

### LAND CONDITION TREND ANALYSIS II

# LCTA II Technical Reference Manual



## *Chapter 8* *Structured Query Language (SQL)*



## 8 Structured Query Language (SQL)

The LCTA Program Manager offers many analyses and utilities to summarize the LCTA data, however; these analyses only provide standardized summaries. In order to summarize the LCTA data beyond the LCTA Program Manager and maintain the database, Structured Query Language (SQL) is required. Although many databases allow for generating queries through the use of a point-and-click tool, this is often time consuming. Once the basic SQL statements are understood, summarizing and maintaining the data will become easier and faster.

### 8.1 Orientation

This section teaches the basic skills needed to create SQL commands. Syntax for both Centura SQLBase and Microsoft Access are presented. Refer to the Microsoft Access documentation for installation instructions. Section 6.5.3 covers installation instructions for SQLBase and older versions of Quest.

### 8.2 Creating SQL Statements in Quest

To open the SQL window select the **NEW** button to the left of the toolbar. Select the **SQL** button and an empty SQL window will open. It is here that the SQL commands are entered. To execute the SQL statements select <SQL> from the menu then select either <Execute SQL Statement> or <Execute All SQL>. To save SQL statements select <File> from the main menu, <Save>, <SQL>, and enter a name for the file. To open existing SQL statements, select the **OPEN** button to the left of the toolbar, then the **SQL** button, and select the desired file name.

### 8.3 Creating SQL Statements in Access

All tables, reports, queries, macros, and modules are available from the Database Window. To create a new query, or SQL statement, in Access select the Query tab from the Database Window and select the **New** button. Queries can be created using the Query By Example (QBE) tool or directly typing SQL statements. For this chapter all SQL statements are entered in the SQL window. After select the New button pick Design View from the list. When the table list appears select the No button, the design grid is presented. Select the **SQL** button from the top toolbar and an SQL window will now be displayed. To execute a statement select <Query> from the main menu and <Run>.

### 8.3.1 Syntax Diagrams

This section will use the following syntax diagrams for SQL statements.

- text between brackets, "[ ]", are optional SQL components
- normal text between braces, "{ }", contains a list of required elements, only one of the elements listed is required
- italics text represents a generic place holder, for example, *tablename* denotes a table name should be entered at that location
- bold text are key words

Each SQL statement covered in this chapter will list the general SQL syntax, an LCTA example, and the English meaning of the statement. Each example a translation will be designated as SQLBase specific, Access specific, or applies to both. Following is an example of an SQL syntax diagram.

#### Syntax Diagram

```
SELECT [distinct] {columnlist,*} FROM tablename  
[WHERE condition [AND condition]]  
[ORDER BY columnlist];
```

#### LCTA Example

```
SELECT PlotID, RecDate, LandUse FROM LandUse  
WHERE PlotID < 300  
ORDER BY PlotID;
```

#### English Translation

Select the plot number, recording date, and land use from the LandUse table where the plot number is less than 300 and sort the list by plotid.

The use of uppercase and mixed case words in the syntax diagram and LCTA example are only used to simplify the reading of these SQL statements. For SQLBase SQL statements, only text entered between single quotes are case sensitive. Refer to the function **@Upper()**, explained below, for more information on case sensitivity in SQL statements. Access is not case sensitive and does not require the use of the **UCASE()** Access function.

In the above example the syntax diagram shows the general statement syntax. Notice the keywords **SELECT**, **FROM**, **WHERE**, **AND**, **ORDER BY**. The word **distinct** is an optional word in this statement. *Columnlist* represents a place holder for a list of column names separated by commas. *tablename* is the place holder for the name of the table, and *condition* is the place holder for the where clause. Not only is the where clause

optional, but the where clause can contain multiple conditions by using the **AND** keyword.

As mentioned earlier, this section will cover only the basic SQL statements. In the Advanced SQL section the SQL syntax diagrams will be expanded to incorporate the added functionality of the statements.

### 8.3.2 SQLBase SQL Functions

SQLBase utilizes many built in functions to help retrieve the data that is needed. Listed here are two of the most commonly used and useful functions in basic SQL statements.

