SQL Operators with Sub-Queries

The SQL language has several special operators that always take sub-queries as arguments:

EXISTS sub-query	returns true if the sub-query produces any results; false if it does not.

value = ANY sub-query	returns true if any of the results of the sub-query are equal to “value.”
value = ALL sub-query	returns true if all of the results of the sub-query are equal to “value.”

Of these three, EXISTS is probably the most useful. The NOT operator may be combined with these special operators.

The following paragraphs build an example in two steps that searches for all-women’s expeditions on which an American woman summited. The first step searches for any expedition on which an American woman summited:

```
SELECT DISTINCT x.peakid, x.year, x.season, x.nation, x.leaders,
        Trim(m1.fname)+" "+ m1.lname AS climber
    FROM exped x, members m1
    WHERE x.expid=m1.expid And m1.msuccess And
        Upper(m1.citizen)="USA" And m1.sex="F"
    ORDER BY x.year, x.season
```

The first 10 records produced in the output are

Peakid	Year	Season	Nation	Leaders	Climber
URKM	1974	3	USA	Bill Roos	Judy Rearick
ANN1	1978	3	USA	Arlene Blum	Irene Miller
HIUP	1981	3	USA	Eric Simonson	Bonnie M. Nobori
HIUP	1981	3	USA	Eric Simonson	Laverne G. Woods
PUMO	1981	4	USA	Ned Gillette	Jan Reynolds
AMAD	1982	1	USA	Sue Giller	Anne Macquaire
AMAD	1982	1	USA	Sue Giller	Jineen (Jini) Griffiths
AMAD	1982	1	USA	Sue Giller	Lucylle (Lucy) Smith
AMAD	1982	1	USA	Sue Giller	Sharon (Shari) Kearney
AMAD	1982	1	USA	Sue Giller	Stacy Allison
…	…	…	…	…	…

The second step uses the EXISTS operator combined with the NOT operator to restrict the output further to include only all-women’s expeditions:

```
SELECT DISTINCT x.peakid, x.year, x.season, x.nation, x.leaders,
        Trim(m1.fname)+" "+ m1.lname AS climber
    FROM exped x, members m1
    WHERE x.expid=m1.expid And m1.msuccess And
        Upper(m1.citizen)="USA" And m1.sex="F" And
        NOT EXISTS
            (SELECT * FROM members m2
                WHERE x.expid=m2.expid And m2.sex="M" And Not
                    m2.hired And Not m2.nottobc And Not m2.bconly)
    ORDER BY x.year, x.season
```

The sub-query selects all records from each expedition that had non-hired males that went above base camp. Using only the EXISTS operator, the full SQL statement would include these expeditions; using NOT EXISTS, the full statement eliminates these expeditions.

The final 13 records of output produced by the query is

Peakid	Year	Season	Nation	Leaders	Climber
ANN1	1978	3	USA	Arlene Blum	Irene Miller
AMAD	1982	1	USA	Sue Giller	Anne Macquaire
AMAD	1982	1	USA	Sue Giller	Jineen (Jini) Griffiths
AMAD	1982	1	USA	Sue Giller	Lucylle (Lucy) Smith
AMAD	1982	1	USA	Sue Giller	Sharon (Shari) Kearney
AMAD	1982	1	USA	Sue Giller	Stacy Allison
AMAD	1982	1	USA	Sue Giller	Susan Ann (Sue) Giller
AMAD	1982	1	USA	Sue Giller	Susan H. Havens
CHOY	1999	1	USA	Amy (Supy) Bullard	Amy (Supy) Bullard
CHOY	1999	1	USA	Amy (Supy) Bullard	Georgie Wilmerding Stanley
CHOY	1999	1	USA	Amy (Supy) Bullard	Kathryn Miller Hess
AMAD	2003	3	USA	Angela Hawse	Angela Jo Hawse
AMAD	2003	3	USA	Angela Hawse	Eleanor K. (Ellie) Pryor
…	…	…	…	…	…

The following example combines both EXISTS and NOT EXISTS to search for all Everest expeditions on which only the hired members summited and produces the accompanying output:

```
SELECT DISTINCT x.peakid, x.year, x.season, x.nation, x.leaders
    FROM exped x
    WHERE x.peakid = "EVER"
        And EXISTS
            (SELECT * FROM members m
                WHERE x.expid=m.expid And m.msuccess And m.hired)
        And NOT EXISTS
            (SELECT * FROM members m
                WHERE x.expid=m.expid And m.msuccess And Not m.hired)
    ORDER BY x.year, x.season
```

Peakid	Year	Season	Nation	Leaders
EVER	1991	1	UK	Harold Taylor
EVER	1995	1	New Zealand	Rob Hall
EVER	1998	1	S Africa	Ian Woodall
EVER	2000	1	USA	Vernon Tejas
EVER	2003	1	Belgium	Robert Huygh
EVER	2003	1	Spain	Jesus Elena Vera
EVER	2003	1	USA	Bill Crouse
EVER	2005	1	S Korea	Um Hong-Gil
EVER	2007	1	Pakistan	Muhammed Faizan
EVER	2009	1	France	Marc Batard
…	…	…	…	…

The following example searches for all Everest summiters that have summited from both sides of the mountain:

```
    SELECT DISTINCT Trim(m1.fname)+" "+m1.lname AS name,
            m1.citizen, m1.yob
        FROM exped x, members m1
        WHERE x.expid=m1.expid And m1.peakid="EVER" And
            x.host=1 And m1.msuccess And EXISTS
            (SELECT * FROM exped x, members m2
                WHERE x.expid=m2.expid And m1.fname=m2.fname And
                    m1.lname=m2.lname And m1.yob=m2.yob And
                    m2.peakid="EVER" And x.host=2 And m2.msuccess)
        ORDER BY name
```

The main query searches for all south-side Everest summiters (x.host=1) and then uses the EXISTS operator with a sub-query to search the result for those that also summited from the north side (x.host=2). Two different aliases (m1 and m2) for the Member table are required since Member fields in the sub-query are compared to Member fields in the main query.

The DISTINCT keyword is needed so that each summiter is listed only once. Note that two output lines are given for Anatoli Boukreev since he was a citizen of two different countries while he was summiting Everest. We could have eliminated the second Boukreev line if "m.citizen" was not included in the query output list, but without that column the table might be less useful.

The output produced is

Name	Citizen	Yob
Abele Blanc	Italy	1954
Abudul Khalim (Abu) Elmezov	Russia	1957
Aldo Hiram Valencia Corona	Mexico	1978
Alexander (Alex) Abramov	Russia	1964
Alexia Zelda Cecile Zuberer	Switzerland	1972
Alf Robin Trygg	Sweden	1986
Ali Nasuh Mahruki	Turkey	1968
Amar Prakash Dogra	India	1963
Anatoli Boukreev	Kazakhstan	1958
Anatoli Boukreev	USSR	1958
Andre Victor Bredenkamp	S Africa	1957
Andrew Atis (Andy) Lapkass	USA	1958
Ang Babu (Jimba Zangbu) Sherpa	Nepal	1974
Ang Chhiring (Ang Tshering) Sherpa	Nepal	1952
Ang Dawa (Dawa) Tamang	Nepal	1972
Ang Dawa Sherpa	Nepal	1964
Ang Dawa Sherpa	Nepal	1978
Ang Dawa Sherpa	Nepal	1982
Ang Dawa Sherpa	Nepal	1984
Ang Gelu Sherpa	Nepal	1970
Ang Gelu Sherpa	Nepal	1986
…	…	…

Combining Multiple Queries with the UNION Clause

The UNION clause is used to merge the output from two or more queries into a single result. The form for merging two queries is

```
query1 UNION query2
```

and for three queries is

```
query1 UNION query2 UNION query3
```

The output from each of the queries must be union-compatible, that is, each query must specify the same number of columns in the output and each column must be of the same type in all of the queries.

The following paragraphs build an example that searches for the youngest and oldest summiters on Everest by combining two queries. The first query searches for summiters that are 15 years old or less:

```
SELECT m.peakid, m.myear, m.mseason, m.calcage AS age,
        Trim(m.fname)+" "+ m.lname AS climber
    FROM members m
    WHERE m.peakid="EVER" And m.msuccess And
        Between(m.calcage,1,15)
    ORDER BY m.calcage
```

The Between function is used to search for ages between 1 and 15 in order to exclude those climbers with an age of 0 (age unknown). The output from the query is

Peakid	Myear	Mseason	Age	Climber
EVER	2010	1	13	Jordan Romero
EVER	2014	1	13	Malavath Poorna
EVER	2003	1	15	Mingkipa Sherpa
EVER	2013	1	15	Raghav Joneja

The second query searches for summiters that are 70 years old or more:

```
SELECT m.peakid, m.myear, m.mseason, m.calcage AS age,
        Trim(m.fname)+" "+ m.lname AS climber
    FROM members m
    WHERE m.peakid="EVER" And m.msuccess And m.calcage >= 70
    ORDER BY m.calcage
```

The output from this query is:

Peakid	Myear	Mseason	Age	Climber
EVER	2003	1	70	Yuichiro Miura
EVER	2006	1	70	Takao Arayama
EVER	2009	1	70	Nikolai Dmitrievich Cherny
EVER	2007	1	71	Katsusuke Yanagisawa
EVER	2011	1	71	Tatsuo Matsumoto
EVER	2014	1	72	William Mitchell (Bill) Burke
EVER	2012	1	73	Tamae Watanabe
EVER	2008	1	75	Yuichiro Miura
EVER	2008	1	76	Min Bahadur Sherchan
EVER	2013	1	80	Yuichiro Miura

The UNION clause now combines these two queries into a single query:

```
SELECT m.peakid, m.myear, m.mseason, m.calcage AS age,
       Trim(m.fname)+" "+ m.lname AS climber
  FROM members m
  WHERE m.peakid="EVER" And m.msuccess And
       Between(m.calcage,1,15)

 UNION

 SELECT m.peakid, m.myear, m.mseason, m.calcage AS age,
       Trim(m.fname)+" "+ m.lname AS climber
  FROM members m
  WHERE m.peakid="EVER" And m.msuccess And m.calcage >= 70

 ORDER BY 4
```

The output from the combined query is:

Peakid	Myear	Mseason	Age	Climber
EVER	2010	1	13	Jordan Romero
EVER	2014	1	13	Malavath Poorna
EVER	2003	1	15	Mingkipa Sherpa
EVER	2013	1	15	Raghav Joneja
EVER	2003	1	70	Yuichiro Miura
EVER	2006	1	70	Takao Arayama
EVER	2009	1	70	Nikolai Dmitrievich Cherny
EVER	2007	1	71	Katsusuke Yanagisawa
EVER	2011	1	71	Tatsuo Matsumoto
EVER	2014	1	72	William Mitchell (Bill) Burke
EVER	2012	1	73	Tamae Watanabe
EVER	2008	1	75	Yuichiro Miura
EVER	2008	1	76	Min Bahadur Sherchan
EVER	2013	1	80	Yuichiro Miura

The ORDER BY clause must use column numbers instead of field names since there no requirement by SQL that the column names be the same for each of the queries. The above example orders the output in ascending order by the 4th column.

Also when you enter the query in the query dialog, make sure that UNION and other keywords have blanks separating them from the rest of the query text.

Using the Results of One Query for a Subsequent Query

The output from a query may be saved in a database table instead of being displayed in a browse window by using the INTO clause

```
INTO TABLE output-table-name
```

Earlier we gave an example to list all American climbers that summited Everest more than once. We now modify this example to search for American climbers that summited ten or more times and to save the output in the table EVER10:

```
SELECT m.fname, m.lname, m.citizen
    FROM members m
    INTO TABLE EVER10
    WHERE m.peakid = "EVER" And
        m.citizen = "USA" And m.msuccess
    GROUP BY m.lname, m.fname, m.citizen
    HAVING COUNT(*) >= 10
    ORDER BY m.lname, m.fname
```

The output produced is

Fname	Lname	Citizen
Guillermo (Willie)	Benegas	USA
David Allen (Dave)	Hahn	USA
Vernon Edward (Vern)	Tejas	USA

We can now use this table in a sub-query to a SQL statement that lists all American climbers that summited Everest six times or more:

```
SELECT m.fname, m.lname, m.citizen, m.msmtdate1
    FROM members m
    WHERE m.peakid ="EVER" And
        m.citizen = "USA" And
        m.msuccess And EXISTS
        (SELECT * FROM EVER10 q
            WHERE q.fname = m.fname And
                q.lname = m.lname And
                q.citizen = m.citizen)
    ORDER BY m.lname, m.fname, m.msmtdate1
```

The output produced is

Fname	Lname	Citizen	Msmtdate1
Guillermo (Willie)	Benegas	USA	12/05/1999
Guillermo (Willie)	Benegas	USA	23/05/2001
Guillermo (Willie)	Benegas	USA	16/05/2002
Guillermo (Willie)	Benegas	USA	17/05/2004
Guillermo (Willie)	Benegas	USA	30/05/2005
Guillermo (Willie)	Benegas	USA	16/05/2007
Guillermo (Willie)	Benegas	USA	21/05/2008
Guillermo (Willie)	Benegas	USA	19/05/2009
Guillermo (Willie)	Benegas	USA	23/05/2010
Guillermo (Willie)	Benegas	USA	25/05/2012
David Allen (Dave)	Hahn	USA	19/05/1994
David Allen (Dave)	Hahn	USA	17/05/1999
David Allen (Dave)	Hahn	USA	22/05/2000
David Allen (Dave)	Hahn	USA	30/05/2003
David Allen (Dave)	Hahn	USA	20/05/2004
David Allen (Dave)	Hahn	USA	30/05/2005
David Allen (Dave)	Hahn	USA	23/05/2006
David Allen (Dave)	Hahn	USA	18/10/2006
David Allen (Dave)	Hahn	USA	21/05/2007
David Allen (Dave)	Hahn	USA	27/05/2008
David Allen (Dave)	Hahn	USA	23/05/2009
David Allen (Dave)	Hahn	USA	25/05/2010
David Allen (Dave)	Hahn	USA	21/05/2011
David Allen (Dave)	Hahn	USA	26/05/2012
David Allen (Dave)	Hahn	USA	23/05/2013
Vernon Edward (Vern)	Tejas	USA	12/05/1992
Vernon Edward (Vern)	Tejas	USA	25/05/2002
Vernon Edward (Vern)	Tejas	USA	30/05/2003
Vernon Edward (Vern)	Tejas	USA	24/05/2004
Vernon Edward (Vern)	Tejas	USA	30/05/2005
Vernon Edward (Vern)	Tejas	USA	20/05/2006
Vernon Edward (Vern)	Tejas	USA	22/05/2007
Vernon Edward (Vern)	Tejas	USA	24/05/2008
Vernon Edward (Vern)	Tejas	USA	24/05/2010
Vernon Edward (Vern)	Tejas	USA	22/05/2013

Since the output of the above query contains a date field, if you save the output to an Excel file, you will be prompted for the date format (Macintosh or Windows) because Excel uses different base dates in the two platforms.

If the first and second queries are done in the same Himal session, the EVER10 table is automatically available. If the second query is done at later time, you will be prompted to locate the EVER10 table (usually it will be found in the Himalayan Database folder).

Special Visual FoxPro Functions and Operators

The following Visual FoxPro program functions can be used in SELECT statements or to enhance the appearance of the output:

Trim(…)	removes trailing blanks from "…"
Upper(…)	converts to "…" to uppercase
Left(…,n)	returns leftmost "n" characters of "…"

The "==" operator forces an exact match for a character string comparison. For example, the expression

```
m.lname = "Hunt"
```

matches Hunt, Hunter, Huntington, etc., while the expression

```
m.lname == "Hunt"
```

matches only Hunt and is equivalent to

```
Left(m.lname,4) = "Hunt"
```

The "==" operator can be used in SQL statement conditions as well as conditional statements for the Browse, Search and Export commands.

Special Himal Functions

The following Himal program functions can be used in SELECT statements to enhance the appearance of the output by translating the underlying numeric field codes to their character definitions:

Season(…)	translates expedition or member season codes to "Spring", "Summer", "Autumn" and "Winter"
Host(…)	translates expedition host codes to "Nepal", "China" and "India"
Phost(…)	translates peak host codes
Reason(…)	translates expedition termination codes
Death(…)	translates member death codes
DeathClass(…)	translates member death class codes
Injury(…)	translates member injury codes
MAboveBC(...)	returns the number of members above base camp for an expedition
GetHost1(expid)	returns expedition host code as N, C or I
GetHost2(expid)	returns expedition host code as 1, 2 or 3

The argument "…" for each of the above functions is the relevant table field-name. The definitions for the above numeric field codes are given with the table structure descriptions in Appendix B.

For example, our first query could be rewritten as

```
SELECT x.year, Season(x.season) AS season, x.mdeaths, x.hdeaths
    FROM exped x
    WHERE x.peakid="EVER" And x.mdeaths+x.hdeaths>0 And
        Between(x.year,"1995","1996")
```

and would produce the output

Year	Season	Mdeaths	Hdeaths
1995	Spring	0	1
1995	Autumn	0	1
1995	Autumn	0	1
1995	Autumn	1	0
1996	Spring	4	0
1996	Spring	1	1
1996	Spring	1	0
1996	Spring	1	0
1996	Spring	1	0
1996	Spring	3	0
1996	Autumn	1	0
1996	Autumn	0	1
1996	Autumn	0	1

The Season function is used with the "AS season" keyword to avoid the internally generated column title of Exp_2.