- **@YEARNO** (*columnname*)  
Retrieve only the year part of a date column
- **@UPPER** (*columnname*)  
Convert a column to uppercase. SQLBase is case sensitive when examining the values of text data contained in the database. This function is used to ensure a match in a where clause.

Examples of these functions are used in the following sections.

### 8.3.3 Access SQL Functions

Listed here are two of the most commonly used functions.

**YEAR**(*columnname*)

Retrieve only the year part of a date column

**UCASE**()

Convert a column to uppercase. This conversion is only temporary and does not save the converted data. Access is not case sensitive; this function is only used to return values in uppercase if desired.

### 8.3.4 The SELECT Statement

A select statement is used to retrieve information from one or more tables. When data is retrieved from more than one table a join is performed, this is covered in the Advanced SQL section.

### **Syntax Diagram**

```
SELECT [distinct] {columnlist,*} FROM tablename  
[WHERE condition [AND condition]]  
[ORDER BY columnlist];
```

### **LCTA Example 1 (Quest and Access)**

```
SELECT * FROM LandUse;
```

### **LCTA Example 2 (Quest and Access)**

```
SELECT distinct PlotID, RecDate, LandUse FROM LandUse  
WHERE PlotID < 300  
ORDER BY PlotID;
```

### **LCTA Example 3 (Quest)**

```
SELECT PlotID, @YEARNO(RecDate), LandUse FROM LandUse  
WHERE PlotID < 300 and @UPPER(LandUse) = 'TRACKED' and  
@YEARNO(RecDate) > 1990  
ORDER BY 1,2;
```

### **LCTA Example 3 (Access)**

```
SELECT PlotID, YEAR(RecDate), LandUse FROM LandUse  
WHERE PlotID < 300 and LandUse = "TRACKED" and YEAR(RecDate) > 1990  
ORDER BY PlotID, YEAR(RecDate);
```

### **English Translation 1**

Select all columns from the table LandUse.

### **English Translation 2**

Select only unique values of plot number, recording date, and land use from the LandUse table where the plot number is less than 300. Sort the list by PlotID.

### **English Translation 3**

Select the plot number, year portion of the recording date, and land use from the LandUse table,  
where the plot number is less than 300, the uppercase land use value is equal to "TRACKED", and the year portion of the recording date is greater than 1990. Sort the list by PlotID and the year portion of the recording date. The Access statement does not require the use of the UCASE() function.

Notice that in example 3, Quest syntax, the **ORDER BY** clause uses 1,2 not PlotID, **@YEARNO**(RecDate). This is a short hand method of listing the variables utilized by SQLBase. The numbers correspond to the list of columns following the **SELECT** key word. When using a function, such as **@YEARNO**(RecDate), in the column list this short hand method is required.

### 8.3.5 The UPDATE Statement

This statement is used to change values of existing data in the database.

#### **Syntax Diagram**

```
UPDATE tablename
SET columnname = expression
[WHERE condition];
```

#### **LCTA Example 1 (Quest and Access)**

```
UPDATE PlotSurv
SET PlotType = 'C';
```

#### **LCTA Example 2 (Quest)**

```
UPDATE PlotSurv
SET PlotType = 'S'
WHERE @UPPER(PlotType) = 'SPUSE';
```

#### **LCTA Example 2 (Access)**

```
UPDATE PlotSurv
SET PlotType = "S"
WHERE PlotType = "SPUSE";
```

#### **LCTA Example 3 (Quest)**

```
UPDATE LandUse
SET LandUse = 'TRACKED'
WHERE @UPPER(LandUse) = 'TRACK';
```

### **LCTA Example 3 (Access)**

```
UPDATE LandUse  
SET LandUse = "TRACKED"  
WHERE LandUse = "TRACK";
```

#### **English Translation 1**

Change all the values of plot type, in the table PlotSurv, to 'C'.

#### **English Translation 2**

Change the plot type of plots in the table PlotSurv to 'S' where the uppercase value of existing plot types is equal to 'SPUSE'. Access does not require the UCASE() function. Also notice that Access used double quotation marks.

#### **English Translation 3**

Change the land use value, in the table LandUse, to 'TRACKED' where the uppercase value of the existing land use is equal to 'TRACK'. Again, Access does not use the UCASE() function.

The update statement is very useful when editing data. Quest does not allow editing returned values in a query or SQL window. There are two alternatives to correct the data in the database, the update statement or manually by opening the table and scrolling through the data to find the incorrect values. Access does allow the editing of values returned in a query. The only exception to this is Access is when aggregation functions are used or in some join statements.

If a large number of changes are needed a Quest SQL script file should be created. A script file contains more than one SQL statement, each ending with a semicolon. Enter all the SQL statements in an SQL window and choose <SQL> <Execute All> from the main menu. Quest will execute each update statement that is properly written in the SQL window. Unfortunately, Access does not allow the use of scripts. Macros can be used for the same purpose, refer to the Access documentation for more information on macros.

### **8.3.6 The INSERT Statement**

The insert statement will insert data into a defined table in the database. Because it is usually easier to open the table in Quest and use the Window's copy and paste functions, this discussion is limited. For information on pasting data into a table consult the database documentation. The insert statement discussed here will demonstrate how to extract data from one table and insert it into another. This statement uses a subselect, which is covered further, in the Advanced SQL section.

For this statement to work, the number of columns and type of data extracted from one table must match that of the receiving table.

### **Syntax Diagram**

```
INSERT INTO tablename[(columnlist)]  
subselect;
```

### **LCTA Example 1 (Quest and Access)**

```
INSERT INTO MyTable  
select PlotID, PlotType from PlotSurv;
```

### **LCTA Example 2 (Quest and Access)**

```
INSERT INTO MyTable2  
select distinct PlotID from PlotSurv;
```

### **LCTA Example 3 (Quest)**

```
INSERT INTO MyTable3(PlotID, Yr)  
select distinct PlotID, @YEARNO(RecDate) from PlotSurv where  
@UPPER(PlotType) = 'C';
```

### **LCTA Example 3 (Access)**

```
INSERT INTO MyTable3(PlotID, Yr)  
select distinct PlotID, YEAR(RecDate) from PlotSurv where PlotType = "C";
```

### **English Translation 1**

Extract plot numbers and plot types from the table PlotSurv and insert the values into the table Mytable, which only contains the columns PlotID and PlotType.

### **English Translation 2**

Extract unique plot numbers from the table PlotSurv and insert the values into the table Mytable2, which only contains the column PlotID.

### **English Translation 3 (Quest)**

Extract unique values of PlotID and the year portion of the recording date from the PlotSurv table, where the uppercase value of plot type is equal to 'C'. Insert these values into the table Mytable3, which has two columns defined.

### **English Translation 3 (Access)**

Extract unique values of PlotID and the year portion of the recording date from the PlotSurv table, where the plot type is equal to "C". Insert these values into the table Mytable3, which has two columns defined.

Notice that in example 3 the optional column list was used after the table name receiving data. Because a function is used in the subselect, these columns must be identified by name in the column list.

### **8.3.7 The DELETE Statement**

The **DELETE** statement is used to delete data from a table. Be careful when using this command, it is easy to delete more data than intended.

#### **Syntax Diagram**

```
DELETE FROM tablename  
[WHERE condition];
```

#### **LCTA Example 1 (Quest and Access)**

```
DELETE FROM LandUse;
```

#### **LCTA Example 2 (Quest and Access)**

```
DELETE FROM LandUse  
WHERE PlotID = 3;
```

#### **LCTA Example 3 (Quest)**

```
DELETE FROM LandUse  
WHERE PlotID = 3 and @YEARNO(RecDate) = 1990;
```

#### **LCTA Example 3 (Access)**

```
DELETE FROM LandUse  
WHERE PlotID = 3 and YEAR(RecDate) = 1990;
```

#### **English Translation 1**

Delete all data in the table LandUse.

### **English Translation 2**

Delete data from the table LandUse where the PlotID is equal to 3.

### **English Translation 3**

Delete data from the table LandUse, where the PlotID is equal to 3 and the year portion of the recording data is equal to 1990.

## **8.3.8 The DROP Statement (Quest Only)**

The drop statement is used to remove a table from the database. Be careful when using this command, the table and all data is removed from the database. Drop is also used to drop views and indexes, consult the Quest User's Guide for more information. This section will only discuss using the drop statement to remove tables from the database.