As another example, we can use the following query to obtain the number of members that went above base camp for all Everest expeditions between 1950 and 1960:

```
SELECT x.expid, x.nation, x.leaders, x.totmembers,
        mabovebc(x.expid) AS "memabvbc"
    FROM exped x
    WHERE x.peakid="EVER" And Between(x.year,"1950","1960")
    ORDER BY x.expid
```

This produces the output:

Expid	Nation	Leaders	Totmembers	Memabvbc
EVER50301	USA	Charles Houston	5	0
EVER51101	Denmark	Klavs Becker-Larsen	1	1
EVER51301	UK	Eric Shipton	6	6
EVER52101	Switzerland	Edouard Wyss-Dunant	11	9
EVER52301	Switzerland	Gabriel Chevalley	7	7
EVER52302	USSR	Pawel Datschnolian	40	0
EVER53101	UK	John Hunt	13	13
EVER56101	Switzerland	Albert Eggler	11	11
EVER58101	China	Xu Jing, Yevgeniy Beletski	13	0
EVER60101	India	Gyan Singh	22	17
EVER60102	China	Shi Zhang-Chun	29	29

We can eliminate the four expeditions that did not attempt the climb by adding the HAVING clause to the query:

```
SELECT x.expid, x.nation, x.leaders, x.totmembers,
        mabovebc(x.expid) AS "memabvbc"
    FROM exped x
    WHERE x.peakid="EVER" And Between(x.year,"1950","1960")
    HAVING mabovebc(x.expid) > 0
    ORDER BY x.expid
```

This removes the two expeditions that had no members above base camp.

The GROUP BY clause can be used to aggregate the counts by year:

```
SELECT x.year, Sum(x.totmembers),
        Sum(mabovebc(x.expid)) AS "sum_memabvbc"
    FROM exped x
    WHERE x.peakid="EVER" And Between(x.year,"1950","1960")
    GROUP BY x.year
    HAVING Sum(mabovebc(x.expid)) > 0
    ORDER BY x.year
```

which produces the result

Year	Sum_totmembers	Sum_memabvbc
1951	7	7
1953	13	13
1956	11	11
1960	51	46

The “x.expid, x.nation, x.leaders, x.totmembers” fields must not be included in the SELECT field list because the GROUP BY clause requires that each of the fields within a group have the same value except for those fields that are being summed or define the group boundaries; otherwise incorrect results may occur.

Note that in Visual FoxPro 9 the HAVING clause must be expressed as

```
HAVING Sum(mabovebc(x.expid)) > 0
```

where the Sum function is used to correspond to Sum function in the field-list. In Visual FoxPro 6, the syntax of the clause is more relaxed and allows

```
HAVING mabovebc(x.expid) > 0
```

The final example uses the GetHost2 function to select the members that summited Everest from both the north and south sides in the same season and returns one record for David Liano Gonzalez (as of 2016):

```
SELECT Trim(m1.fname)+" "+Trim(m1.lname) AS name, m1.citizen,
        m1.residence, m1.calcage, m1.myear, m1.mseason, m1.msmtdate1
         AS EverestS, m2.msmtdate1 AS EverestN
    FROM members m1, members m2
    WHERE m1.myear=m2.myear And m1.mseason=m2.mseason
        And m1.peakid="EVER" And Gethost2(m1.expid)=1
        And m2.peakid="EVER" And Gethost2(m2.expid)=2
        And m1.msuccess And m2.msuccess
        And m1.fname=m2.fname And m1.lname=m2.lname
        And m1.yob=m2.yob And m1.residence=m2.residence
    ORDER BY m1.myear, m1.mseason
```

Notes on Visual FoxPro and Excel Date Formats

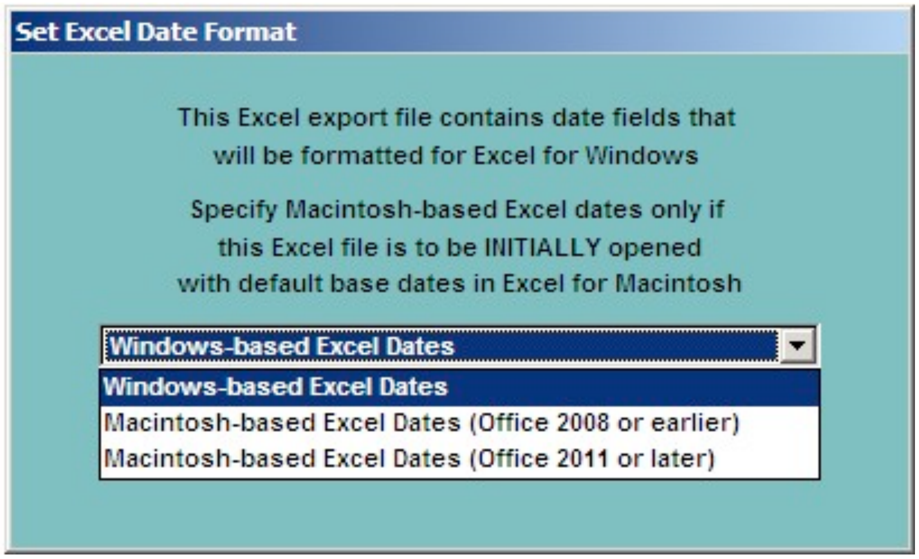
Dates in Visual FoxPro and Microsoft Excel are stored internally in Julian format that is defined as the number of days from a fixed base date. The base dates (Julian day 0) used are:

Microsoft Visual FoxPro	September 14, 1752
Microsoft Excel for Windows	January 0, 1900
Microsoft Excel for Macintosh (Office 2008 or earlier)	January 1, 1904
(a 1462-day difference between versions of Excel)	
Microsoft Excel for Macintosh (Office 2011 or earlier)	January 0, 1900

Excel for Macintosh in Office 2008 or earlier had a different base date because it was trying to avoid the problem that 1900 was not a leap year. To calculate a leap year, the year must be divisible by 4, but not 100, with the exception that it is a leap year when divisible by 400 (thus 1900 and 2100 are not leap years, but 2000 is a leap year). But this difference in base dates caused other issues when tying to reconcile dates in Excel sheets between the Windows and the Macintosh operating systems. With Excel for Macintosh in Office 2011 and later, both versions of Excel became the same with regards to the base dates.

This difference in Julian base dates can cause certain problems when exporting data containing dates from the Himalayan Database to Excel files.

If you are working entirely within the Windows operating systems (the normal case for most users), then the Julian base date conversions are handled automatically between Visual FoxPro and Excel. If your export file contains any date fields, you are prompted to specify the base date format:


Set Excel Date Format

This Excel export file contains date fields that will be formatted for Excel for Windows

Specify Macintosh-based Excel dates only if this Excel file is to be INITIALLY opened with default base dates in Excel for Macintosh

Windows-based Excel Dates
Windows-based Excel Dates
Macintosh-based Excel Dates (Office 2008 or earlier)
Macintosh-based Excel Dates (Office 2011 or later)

Dedicated Windows users should use the default Windows-based date format choice and just click the OK button (Macintosh users will see a corresponding screen that defaults to the Macintosh-based date format). Visual FoxPro then exports an unmarked file with dates that use the specified Julian base date format (the file is unmarked as to which format is being used).

When Excel opens the file for the first time, it marks the file as to the base date format assumed: Excel for Windows assumes Windows-based dates and Excel for Macintosh assumes Macintosh-based dates (the file must be saved to retain the marking.)

The file may be reopened at a later time in either version of Excel (Windows or Macintosh) and the date fields will be properly adjusted for the 1462-day difference in base dates since the base date format was marked on the first opening. Excel for Windows adjusts for Macintosh-based dates, and Excel for Macintosh adjusts for Window-based dates.

However, if you work with both the Windows and Macintosh operating systems, then the base date issue becomes more complex. When you export Excel files from Visual FoxPro, you must select the correct base date format for the version of Excel that you plan to use for the first opening of the file. If you are running the Windows version of the Himal program and wish to open the file first in Excel for Macintosh, you must select one of the Macintosh-based dates depending upon the version of Office/Excel you are using; if you are running the Macintosh version of the Himal program and wish to open the file first in Excel for Windows, you must select Windows-based dates.

For those users that are running the Windows version of the Himal program on an Intel-based Macintosh using Parallels (or Fusion), there are additional considerations. You have the option of installing and using Excel for Windows in your Parallels Window-partition, or having Parallels invoke Excel for Macintosh from your Macintosh partition while running in your Windows environment. If you wish to use Excel for Windows, you must select Windows-based dates. If you wish to use Excel for Macintosh, you must select Macintosh-based dates and also move the file from the Windows partition to the Macintosh partition before first opening it, since Excel for Macintosh cannot properly open in the Windows partition files exported from Visual FoxPro that contain dates (the error message "File format not valid" is displayed and the file cannot be opened).

For those users that are running the Windows version of the Himal program on an Intel-based Macintosh using CrossOver or WineBottler, you should select one of the two Macintosh-based dates depending upon the version of Excel for Macintosh you are using (Office 2008 or earlier, or Office 2011 or later).