### **Syntax Diagram**

**DROP** *tablename*;

### **LCTA Example 1**

**DROP** LandUse;

### **LCTA Example 2**

**DROP** mytable;

### **English Translation 1**

Remove the table LandUse from the database.

### **English Translation 2**

Remove the table mytable from the database.

## **8.3.9 The CREATE VIEW Statement (Quest Only)**

The create view statement is used to create a view on one or more tables in SQLBase. If a view is created from only one table the data in the view can be modified. Views are useful for showing only certain information from a table. Views are also useful for storing the joining of information from two tables. The difference between a saved query, or SQL statement, and a view is that a view can be treated as a table and used in other SQL statements. When a view is created the data is not stored in the database as another

table, only the SQL statement that creates the view is stored. This is useful because every time the view is opened the data is retrieved from the appropriate tables, ensuring that the view is always up-to-date.

Once a view is created, open the view by selecting the **OPEN** and **TABLE** buttons, the view is listed with the other tables. In Access, saved queries are functionally the same as a view in SQLBase.

### **Syntax Diagram**

```
CREATE VIEW view-name [(columnnames)]  
AS SELECT selectstatement;
```

View name is a name given to the view. The optional *columnnames* is used to give different names to the columns of the view, different from the column names specified in the select statement. The *selectstatement* is any valid select statement. Only basic select statements are shown here, but the Advanced SQL section will show how to write more advanced ones.

### **LCTA Example 1**

```
CREATE VIEW myview  
AS SELECT PlotID, PlotType FROM PlotSurv;
```

### **LCTA Example 2**

```
CREATE VIEW myview2(PlotNum, CoreOrSpUse)  
AS SELECT PlotID, PlotType FROM PlotSurv;
```

### **English Translation 1**

Create a view, named myview, containing plot number and plot type from the table PlotSurv.

### **English Translation 2**

Create a view, named myview2, containing plot number and plot type from the table PlotSurv. Name these columns PlotNum and CoreOrSpuse respectively.

## **8.3.10 The CHECK DATABASE Statement (Quest Only)**

The check database statement performs integrity checks on the entire database. This statement is useful for ensuring the database does not contain a problem that can cause errors or loss of data.

### **Syntax Diagram**

**CHECK DATABASE;**

### **LCTA Example 1**

**CHECK DATABASE:**

### **English Translation 1**

Check the integrity of the entire database.

This command may take a while to run. When Quest has completed the check and no errors are found, “Ready” will appear in the bottom left corner of the Quest window. If errors are found they are reported on the screen. Most of the error statements are fairly cryptic, for help with errors contact the LCTA Support Center.

## **8.4 Advanced SQL Statements**

In the previous section basic commonly used SQL statements were covered. In this section more advanced SQL statements are explained. All of the advanced SQL statements discussed here utilize the select statement and further the functionality of this command.

### **8.4.1 Selecting Data From More Than One Table (Joining)**

#### **Syntax Diagram**

```
SELECT [distinct] {columnlist,*} FROM tablenames  
[WHERE condition [AND condition]]  
[ORDER BY columnlist];
```

The only difference between this statement and the basic select statement, in the previous section, is the *columnlist ,tablenames* and the where conditions. Often when joining tables column names must be identified by the table they are found in. Using the syntax *tablename.columnname*, for example *plotsurv.plotid*, does this. This is only needed if the same column exists in multiple tables referenced in the SQL statement, however; it is good practice to use this syntax to avoid errors. In the FROM clause of the SELECT statement there is a list of tables separated by commas. The where condition must have a least one join statement. A join statement sets two columns of the same data, in two tables, to be equal. Tables cannot be joined if they do not have at least one column in common.