Additional SQL Features Available for Visual FoxPro 6

With newer versions of Visual FoxPro, enhancements have been added to SQL support. The following SQL92 language enhancements have been added to Visual FoxPro 6 and later and are available to the SELECT command in the original (CD-Rom) VFP-6 Windows version 1.x of *The Himalayan Database*.

The simplest form of the SELECT statement placed into the WHERE clause the join-conditions that described how multiple tables are linked:

```
SELECT field-list FROM table-list
    WHERE join-conditions AND search-conditions
    ORDER BY order-list
```

But now these "join-conditions" may be placed into the FROM clause instead

```
SELECT field-list FROM table-list join-conditions
    WHERE search-conditions
    ORDER BY order-list
```

which helps to give greater clarity to the SELECT command and also offers some additional capability.

For example, the earlier two-table SQL example that searched for all climbers over age 50 that have summited an 8000m peak

```
SELECT p.peakid, p.pkname, m.lname, m.fname, m.citizen, m.calcage
    FROM members m, peaks p
    WHERE m.peakid=p.peakid And m.msuccess And
        m.calcage>=50 And p.heightm>=8000
```

could be rewritten using the JOIN-ON clause as

```
SELECT p.peakid, p.pkname, m.lname, m.fname, m.citizen, m.calcage
    FROM members m
        JOIN peaks p ON m.peakid=p.peakid
    WHERE m.msuccess And m.calcage>=50 And p.heightm>=8000
```

and the two-table SQL example that searched for all American and Canadian women that attempted Everest before the year 2000 from the Nepal side

```
SELECT m.peakid, x.host, m.fname, m.lname, m.citizen
    FROM members m, exped x
    WHERE m.expid=x.expid And m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA") And
        m.myear<"2000" And x.host=1 And m.peakid="EVER"
    ORDER BY m.myear, m.mseason
```

could be rewritten as

```
SELECT m.peakid, x.host, m.fname, m.lname, m.citizen
    FROM members m JOIN exped x ON m.expid=x.expid
    WHERE m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA") And
        m.myear<"2000" And x.host=1 And m.peakid="EVER"
    ORDER BY m.myear, m.mseason
```

and the three-table example that substituted the peak name into the result of the query instead of the peak ID

```
SELECT p.pkname, x.host, m.fname, m.lname, m.citizen
    FROM members m, peaks p, exped x
    WHERE m.peakid=p.peakid And m.expid=x.expid And m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA") And
        m.myear<"2000" And x.host=1 And m.peakid="EVER"
    ORDER BY m.myear, m.mseason
```

could be rewritten as

```
SELECT p.pkname, x.host, m.fname, m.lname, m.citizen
    FROM members m
        JOIN peaks p ON m.peakid=p.peakid
        JOIN exped x ON m.expid=x.expid
    WHERE m.sex="F" And
        Inlist(Upper(m.citizen),"USA","CANADA") And
        m.myear<"2000" And x.host=1 And m.peakid="EVER"
    ORDER BY m.myear, m.mseason
```

In each of the JOIN-ON variations given above, the parent table Members is specified first after the FROM keyword and the child tables, Exped and/or Peaks, are specified after the JOIN keywords. In the last case where there are two child tables, both joins are specified sequentially. The join conditions could also be given as

```
    FROM peaks p
        JOIN members m ON m.peakid=p.peakid
        JOIN exped x ON m.expid=x.expid
```

but not as

```
FROM peaks p
    JOIN exped x ON m.expid=x.expid
    JOIN members m ON m.peakid=p.peakid
```

since “m.expid” cannot be used in the first JOIN before “members m” is specified in the second JOIN.

In all of the above cases, whether specifying the join-conditions by using the WHERE clause or by using the JOIN-ON clause, the joins are *inner joins* which means that the matching values must be contained in both of the joined tables.

The full syntax is actually specified as

```
FROM members m
    INNER JOIN exped x ON m.expid=x.expid
```

but since the INNER keyword is the default, it is usually never given.

The additional capabilities alluded to above allow the specification of *outer joins* where all records from one table (whether they match or not with records in the other table) are included in the query result. A *left outer join* includes all records of the table to the left of the JOIN keyword, while a *right outer join* includes all records of the table to the right of the JOIN keyword; they are specified as

```
LEFT OUTER JOIN (or just LEFT JOIN)
RIGHT OUTER JOIN (or just RIGHT JOIN)
```

The concept of outer joins is best illustrated by the following series of three examples that generate lists of expeditions with decedents from the autumn 1984 Everest expeditions that had fatalities. The first example using inner joins shows three equivalent ways to generate a simple list:

(1)
```
SELECT x.expid, x.nation, x.leaders,
        Trim(m.fname)+" "+ m.lname AS name
    FROM exped x, members m
    WHERE x.expid=m.expid And x.peakid="EVER" And
        x.year="1984" And x.season=3 And m.death
    ORDER BY x.expid
```

(2)
```
SELECT x.expid, x.nation, x.leaders,
        Trim(m.fname)+" "+ m.lname AS name
    FROM exped x
        JOIN members m ON x.expid=m.expid
    WHERE x.peakid="EVER" And x.year="1984" And
        x.season=3 And m.death
    ORDER BY x.expid
```

```
(3)     SELECT x.expid, x.nation, x.leaders,
                Trim(m.fname)+" "+ m.lname AS name
            FROM exped x
                JOIN members m ON x.expid=m.expid And m.death
            WHERE x.peakid="EVER" And x.year="1984" And x.season=3
            ORDER BY x.expid
```

The result of query (given below) lists only those expeditions that had fatalities:

Expid	Nation	Leaders	Name
EVER84302	Nepal	Yogendra Thapa	Yogendra Bahadur Thapa
EVER84302	Nepal	Yogendra Thapa	Ang Dorje Sherpa
EVER84304	New Zealand	Peter Hillary	William Robert (Fred) From
EVER84304	New Zealand	Peter Hillary	Craig Rupert Nottle
EVER84306	Czechoslovakia	Frantisek Kele	Jozef Psotka

The third variation moves the "m.death" condition from the WHERE clause to the ON predicate of the JOIN-ON clause. When the death condition is included with the WHERE clause, it applies to all tables in the SELECT command, but when it is included with the ON predicate, it only applies to the Members table. For an inner join, there is no difference in the result, but it does make a difference when using outer joins. Thus we can rewrite the last example using a left outer join as

```
        SELECT x.expid, x.nation, x.leaders,
                Trim(m.fname)+" "+ m.lname AS name
            FROM exped x
                LEFT JOIN members m ON x.expid=m.expid And m.death
            WHERE x.peakid="EVER" And x.year="1984" And x.season=3
            ORDER BY x.expid
```

to produce an expanded table that also lists the expeditions that did not have fatalities:

Expid	Nation	Leaders	Name
EVER84301	Australia	Geoffrey Bartram	.NULL.
EVER84302	Nepal	Yogendra Thapa	Yogendra Bahadur Thapa
EVER84302	Nepal	Yogendra Thapa	Ang Dorje Sherpa
EVER84303	USA	Lou Whittaker	.NULL.
EVER84304	New Zealand	Peter Hillary	William Robert (Fred) From
EVER84304	New Zealand	Peter Hillary	Craig Rupert Nottle
EVER84305	Netherlands	Herman Plugge	.NULL.
EVER84306	Czechoslovakia	Frantisek Kele	Jozef Psotka

The .NULL. result is inserted into the name field of the rows with no matches, and when exported to Excel, they appear as blank cells.

Additional SQL Features Available for Visual FoxPro 9

The following SQL92 language enhancements have been added to Visual FoxPro 9 and are available to the SELECT command in the current VFP-9 Windows version 2.0 of *The Himalayan Database*.

The restrictions on the use of sub-queries mentioned earlier are removed:

(1) multiple sub-queries can be included in the WHERE clause of the primary query (the previous restriction was two sub-queries);
(2) sub-queries may be nested within other sub-queries;
(3) aggregate functions (such as COUNT, SUM, AVG, etc.) can be used in sub-queries.

The removal of last restriction on the use of aggregate functions with sub-queries is particularly fortunate as it greatly enhances the usefulness of correlated sub-queries (queries that refer to a table in the primary query).

For example, in standard SQL92, to search for all ethnic Sherpas that summited Everest fifteen times or more, the following SELECT statement could be used:

```
SELECT Trim(m1.fname)+" "+m1.lname AS name, m1.residence,
        m1.yob, m1.msmtdate1
    FROM members m1
    WHERE 15 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.residence=m2.residence And m1.yob=m2.yob And
                m1.sherpa=m2.sherpa And m1.msuccess=m2.msuccess And
                m1.peakid=m2.peakid And m2.peakid="EVER" And
                m2.sherpa And m2.msuccess)
    ORDER BY m1.lname, m1.fname, m1.msmtdate1
```

In the above example, the phrases “m1.residence=m2.residence” and “m1.yob=m2.yob” are required in the sub-query WHERE clause since individual Sherpas in the database are identified by their home (birth) village and year of birth as well as their name.