### **LCTA Example 1 (Quest and Access)**

```
SELECT PlotSurv.PlotID, PlotSurv.PlotType, LandUse.LandUse FROM PlotSurv,
LandUse
WHERE PlotSurv.PlotID=LandUse.PlotID
AND PlotSurv.RecDate = LandUse.RecDate;
```

### **LCTA Example 2 (Quest)**

```
SELECT PlotSurv.PlotID, PlotSurv.PlotType, LandUse.LandUse FROM PlotSurv,
LandUse
WHERE PlotSurv.PlotID=LandUse.PlotID
AND @UPPER(PlotSurv.PlotType) = 'C'
AND @YEARNO(PlotSurv.RecDate) = 1990
AND PlotSurv.RecDate = LandUse.RecDate
ORDER BY PlotSurv.PlotID;
```

### **LCTA Example 2 (Access)**

```
SELECT PlotSurv.PlotID, PlotSurv.PlotType, LandUse.LandUse FROM PlotSurv,
LandUse
WHERE PlotSurv.PlotID=LandUse.PlotID
AND PlotSurv.PlotType = "C"
AND YEAR(PlotSurv.RecDate) = 1990
AND PlotSurv.RecDate = LandUse.RecDate
ORDER BY PlotSurv.PlotID;
```

### **LCTA Example 3 (Quest)**

```
SELECT PCSDPlotSum.PlotID,(PCSDPlotSum.GCBare*100/PCSDPlotSum.GCObs)
FROM PlotSurv, PCSDPlotSum
WHERE PlotSurv.PlotID=PCSDPlotSum.PlotID
AND @UPPER(PlotSurv.PlotType) = 'C'
AND @YEARNO(PlotSurv.RecDate) = PCSDPlotSum.AnalYear;
```

### **LCTA Example 3 (Access)**

```
SELECT PCSDPlotSum.PlotID,(PCSDPlotSum.GCBare*100/PCSDPlotSum.GCObs)
FROM PlotSurv, PCSDPlotSum
WHERE PlotSurv.PlotID=PCSDPlotSum.PlotID
AND PlotSurv.PlotType = "C"
AND YEAR(PlotSurv.RecDate) = PCSDPlotSum.AnalYear;
```

### **English Translation 1**

Select plot number and plot type from the tables PlotSurv and LandUse where plot numbers from both are equal.

### **English Translation 2 (Quest)**

Select plot numbers and plot types from the tables PlotSurv and LandUse where plot numbers from both tables are equal. Also, the uppercase value of PlotType must be 'C' and the year portion of the recording date from the PlotSurv table is 1990, sort the information by plot number.

### **English Translation 2 (Access)**

Select plot numbers and plot types from the tables PlotSurv and LandUse where plot numbers from both tables are equal. Also, PlotType must be 'C' and the year portion of the recording date from the PlotSurv table is 1990, sort the information by plot number.

### **English Translation 3 (Quest)**

Select plot number and the average bare ground value from the table PCSDPlotSum, where the plot numbers from PlotSurv and PCSDPlotSum are equal. Also, the uppercase value of the plot type is equal to 'C' and the year part of the recording date, from the PlotSurv table, is equal to AnalYear from the PCSDPlotSum table.

### **English Translation 3 (Access)**

Select plot number and the average bare ground value from the table PCSDPlotSum, where the plot numbers from PlotSurv and PCSDPlotSum are equal. Also, the plot type is equal to 'C' and the year part of the recording date, from the PlotSurv table, is equal to AnalYear from the PCSDPlotSum table. The average bare ground value returned in this statement is expressed as a percentage. To return the decimal representation of this value, remove (\*100) from the statement.

Example 3, Quest syntax, uses a mathematical expression to calculate the average bare ground per plot from the data summary table PCSDPlotSum. The Plant Cover Surface Disturbance (PCSD) analysis, from the LCTA Program Manager, stores data in the table PCSDPlotSum for each plot. GCBare is the number of bare ground hits for a plot and GCObs is the number of total ground hits for that plot. Because both of these values are integers dividing them returns the value of integer division, zero. In order to force SQLBase to perform non-integer division, multiply the numerator by 100. The resulting value is the percent bare ground.

## 8.4.2 Grouping Data

Grouping data is useful when trying to combine values in a larger unit other than plots, for example year. Use the GROUP BY clause for aggregation functions such as, sum, average, and count.