The output produced is a rather long list of Sherpas in alphabetic order with one row for each summit (along with the summit date).

Name	Residence	Yob	Msmtdate1
Apa (Appa) Sherpa	Thami Og, Khumbu	1960	5/10/1990
…	…	…	…
Apa (Appa) Sherpa	Thami Og, Khumbu	1960	1/05/2011
Chhuwang Nima Sherpa	Tesho, Khumbu	1967	13/05/1994
…	…	…	…
Chhuwang Nima Sherpa	Tesho, Khumbu	1967	05/05/2010
Chuldim Dorje (Ang Dorje) Sherpa	Pangboche, Khumbu	1964	12/05/1992
…	…	…	…
Chuldim Dorje (Ang Dorje) Sherpa	Pangboche, Khumbu	1964	19/05/2016
Dorje (Dorje Lambu, Big Dorje) Sherpa	Thamo, Khumbu	1965	15/05/1992
…	…	…	…
Dorje (Dorje Lambu, Big Dorje) Sherpa	Thamo, Khumbu	1965	15/05/1992
Kami Rita (Topke) Sherpa	Thami, Khumbu	1970	13/05/1994
…	…	…	…

But if we want a more compact list that gives only the names of the Sherpas along with a count of their summits, we can modify the above example by placing the "COUNT(*) AS count" expression in the output field list in place of the summit date:

```
SELECT Trim(m1.fname)+" "+m1.lname AS name, m1.residence,
      m1.yob, COUNT(*) AS count
   FROM members m1
   WHERE 15 <=
      (SELECT COUNT(*) FROM members m2
         WHERE m1.fname=m2.fname And m1.lname=m2.lname And
            m1.residence=m2.residence And m1.yob=m2.yob And
            m1.sherpa=m2.sherpa And m1.msuccess=m2.msuccess And
            m1.peakid=m2.peakid And m2.peakid="EVER" And
            m2.sherpa And m2.msuccess)
   GROUP BY name, m1.residence, m1.yob
   ORDER BY count DESC, name
```

The output produced list each Sherpa in alphabetic order and each row gives the number of summits for that Sherpa:

Name	Residence	Yob	Count
Apa (Appa) Sherpa	Thami Og, Khumbu	1960	21
Chuldim Dorje (Ang Dorje) Sherpa	Pangboche, Khumbu	1964	18
Kami Rita (Topke) Sherpa	Thami, Khumbu	1970	18
Mingma Tshering/Tsiri Sherpa	Beding, Dolakha	1970	18
Chuwang Nima Sherpa	Tesho, Khumbu	1967	17
Nima Gombu (Gombu) Sherpa	Beding, Dolakha	1969	17
Lhakpa Rita Sherpa	Thami, Khumbu	1966	16
Pasang Dawa (Pa Dawa, Pando) Sherpa	Pangboche, Khumbu	1977	16
Tshering Dorje Sherpa	Kharikhola, Solukhumbu	1970	16
Dorje (Dorje Lambu, Big Dorje) Sherpa	Thamo, Khumbu	1965	15
Lhakpa Gelu Sherpa	Kharikhola, Solukhumbu	1967	15
Mingma Chhiri/Chhiring Sherpa	Thami, Khumbu	1968	15
Ngima Nuru (Nima Nuru) Sherpa	Tesho, Khumbu	1981	15

The inclusion of COUNT(*) in the output field list requires the GROUP BY clause so that the count values can be calculated for each Sherpa. In this case, the GROUP BY clause must specify all of the other fields in the output list. The ORDER BY clause sorts the final result in descending order of summit counts.

Normally the ORDER BY clause can sort on table fields that are not specified in the output field list as illustrated in the first example that sorts on " m1.lname, m1.fname" instead of "name". But when a GROUP BY clause is specified as in the second example, only field names used in the output list may be used.

A similar example for searching for all non-Sherpas that summited Everest four times or more has some important differences due to the way that members are designated in the Himalayan Database. For Nepali Sherpas, Tamangs, Gurungs and Tibetans (those that are hired for expeditions) and Chinese Tibetans, residence field is considered a part of the member's identification and refers to that member's home (birth) village. For other members (usually foreign), the residence field gives the current residence at the time of the expedition (and thus may change from year to year) and is not a part of that member's identification. Thus our command is altered somewhat:

```
SELECT Trim(m1.fname)+" "+m1.lname AS name,
        m1.citizen, m1.yob, COUNT(*) AS count
    FROM members m1
    WHERE 4 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.yob=m2.yob And m1.sherpa=m2.sherpa And
                m1.msuccess=m2.msuccess And m1.peakid=m2.peakid And
                m2.peakid="EVER" And m2.msuccess And Not m2.sherpa)
    GROUP BY name, m1.citizen, m1.yob
    ORDER BY count DESC, name
```

The DESC keyword in the ORDER BY clause specifies descending order by count (the default is ascending order).

The output produced is given below. There is one issue to note with the output presented that regards the last two climbers, Anatoli Boukreev and Evgeni Vinogradski. Both climbers summited Everest four times, but each one climbed under two different citizenships as a result of the breakup of the USSR in 1992. Hence their totals should be combined. The query produced the correct total of four summits for each (since the "m1.citizen=m2.citizen" condition was not included in the sub-query), but the results were listed across two lines since "m1.citizen" was included in the output list. If "m1.citizen=m2.citizen" were included in the sub-query, they would have been omitted from the result since they would have been treated as four individuals, not two.

These types of issues can often occur with SQL query output since most databases do not contain absolutely perfect data.

Name	Citizen	Yob	Count
David Allen (Dave) Hahn	USA	1961	15
Kenton Edward Cool	UK	1973	11
Tashi Phuntsok (Tashi Phinzo)	China	1983	11
Guillermo (Willie) Benegas	USA	1968	10
Vernon Edward (Vern) Tejas	USA	1953	10
Chayang Jangbu (Chhyang Jyalbu) Bhote	Nepal	1979	9
Dean Douglas Staples	New Zealand	1964	9
Gheorghe Dijmarescu	USA	1961	9
Mark Wynton Woodward	New Zealand	1963	9
Tashi Tsering (Small)	China	1982	9
David William Hamilton	UK	1961	8
Michael John (Mike) Roberts	New Zealand	1961	8
Ngawang Norbu (Awang Luobo)	China	1980	8
Noel Richmond Hanna	UK	1967	8
Samduk Dorje (Sanduk Dorje) Tamang	Nepal	1983	8
Alexander (Alex) Abramov	Russia	1964	7
Charles Scott Woolums	USA	1957	7
…	…	…	…
Simone Moro	Italy	1967	4
Tashi Tsering (Big)	China	1979	4
Timothy John (Tim) Mosedale	UK	1965	4
Tsering Dorje (Cering Dorje)	China	1982	4
Victor Bobok	Russia	1961	4
Wallace Wayne (Wally) Berg	USA	1955	4
Yuri Contreras Cedi	Mexico	1963	4
Anatoli Boukreev	Kazakhstan	1958	3
Evgeni Vinogradski	Russia	1946	3
Anatoli Boukreev	USSR	1958	1
Evgeni Vinogradski	USSR	1946	1

The following example used nested sub-queries to list all Americans that have summited all three of the popular commercial peaks, Everest, Cho Oyu and Ama Dablam:

```
SELECT DISTINCT Trim(m1.fname)+" "+m1.lname AS name,
      m1.citizen, m1.yob
   FROM members m1
   WHERE m1.msuccess And m1.citizen="USA" And
      m1.peakid="EVER" And EXISTS
      (SELECT * FROM members m2
       WHERE m1.fname=m2.fname And m1.lname=m2.lname And
         m1.yob=m2.yob And m1.msuccess=m2.msuccess And
         m2.citizen="USA" And m2.peakid="CHOY" And
         m2.msuccess And EXISTS
      (SELECT * FROM members m3
       WHERE m2.fname=m3.fname And m2.lname=m3.lname And
         m2.yob=m3.yob And m2.msuccess=m3.msuccess And
         m3.citizen="USA" And m3.peakid="AMAD" And
         m3.msuccess))
   ORDER BY name
```

The output produced is

Name	Citizen	Yob
Andrew Atis (Andy) Lapkass	USA	1958
Brad Allen Johnson	USA	1955
Carlos Paltenghe Rockhold Buhler	USA	1954
Charles Scott Woolums	USA	1957
Christine Joyce Feld Boskoff	USA	1967
Christopher Bernard (Chris) Warner	USA	1964
Cleonice Pacheco (Cleo) Weidlich	USA/Brazil	1964
Daniel Lee (Dan) Mazur	USA	1960
David Charles (Dave) Morton	USA	1971
Douglas Lyle (Doug) Mantle	USA	1950
Eben Fleming Reckord	USA	1983
Ellen Elizabeth Miller	USA	1959
Emily Anne Harrington	USA	1986
Eric Lane Dalzell	USA	1983
Gary Scott Pfisterer	USA	1952
Guillermo (Willie) Benegas	USA	1968
Joby David Ogwyn	USA	1974
Justin Reese Merle	USA	1978
Kurt Alan Wedberg	USA	1966
Patrick J. Kenny	USA	1964
Peter Novak Anderson	USA	1979
Peter George (Pete) Athans	USA	1957
Robert Vincent (Bob) Jen	USA	1953
Stuart Gregory Smith	USA	1959
Tapley M. (Tap) Richards	USA	1974
Wallace Wayne (Wally) Berg	USA	1955