### Syntax Diagram

```
SELECT [distinct] {columnlist,*} FROM tablename  
[WHERE condition [AND condition]]  
GROUP BY columnlist  
[ORDER BY columnlist];
```

### LCTA Example 1 (Quest)

```
SELECT @YEARNO(RecDate), PlotType, COUNT(PlotType) FROM PlotSurv  
GROUP BY 1,2;
```

### LCTA Example 1 (Access)

```
SELECT YEAR(RecDate), PlotType, COUNT(PlotType) FROM PlotSurv  
GROUP BY YEAR(RecDate), PlotType;
```

### LCTA Example 2 (Quest)

```
SELECT @YEARNO(RecDate),PlotType, COUNT(PlotType) FROM PlotSurv  
WHERE PlotID < 300  
GROUP BY 1;
```

### LCTA Example 2 (Access)

```
SELECT YEAR(RecDate),PlotType, COUNT(PlotType) FROM PlotSurv  
WHERE PlotID < 300  
GROUP BY YEAR(RecDate),PlotType;
```

### LCTA Example 3 (Quest)

```
SELECT PCSDPlotSum.AnalYear,  
AVG(PCSDPlotSum.GCBare*100/PCSDPlotSum.GCObs) As AverageBG FROM  
PlotSurv, PCSDPlotSum  
WHERE PlotSurv.PlotID=PCSDPlotSum.PlotID  
AND @UPPER(PlotSurv.Plotype) = 'C'  
AND PCSDPlotSum.GCObs <> 0  
GROUP BY pcsdplotsum.analyear;
```

### LCTA Example 3 (Access)

```
SELECT PCSDPlotSum.AnalYear,  
AVG(PCSDPlotSum.GCBare*100/PCSDPlotSum.GCObs) As AverageBG FROM  
PlotSurv, PCSDPlotSum  
WHERE PlotSurv.PlotID=PCSDPlotSum.PlotID  
AND PlotSurv.Plotype = "C"  
AND PCSDPlotSum.GCObs <> 0  
GROUP BY pcsdplotsum.analyear;
```

### English Translation 1 (Quest and Access)

For each plot type, count the number of occurrences for each year.

### English Translation 2 (Quest and Access)

For each plot type, count the number of occurrences for each year. Only examine data for plots less than 300.

### English Translation 3 (Quest and Access)

Select plot number and the average mean bare ground from the table PCSDPlotSum, where the plot numbers from the tables PlotSurv and PCSDPlotSum are equal. Only include data that have plot types of 'C' and GCObs greater than zero.

Notice in examples 1 and 2, Quest syntax, the **ORDER BY** clause uses 1 not **@YEARNO**(RecDate). This is a short hand method of listing the variables in SQLBase. The number corresponds to the list of columns following the **SELECT** key word. When using a function such as **@YEARNO**(RecDate) in the column list, the short hand method is required in SQLBase. This is not the case in Access.

Also notice that example 3 uses the average function (**AVG**). The where clause of this example contains the line “**AND PCSDPlotSum.GCObs <> 0**”, this ensures that there is no division by 0. Example 3 also takes advantage of the **AS** clause to rename the output column of the average bare ground. This is for display purposes only.

## 8.4.3 Using Collections (IN and NOT IN Statements) (Quest Only)

The IN predicate is used to compare one value to a collection of other values. The collection of values can be listed in the SQL statement or be the result of a subselect. A subselect is a SELECT statement nested within another. Although Access can use the same syntax, an alternative method of writing this SQL seems to run faster in Access for some types of collections. The next section describes this alternative method.

Notice that example 2 shows both Quest and Access syntax. This type of collection can be used in Access with no problems. The Access syntax for examples 2 and 3 are presented in the next section.

### **Syntax Diagram (Quest and Access)**

```
SELECT [distinct] {columnlist,*} FROM tablename  
[WHERE expression [NOT] IN {subselect, listofvalues} [AND condition]]  
[ORDER BY columnlist];
```

### **LCTA Example 1 (Quest)**

```
SELECT distinct VegID FROM AerCover  
WHERE @UPPER(VegID) NOT IN (SELECT @UPPER(VegID) FROM PlntList);
```

### **LCTA Example 2 (Quest)**

```
SELECT PlotID, LandUse FROM LandUse  
WHERE @UPPER(LandUse) IN ('TRACKED', 'WHEELED');
```