A similar example produces a list of all climbers that have summited all of the eight 8000m peaks in Nepal:

```
SELECT DISTINCT Trim(m1.fname)+" "+m1.lname AS name,
        m1.citizen, m1.yob
    FROM members m1
    WHERE m1.peakid="KANG" And m1.msuccess And 0 <
    (SELECT COUNT(*) FROM members m2
        WHERE m1.fname=m2.fname And m1.lname=m2.lname And
            m1.msuccess=m2.msuccess And m2.peakid="MAKA" And
            m2.msuccess And 0 <
    (SELECT COUNT(*) FROM members m3
        WHERE m2.fname=m3.fname And m2.lname=m3.lname And
            m2.msuccess=m3.msuccess And m3.peakid="EVER" And
            m3.msuccess And 0 <
    (SELECT COUNT(*) FROM members m4
        WHERE m3.fname=m4.fname And m3.lname=m4.lname And
        m3.msuccess=m4.msuccess And m4.peakid="LHOT" And
        m4.msuccess And 0 <
    (SELECT COUNT(*) FROM members m5
        WHERE m4.fname=m5.fname And m4.lname=m5.lname And
            m4.msuccess=m5.msuccess And m5.peakid="CHOY" And
            m5.msuccess And 0 <
    (SELECT COUNT(*) FROM members m6
        WHERE m5.fname=m6.fname And m5.lname=m6.lname And
            m5.msuccess=m6.msuccess And m6.peakid="MANA" And
            m6.msuccess And 0 <
    (SELECT COUNT(*) FROM members m7
        WHERE m6.fname=m7.fname And m6.lname=m7.lname And
            m6.msuccess=m7.msuccess And m7.peakid="ANN1" And
            m7.msuccess And 0 <
    (SELECT COUNT(*) FROM members m8
        WHERE m7.fname=m8.fname And m7.lname=m8.lname And
            m7.msuccess=m8.msuccess And m8.peakid="DHA1" And
            m8.msuccess)))))))
ORDER BY name
```

The output produced is

Name	Citizen	Yob
Abele Blanc	Italy	1954
Alberto Inurrategi Iriarte	Spain	1968
Andrew James Lock	Australia	1961
Carlos Miguel Carsolio Larrea	Mexico	1962
Chang-Ho Kim	S Korea	1969
Chhang Dawa Sherpa	Nepal	1982
Chun-Feng Yang	China	1968
Denis V. Urubko	Kazakhstan	1973
Denis V. Urubko	Russia/Kazakhstan	1973
Edmund Karl (Ed) Viesturs	USA	1959
Edurne Pasaban Lizarribar	Spain	1973
Eero Viekka Juhani Gustafsson	Finland	1968
Erhard Loretan	Switzerland	1959
…	…	…
Radek Jaros	Czech Republic	1964
Ralf Dujmovits	Germany	1961
Reinhold Messner	Italy	1944
Ren Na	China	1966
Samuli (Mika) Mansikka	Finland	1978
Serap Jangbu Sherpa	Nepal	1969
Sergio Martini	Italy	1949
Silvio Mondinelli	Italy	1958
Sung-Ho Seo	S Korea	1979
Tshering Dorje (Cerin Duoji)	China	1960
Vassily T. Pivtsov	Russia	1975
Vladislav Terzyul	Ukraine	1953
Wang-Yong Han	S Korea	1966
Young-Seok Park	S Korea	1963

Three comments should be made about this example. First, in order to abbreviate the length of SELECT command, the comparisons “m1.yob=m2.yob, etc.” are excluded which assumes that there are no conflicts in the database with climbers of the same name that summited 8000m peaks.

Second, the phrase “0 < (SELECT COUNT(*) FROM …” is used in place of “EXISTS (SELECT * FROM …”. The two phrases are logically equivalent, but due to limitations of SQL processing in VFP-9, only seven levels of the EXISTS form are allowed, whereas eight levels of the first form are permissible. When possible, the EXISTS form is preferred since it executes the query more efficiently.

Third, Denis V. Urubko appears twice since he changed his citizenship from Kazakhstan to Russian. This duplication could be eliminated if we had included the citizenship the comparisons “m1.citizen=m2.citizen, etc.”, but would have made the SELECT statements much more cumbersome.

An earlier VFP-6 example using a sub-query to search for climbers that have summited Everest from both sides

```
SELECT DISTINCT Trim(m1.fname)+" "+m1.lname AS name,
        m1.citizen, m1.yob
    FROM exped x, members m1
    WHERE x.expid=m1.expid And m1.peakid="EVER" And
        x.host=1 And m1.msuccess And EXISTS
        (SELECT * FROM exped x, members m2
            WHERE x.expid=m2.expid And m1.fname=m2.fname And
                m1.lname=m2.lname And m1.yob=m2.yob And
                m2.peakid="EVER" And x.host=2 And m2.msuccess)
    ORDER BY name
```

could be rewritten in VFP-9 using the JOIN-ON clause as

```
SELECT DISTINCT Trim(m1.fname)+" "+m1.lname AS name,
        m1.citizen, m1.yob
    FROM exped x JOIN members m1 ON x.expid=m1.expid
    WHERE m1.peakid="EVER" And x.host=1 And
        m1.msuccess And EXISTS
        (SELECT *
            FROM exped x JOIN members m2 ON x.expid=m2.expid
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.yob=m2.yob And m2.peakid="EVER" And
                x.host=2 And m2.msuccess)
    ORDER BY name
```

Additional Examples for Visual Foxpro 6 and 9

The following example selects members who have summited Everest and Lhotse in the same season. The output is ordered by age. The residence check is needed to distinguish Sherpas with the same name and age, but from different villages.

```
SELECT Trim(m1.fname)+" "+m1.lname AS name,
        m1.citizen, m1.residence, m1.calcage AS Age,
        m1.msmtdate1 AS Everest,m2.msmtdate1 AS Lhotse
    FROM members m1, members m2
    WHERE m1.myear=m2.myear And m1.mseason=m2.mseason
        And m1.peakid="EVER" And m2.peakid="LHOT"
        And m1.msuccess And m2.msuccess
        And m1.fname=m2.fname And m1.lname=m2.lname
        And m1.yob=m2.yob And m1.residence=m2.residence
    ORDER BY m1.calcage
```

The TOP clause can be used to extract the first *n* rows or *n* percentage of rows from the query result

```
SELECT TOP n [PERCENT] field-list FROM table-list WHERE condition
    ORDER BY order-list
```

The use of the TOP clause requires the inclusion of the ORDER BY clause in the SELECT statement. Thus to get the ten youngest to summit both Everest and Lhotse in the same season, specify

```
SELECT TOP 10 Trim(m1.fname)+" "+m1.lname AS name,
        m1.citizen,m1.residence, m1.calcage AS Age,
        m1.msmtdate1 AS Everest, m2.msmtdate1 AS Lhotse
    FROM members m1, members m2
    WHERE m1.myear=m2.myear And m1.mseason=m2.mseason
        And m1.peakid="EVER" And m2.peakid="LHOT"
        And m1.msuccess And m2.msuccess
        And m1.fname=m2.fname And m1.lname=m2.lname
        And m1.yob=m2.yob And m1.residence=m2.residence
    ORDER BY m1.calcage
```

Name	Citizen	Residence	Age	Everest	Lhotse
Mingma Tenzi Sherpa	Nepal	Yaphu-9, Makalu-Barun	22	16/05/2007	04/05/007
Phura Chhetan Sherpa	Nepal	Phortse, Khumbu	22	23/05/2013	17/05/2013
Edwin Spottswood Bailey	USA	Boulder, Colorado	22	18/05/2013	19/05/2013
Dawa Steven Sherpa	Nepal	Kathmandu	24	26/05/2008	21/05/2008
Norbu (Nuru) Sherpa	Nepal	Beding, Dolakha	25	09/10/1993	04/10/1993
Lhakpa Wangchu Sherpa	Nepal	Pangboche, Khumbu	25	19/05/2012	26/05/2012
Gyalzen Dorje Sherpa	Nepal	Phortse, Khumbu	25	10/05/2013	17/05/2013
Pasang Rinji Sherpa	Nepal	Kharikhola, Solukhumbu	26	22/05/2003	13/05/2003
Nima Gyalzen Sherpa	Nepal	Beding, Dolakha	26	18/05/2012	25/05/2012
Lhakpa Wangchu Sherpa	Nepal	Pangboche, Khumbu	26	13/05/2013	23/05/2013

To select climbers who summited Everest and Lhotse in the same season without oxygen, we would just add the phrase “And m1.mo2none And m2.mo2none” to the WHERE clause.

To select climbers who summited Lhotse the day after summiting Everest (an example of using calculations in the WHERE clause):

```
SELECT Trim(m1.fname)+" "+Trim(m1.lname) AS name, m1.citizen,
        m1.residence, m1.calcage AS age,
        m1.msmtdate1 AS Everest, m2.msmtdate1 AS Lhotse
    FROM members m1, members m2
    WHERE m1.myear=m2.myear And m1.mseason=m2.mseason
        And m1.peakid="EVER" And m2.peakid="LHOT"
        And m1.msuccess And m2.msuccess
        And m1.fname=m2.fname And m1.lname=m2.lname
        And m1.yob=m2.yob And m1.residence=m2.residence
        And m2.msmtdate1-m1.msmtdate1<=1
        And m2.msmtdate1>m1.msmtdate1
    ORDER BY m1.msmtdate1
```

Name	Citizen	Residence	Age	Everest	Lhotse
James Michael Horst	USA	Seattle, Washington	32	14/05/2011	15/05/2011
Thomas Halliday	USA	Chicago, Illinois	49	19/05/2011	20/05/2011
Garrett Christian Madison	USA	Bainbridge Island, Was...	32	19/05/2011	20/05/2011
Kristoffer Jon Erickson	USA	Livingston, Montana	38	25/05/2012	26/05/2012
Hilaree Janet O'Neill	USA	Telluride, Colorado	39	25/05/2012	26/05/2012
Michael Joseph Moniz	USA	Boulder, Colorado	50	26/05/2012	27/05/2012
Edwin Spottswood Bailey	Nepal	Boulder, Colorado	22	18/05/2013	19/05/2013
…	…	…	…	…	…

To select all 7000m peaks that were summited by S Koreans:

```
SELECT DISTINCT m.peakid, p.pkname, m.citizen
    FROM members m, peaks p
    WHERE m.peakid=p.peakid And
        Between(p.heightm,7000,7999) And
        m.citizen="S Korea"
    ORDER BY m.peakid
```

Peakid	Pkname	Citizen
ANN2	Annapurna II	S Korea
ANN3	Annapurna III	S Korea
ANN4	Annapurna IV	S Korea
ANNS	Annapurna South	S Korea
APIM	Api Main	S Korea
BARU	Baruntse	S Korea
CHAM	Chamlang	S Korea
CHRE	Churen Himal East	S Korea
CHRW	Churen Himal West	S Korea
CHUR	Churen Himal Central	S Korea
DHA6	Dhaulagiri VI	S Korea
…		…

To select all Australians that summited **one or more** Nepali main 8000ers:

```
SELECT Trim(m.lname)+",  "+Trim(m.fname) AS name,
        m.citizen, m.calcage AS age, m.sex, m.peakid AS peak,
        m.msmtdate1 AS smt_dt
    FROM members m
    WHERE Inlist(m.peakid,"KANG","MAKA","LHOT","EVER","CHOY",
            "MANA","ANN1","DHA1")
        And m.msuccess And Upper(m.citizen)="AUSTRALIA"
    ORDER BY name, m.msmtdate1
```

The output produced will contain one line for each 8000er summited by each Australian. To limit the selection to all Australians that summited **multiple** Nepali main 8000ers:

```
SELECT DISTINCT Trim(m1.lname)+", "+Trim(m1.fname) AS name,
        m1.citizen, m1.calcage AS age,
        m1.sex, m1.peakid AS peak,
        m1.msmtdate1 AS smt_dt
    FROM members m1, members m2
    WHERE Inlist(m1.peakid,"KANG","MAKA","LHOT","EVER","CHOY",
            "MANA","ANN1","DHA1")
        And Inlist(m2.peakid,"KANG","MAKA","LHOT","EVER","CHOY",
            "MANA","ANN1","DHA1")
        And m1.peakid<>m2.peakid
        And m1.msuccess And m2.msuccess
        And m1.fname=m2.fname And m1.lname=m2.lname
        And m1.yob=m2.yob
        And Upper(m1.citizen)="AUSTRALIA"
    ORDER BY name, m1.msmtdate1
```

The DISTINCT keyword is required to prevent redundant entries in the output.

Name	Citizen	Age	Sex	Peak	Smt_dt
Baldry, Anthony Donald	Australia	40	M	CHOY	27/09/2003
Baldry, Anthony Donald	Australia	41	M	EVER	27/05/2004
Baldry, Anthony Donald	Australia	49	M	MANA	11/05/2012
Baldry, Anthony Donald	Australia	50	M	LHOT	22/05/2013
Bart, Cheryl Sarah	Australia	48	F	CHOY	02/10/2007
Bart, Cheryl Sarah	Australia	49	F	EVER	24/05/2008
Bart, Nicole Karina (Nikki)	Australia	22	F	CHOY	02/10/2007
Bart, Nicole Karina (Nikki)	Australia	23	F	EVER	24/05/2008
Buck, Piers McAuley	Australia	29	M	CHOY	27/09/2003
Buck, Piers McAuley	Australia	30	M	CHOY	05/06/2005
…	…	…	…	…	…

Care must be used when using the ORDER BY clause when the DISTINCT keyword is present. When specifying compound items in the ORDER BY clause, we recommend using the AS-version of the item. In Visual FoxPro 9, the DISTINCT keyword may cause errors if the compound items are used directly.

In the first example without the DISTINCT keyword, the ORDER BY clause could be given in either Visual FoxPro 6 or 9 as any of

```
ORDER BY name, m.msmtdate1
ORDER BY name, smt_dt
ORDER BY m.lname, m.fname, m.msmtdate1
ORDER BY m.lname, m.fname, smt_dt
```

But with the DISTINCT keyword, in Visual FoxPro 9 only the following are allowed

```
ORDER BY name, m.msmtdate1
ORDER BY name, smt_dt
```

This restriction is likely due to a SQL implementation error in Visual FoxPro 9.

To select women that summited multiple Nepali main 8000ers in the same season:

```
SELECT DISTINCT Trim(m1.lname)+", "+Trim(m1.fname) AS name,
        m1.citizen, m1.calcage AS Age,
        m1.peakid AS peak1, m1.msmtdate1 AS peak1_dt,
        m2.peakid AS peak2, m2.msmtdate1 AS peak2_dt
    FROM members m1, members m2
    WHERE m1.myear=m2.myear And m1.mseason=m2.mseason
        And Inlist(m1.peakid,"KANG","MAKA","LHOT","EVER","CHOY",
            "MANA","ANN1","DHA1")
        And Inlist(m2.peakid,"KANG","MAKA","LHOT","EVER","CHOY",
            "MANA","ANN1","DHA1")
        And m1.peakid<>m2.peakid
        And m1.msuccess And m2.msuccess
        And m1.fname=m2.fname And m1.lname=m2.lname
        And m1.yob=m2.yob And m1.sex="F"
        And m1.residence=m2.residence
        And m1.msmtdate1>m2.msmtdate1
    ORDER BY m1.msmtdate1, name
```

Name	Citizen	Age	Peak1	Peak1_dt	Peak2	Peak2_dt
Rutkiewicz, Wanda	Poland	48	ANN1	22/10/1991	CHOY	26/09/1991
Mauduit, Chantal	France	32	MANA	24/05/1996	LHOT	10/05/1996
Oh, Eun-Sun	S Korea	42	LHOT	26/05/2008	MAKA	13/05/2008
Go, Mi-Sun	S Korea	41	KANG	18/05/2009	MAKA	01/05/2009
Oh, Eun-Sun	S Korea	42	DHA1	21/05/2009	KANG	06/05/2009
Denis, Sophie	France	32	LHOT	19/05/2011	CHOY	05/05/2011
Kazemi, Parvaneh	Iran	41	LHOT	25/05/2012	EVER	18/05/2012
O'Neill, Hilaree Janet	USA	39	LHOT	26/05/2012	EVER	25/05/2012
Weidlich, Cleonice Pacheco…	USA/Brazil	48	DHA1	26/05/2012	ANN1	20/04/2012
Gayen, Chhanda	India	33	LHOT	20/05/2013	EVER	18/05/2013
Luo, Jing	China	40	EVER	15/05/2016	ANN1	01/05/2016

To select members that summited Everest twice within 7 days (another example of using a calculation in the WHERE clause):

```
SELECT Trim(m.lname)+", "+Trim(m.fname) AS name,
        m.citizen, m.residence, m.calcage AS age,
        m.msmtdate1 AS smt_dt1,
        m.msmtdate2 AS smt_dt2
    FROM members m
    WHERE m.peakid="EVER"
        And m.msuccess
        And m.msmtdate2-m.msmtdate1 <= 7
        And Not Empty(m.msmtdate2)
    ORDER BY smt_dt1
```