### **LCTA Example 2 (Access)**

```
SELECT PlotID, LandUse FROM LandUse  
WHERE LandUse IN ("TRACKED", "WHEELED");
```

### **LCTA Example 3 (Quest)**

```
SELECT GndCover.PlotID, GndCover.VegID FROM GndCover, PlotSurv  
WHERE PlotSurv.PlotID = GndCover.PlotID  
AND @UPPER(PlotSurv.InvType) = 'I'  
AND @UPPER(PlotSurv.PlotType) = 'C'  
AND @UPPER(GndCover.VegID) IN (SELECT @UPPER(VegID) FROM PlntList)  
ORDER BY GndCover.PlotID, GndCover.VegID;
```

### **English Translation 1**

Select unique values of vegetation codes from the AerCover table, where the uppercase of the vegetation code is in the set of uppercase vegetation codes taken from the table PlntList. The result set of this statement will contain only known vegetation codes.

### **English Translation 2 (Quest and Access)**

Select plot number and LandUse values from the LandUse table where the value LandUse is equal to 'TRACKED' or 'WHEELED'. Notice that Access does not use the UCASE() function.

### **English Translation 3**

Select plot number and vegetation code from the GndCover table, where the plot numbers in the tables GndCover and PlotSurv are equal. Include only plots with an inventory type of 'I' and a plot type of 'C'. In addition, the uppercase value of the vegetation codes from GndCover must be in the set of uppercase vegetation codes from the table PlntList. In other words, return PlotID and known vegetation codes from GndCover for initial inventory measurements that are core plots.

#### **8.4.4 An Alternative to Using Collections in Access**

The syntax presented here often runs faster in Access than the syntax described above. This syntax uses the outer join of Access, denoted as **Left Join**. Access also uses the Right Join form of the outer join, which is not presented here. Search the Access on-line help for Left or Right Joins for more information.

#### **Syntax Diagram (Access)**

```
SELECT [distinct] {columnlist, *}  
FROM tablename LEFT JOIN tablename ON columnname = columnname  
WHERE columnname Is Null;
```

#### **LCTA Example 1 (Access)**

```
SELECT DISTINCT AerCover.VEGID  
FROM AerCover LEFT JOIN PLNTLIST ON AerCover.VEGID = PLNTLIST.VEGID  
WHERE PLNTLIST.VEGID Is Not Null;
```

#### **LCTA Example 2 (Access)**

```
SELECT DISTINCT GNDCOVER.PLOTID, GNDCOVER.VEGID  
FROM PLOTSURV  
INNER JOIN (GNDCOVER LEFT JOIN PLNTLIST ON GNDCOVER.VEGID =  
PLNTLIST.VEGID) ON PLOTSURV.PLOTID = GNDCOVER.PLOTID  
WHERE (((PLOTSURV.INVTYPE)="I")  
AND ((PLOTSURV.PLOTTYPE)="C")  
AND ((PLNTLIST.VEGID) Is Null));
```

#### **English Translation 1 (Access)**

Select unique values of vegetation codes from the AerCover table, where the vegetation code is in the set of vegetation codes taken from the table PlntList. The result set of this statement will contain only known vegetation codes.

### English Translation 2 (Access)

Select plot number and vegetation code from the GndCover table, where the plot numbers in the tables GndCover and PlotSurv are equal. Include only plots with an inventory type of 'I' and a plot type of 'C'. In addition, the vegetation codes from GndCover must be in the set of vegetation codes from the table PlntList. In other words, return PlotID and known vegetation codes from GndCover for initial inventory measurements that are core plots. Notice in this example the use of both an inner and outer join. The outer join uses the **Left Join** syntax, "(GNDCOVER **LEFT JOIN** PLNTLIST **ON** GNDCOVER.VEGID = PLNTLIST.VEGID)". The inner join portion is, "**ON** PLOTSURV.PLOTID = GNDCOVER.PLOTID".

#### 8.4.5 The Union Statement

The UNION clause is used to merge the results of two or more SELECT statements. Duplicate rows are eliminated unless the ALL qualifier is used. Views are not allowed in a UNION clause. The number of columns and data types must be the same in each select statement.

UNION is useful when retrieving a set of data, that requires a number of SQL statements, into one output. In some cases OR, in the WHERE condition, is used to get the same results.