Name	Citizen	Residence	Age	Smt_dt1	Smt_dt2
Rhoads, Jeffery E. (Jeff)	USA	Pocatello, Idaho	43	20/05/1998	27/05/1998
Sherpa, Tashi Tshering	Nepal	Pangboche, Khumbu	28	20/05/1998	27/05/1998
Sherpa, Pasang Dawa…	Nepal	Pangboche, Khumbu	29	18/05/2006	25/05/2006
Sherpa, Lhakpa Thundu…	Nepal	Pangboche, Khumbu	35	18/05/2006	25/05/2006
Sherpa, Dawa Nuru	Nepal	Phortse, Khumbu	27	20/05/2006	23/05/2006
Sherpa, Pemba Dorje	Nepal	Beding, Dolakha	30	08/05/2007	15/05/2007
Cool, Kenton Edward	UK	Chamonix, Haute…	33	17/05/2007	24/05/2007
Casserley, Robert Hargr…	UK	Bath, Avon, England	31	17/05/2007	24/05/2007
Sherpa, Pasang Dawa…	Nepal	Pangboche, Khumbu	30	17/05/2007	24/05/2007
Sherpa, Pasang Rita…	Nepal	Yilajung, Khumbu	36	07/05/2011	13/05/2011
…	…		…	…	…

To get members that summited Everest twice within the same season, increase the count from 7 to 60 or more.

Two UNION examples (**for Visual FoxPro 9 only**):

Example 1:

To select non-Sherpas that summited Everest 10 or more times and non-Sherpas that summited 16 or more times:

```
SELECT Trim(m1.fname)+" "+m1.lname AS name, " " AS residence,
        m1.citizen, m1.yob, COUNT(*) AS count
    FROM members m1
    WHERE 10 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.citizen=m2.citizen And m1.yob=m2.yob And
                m1.msuccess=m2.msuccess And
                m1.peakid=m2.peakid And m2.peakid="EVER" And
                Not m2.sherpa And m2.msuccess)
    GROUP BY name, m1.citizen, m1.yob

UNION

SELECT Trim(m1.fname)+" "+m1.lname AS name, m1.residence AS
        residence, m1.citizen, m1.yob, COUNT(*) AS count
    FROM members m1
    WHERE 16 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.residence=m2.residence And m1.yob=m2.yob And
                m1.msuccess=m2.msuccess And
                m1.peakid=m2.peakid And m2.peakid="EVER" And
                m2.sherpa And m2.msuccess)
    GROUP BY name, residence, m1.citizen, m1.yob

ORDER BY count DESC, name
```

Note the different uses of "citizen" and "residence" in each of the two parts of the UNION clause due to the fact the residence is an integral part of a Sherpa's identity in the Himalayan Database. The use of residence is necessary to prevent the combination of counts from Sherpas with the same name and YOB (unlikely, but still possible).

Example 2:

Select all climbers that have summited Everest and Cho Oyu six or more times each:

```
SELECT Trim(m1.fname)+" "+m1.lname AS name, m1.citizen,
        m1.yob, COUNT(*) AS evercnt
    FROM members m1
    WHERE 6 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.citizen=m2.citizen And m1.yob=m2.yob And
                m1.msuccess=m2.msuccess And
                m1.peakid=m2.peakid And m2.peakid="EVER" And
                m2.msuccess And 6 <=
        (SELECT COUNT(*) FROM members m3
            WHERE m2.fname=m3.fname And m2.lname=m3.lname And
                m2.citizen=m3.citizen And m2.yob=m3.yob And
                m2.msuccess=m3.msuccess And
                m3.peakid="CHOY" And m3.msuccess))
    GROUP BY name, m1.citizen, m1.yob
    ORDER BY evercnt DESC, name
```

The output produced is

Name	Citizen	Yob	Evercnt
Kami Rita (Topke) Sherpa	Nepal	1970	18
Lhakpa Rita Sherpa	Nepal	1966	16
Tshering Dorje Sherpa	Nepal	1970	16
Dawa Nuru (Danuru) Sherpa	Nepal	1978	14
Kami Tshering (Ang Chhiring) Sherpa	Nepal	1962	12
Chhiring Dorje Sherpa	Nepal	1974	11
Norbu/Nurbu (Nuru) Sherpa	Nepal	1968	10
Jangbu Sherpa	Nepal	1967	9
Karsang Namgyal/Namgel Sherpa	Nepal	1971	9
Mingma Tenzing Sherpa	Nepal	1986	7
Tamtin (Thomting, Tamding) Sherpa	Nepal	1974	7
Lobsang Temba (Lupsang Temba) Sherpa	Nepal	1968	6
Michael Aaron Hamill	USA	1977	6

The limitation of the above example is that it only gives the summit count for Everest and not for Cho Oyu. In order to get both summit counts, the following can be used.

```
SELECT Trim(m1.fname)+" "+m1.lname AS name, m1.citizen,
        m1.yob, COUNT(*) AS evercnt, 0 AS choycnt
    FROM members m1
    WHERE 6 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.citizen=m2.citizen And m1.yob=m2.yob And
                m1.msuccess=m2.msuccess And
                m1.peakid=m2.peakid And m2.peakid="EVER" And
                m2.msuccess And 6 <=
        (SELECT COUNT(*) FROM members m3
            WHERE m2.fname=m3.fname And m2.lname=m3.lname And
                m2.citizen=m3.citizen And m2.yob=m3.yob And
                m2.msuccess=m3.msuccess And
                m3.peakid="CHOY" And m3.msuccess))
    GROUP BY name, m1.citizen, m1.yob

UNION

SELECT Trim(m1.fname)+" "+m1.lname AS name, m1.citizen,
        m1.yob, 0 AS evercnt, COUNT(*) AS choycnt
    FROM members m1
    WHERE 6 <=
        (SELECT COUNT(*) FROM members m2
            WHERE m1.fname=m2.fname And m1.lname=m2.lname And
                m1.citizen=m2.citizen And m1.yob=m2.yob And
                m1.msuccess=m2.msuccess And
                m1.peakid=m2.peakid And m2.peakid="CHOY" And
                m2.msuccess And 6 <=
        (SELECT COUNT(*) FROM members m3
            WHERE m2.fname=m3.fname And m2.lname=m3.lname And
                m2.citizen=m3.citizen And m2.yob=m3.yob And
                m2.msuccess=m3.msuccess And
                m3.peakid="EVER" And m3.msuccess))
    GROUP BY name, m1.citizen, m1.yob

INTO TABLE everchoy
ORDER BY name, yob
```

The output produced into the table “Everchoy” is

Name	Citizen	Yob	Evercnt	Choycnt
Chhiring Dorje Sherpa	Nepal	1974	0	7
Chhiring Dorje Sherpa	Nepal	1974	11	0
Dawa Nuru (Danuru) Sherpa	Nepal	1978	0	9
Dawa Nuru (Danuru) Sherpa	Nepal	1978	14	0
Jangbu Sherpa	Nepal	1967	0	6
Jangbu Sherpa	Nepal	1967	9	0
Kami Rita (Topke) Sherpa	Nepal	1970	0	8
Kami Rita (Topke) Sherpa	Nepal	1970	18	0
Kami Tshering (Ang Chhiring) Sherpa	Nepal	1962	0	7
Kami Tshering (Ang Chhiring) Sherpa	Nepal	1962	12	0
Karsang Namgyal/Namgel Sherpa	Nepal	1971	0	6
Karsang Namgyal/Namgel Sherpa	Nepal	1971	9	0
…	…	…	…	…

which gives two entries for each climber. The following subsequent query on the table “everchoy”

```
SELECT name, citizen, yob, Sum(evercnt) AS everest,
        Sum(choycnt) AS choy_oyu
    FROM everchoy
    GROUP BY name, citizen, yob
    ORDER BY everest DESC, name
```

merges the two entries for each climber and produces the final output

Name	Citizen	Yob	Everest	Choy_oyu
Kami Rita (Topke) Sherpa	Nepal	1970	18	8
Lhakpa Rita Sherpa	Nepal	1966	16	10
Tshering Dorje Sherpa	Nepal	1970	16	6
Dawa Nuru (Danuru) Sherpa	Nepal	1978	14	9
Kami Tshering (Ang Chhiring) Sherpa	Nepal	1962	12	7
Chhiring Dorje Sherpa	Nepal	1974	11	7
Norbu/Nurbu (Nuru) Sherpa	Nepal	1968	10	6
Jangbu Sherpa	Nepal	1967	9	6
Karsang Namgyal/Namgel Sherpa	Nepal	1971	9	6
Mingma Tenzing Sherpa	Nepal	1986	7	6
Tamtin (Thomting, Tamding) Sherpa	Nepal	1974	7	6
Lobsang Temba (Lupsang Temba) Sherpa	Nepal	1968	6	6
Michael Aaron Hamill	USA	1977	6	8

The Example 2 statements are not valid in Visual FoxPro 6 due to sub-query nesting restrictions.