#### Syntax Diagram

```
SELECT [distinct] {columnlist,*} FROM tablename
[WHERE condition [AND condition]]
[ORDER BY columnlist]
UNION
SELECT [distinct] {columnlist,*} FROM tablename
[WHERE condition [AND condition]]
[ORDER BY columnlist];
```

#### LCTA Example 1 (Quest and Access)

```
SELECT PlotID, LandUse FROM LandUse
WHERE PlotID = 1
UNION
SELECT PlotID, LandUse FROM LandUse
WHERE PlotID = 10;
```

### **LCTA Example 2 (Quest)**

```
SELECT 'TRAIL ' AS Dist, (gdtrail*100/gdobs) FROM PCSDPlotSum
WHERE AnalYear = 1989
UNION
SELECT 'PASS',(gdpass*100/gdobs) FROM PCSDPlotSum
WHERE AnalYear = 1989
UNION
SELECT 'OTHERTYPE',(gdothet*100/gdobs) FROM PCSDPlotSum
WHERE AnalYear = 1989
ORDER BY 1;
```

### **LCTA Example 2 (Quest)**

```
SELECT "TRAIL" AS Dist, (gdtrail*100/gdobs) FROM PCSDPlotSum
WHERE AnalYear = 1989
UNION
SELECT "PASS", (gdpass*100/gdobs) FROM PCSDPlotSum
WHERE AnalYear = 1989
UNION
SELECT "OTHERTYPE", (gdothet*100/gdobs) FROM PCSDPlotSum
WHERE AnalYear = 1989
ORDER BY Dist;
```

### **English Translation 1 (Quest and Access)**

Return the plot number and land use values for plot number 1 and 10. The resulting set of values from both statements is appended to create one result set.

### **English Translation 2 (Quest and Access)**

Return a user defined text field, named Dist, and the mean trail, pass, and other disturbances for 1989

Example 2, Quest syntax, contains some minor details of the **UNION** clause to be aware of in SQLBase. The defined value for Dist, which is manually defined, uses the **AS** key word to rename the output column. The **AS** key word is only used in the first **SELECT** statement. The length of the first defined variable Dist must be as long or longer than the longest string defined. Note the blank spaces at the end of 'TRAIL'. Also, example 2 uses 1 not Dist in the **ORDER BY** clause. This is a short hand method of listing the variables in SQLBase. The number corresponds to the list of columns following the **SELECT** key word. The short hand method is required in **UNION** clauses.

The Access syntax does not need the additional spaces in the string 'TRAIL'. It also uses Dist as the order by variable, not 1.

Example 2, Quest syntax, uses a mathematical expression to calculate the average trail, pass, and other disturbances per plot from the data summary table PCSDPlotSum. The Plant Cover Surface Disturbance (PCSD) analysis, from the LCTA Program Manager, stores data in the table PCSDPlotSum. GDPass, GDTrail and GDOther are the number of pass, trail, and other disturbance hits on the ground for a plot. GDObs is the number of total disturbance hits on the ground for that plot. Because all of these values are integers, dividing them returns the value of an integer divide, 0. To force SQLBase to perform a non-integer divide the numerator is multiplied by 100, the value returned is the percent bare ground.

In Access, integer division is not a worry. In example 2 the returned value of calculated mean is a percent. Remove (\*100) from the statement to express the value as a decimal.

## 8.5 References

Anderson, A.B., Sprouse, W., Kowalski, D., and Brozka, R. *Land Condition Trend Analysis (LCTA) Data Collection Software Users Manual: Version 1.0*. USACERL ADP Report 95/13, 1995. Champaign, IL.

Anderson, A.B., Sprouse, W., Guertin, P., and Kowalski, D. (1995). *LCTA Users Interface Program Users Manual: Version 1.0*. USACERL ADP Report 95/24. Champaign, IL.

Sprouse, W. and Anderson, A. B. *Land Condition Trend Analysis (LCTA) Program Data Dictionary: Version 1.0*. USACERL ADP Report EN-95/03. Champaign, IL.

Tazik, D.J., Warren, S.D., Diersing, V.E., Shaw, R.B., Brozka, R.J., Bagley, C.F., and Whitworth, W.R. (1992). *U.S. Army Land Condition-Trend Analysis (LCTA) Plot Inventory Field Methods*. USACERL Technical Report N-92/03. Champaign, IL.