Cruise sql cookbook

A collection of sql snippets designed to be used with cruise database files

Table of Contents

[Relationships 3](#_Toc425264688)

[Maintaining Relationships 3](#_Toc425264689)

[Foreign Key Check 3](#_Toc425264690)

[Using JOINs to Find Foreign Key Errors 4](#_Toc425264691)

[Deleting Records with Foreign Key Errors 4](#_Toc425264692)

[General Commands 5](#_Toc425264693)

[Finding out what tables reference each other 5](#_Toc425264694)

[Delete all field data 5](#_Toc425264695)

[Delete data from whole table columns 5](#_Toc425264696)

[Fixes for known issues 7](#_Toc425264697)

[Foreign key error from deleted stratum 7](#_Toc425264698)

[Cutting Unit Table 9](#_Toc425264699)

[Delete Cutting Unit Records 9](#_Toc425264700)

[Stratum Table 10](#_Toc425264701)

[Delete Stratum Records 10](#_Toc425264702)

[Sample Group Table 11](#_Toc425264703)

[Delete Sample Group Records 11](#_Toc425264704)

[Tree Default Value 12](#_Toc425264705)

[Find Tree Default Values that belong to a Sample Group with a different primary product 12](#_Toc425264706)

[Count Tree 13](#_Toc425264707)

[Create Count Records For newly added Cutting Units 13](#_Toc425264708)

[Create Count Records: Tally by Sample Group 13](#_Toc425264709)

[Create Count Records: Tally by Species 14](#_Toc425264710)

[Appendix i: Recommended Software 15](#_Toc425264711)

[SQLite Shell/Command line interface 15](#_Toc425264712)

[SQLite Admin 15](#_Toc425264713)

[SQLiteman 15](#_Toc425264714)

[SQLite Expert 15](#_Toc425264715)

[SQLite Compare 15](#_Toc425264716)

[Appendix ii: SQL JOIN 16](#_Toc425264717)

# Relationships

At the time of writing (July 2015) there are 39 different tables that make up the structure of a cruise file. Each table has a purpose and is a piece of the larger picture that makes up the cruise file. Many of these tables are interconnected using foreign keys as the mechanism that connects them together. To help understand how this interconnected web works let’s look at different types of relationships that make up this web.

**One to Many relationships:**  This can be considered the fundamental building block of all relationships. This is when we say one record has or is linked to zero or more records. In the cruise structure some examples of one to many relationships are log to tree, a tree can have zero or many logs associated with it. Another example is stratum to sample group, a stratum can have one or more sample groups. The participants in this relationship are usually called parent and child, where the parent is the one part of the relationship and the children are the many. The way this relationship is defined in the database is by having a foreign key on the child that references the primary key of the parent.

**Many to Many relationships:** As in life not all relationships can be as one directional as one to many relationships. We have our own friends and they have their own maybe different set of friends; however the limitation of a SQL database doesn’t allow us to keep adding more foreign key fields to records when want to link more records. This is where mapping tables come in. A mapping table is generally just a table with two foreign key fields. Usually it doesn’t have a primary key or other data, but can. In the cruise database mapping tables are named by the two tables they link together, such as: CuttingUnitStratum, SampleGroupTreeDefaultValue, and TreeDefaultValueTreeAuditValue.

# Maintaining Relationships

One of the benefits of a SQL database is that when a foreign key is use to establish a relationship, the database becomes aware of that relationship and has features that help you maintain that relationship.

## Foreign Key Check

The foreign key check command is a useful command for checking the integrity of the data in the database. The database will check all foreign keys to ensure that they point to a record. If there are any errors the result will tell you the child table and the rowed (or primary key value) of the bad record.

TIP: **Use this command frequently whenever you modify foreign key for primary key values using a SQL editor**, **it’s easier to catch errors early.**

pragma foreign\_key\_check;

TIP: Optionaly you can use the command PRAGMA foreign\_keys = ON; to turn foreign key enforcement on. This will prevent modifications you make on the database from creating a foreign key error.

## Using JOINs to Find Foreign Key Errors

The foreign\_key\_check command can be useful, because it can check the whole database at once, but sometimes you may need do pull up the whole record, or integrate your results into other commands. To build a select command that returns all the child records with foreign key errors: Start off with a select on the child table, then we do a left join with the parent table (Note, as you can see in Appendix II a left join will give us all the records in our A table as well as where it overlaps with B) , add a USING clause that says what foreign key field we want to join on, and then tell it to only return records where the matching primary key value is missing (no matching record found)

An example using the Log and Tree tables would look like this. Note that in our where clause we have to specify that we want records where Tree.Tree\_CN is null. This is because in the result from our join, we will have two fields called Tree\_CN, one from our log table and one from our tree table.

SELECT \* FROM Log LEFT JOIN Tree USING (Tree\_CN) WHERE Tree.Tree\_CN IS NULL;

## Deleting Records with Foreign Key Errors

So we just showed how to find records with foreign key errors using a select. Using that we can build a command to delete those records. Delete and update commands don’t allow us to make use of JOINs like select commands do. To tell the delete command what records need to be deleted we will need to feed it the row ids from our select command shown above.

Here is an example of a delete command that incorporates the example above. We will change the select command a little, because we only need the row id (Log\_CN) value of the logs we want to delete rather than all columns (\*).

DELETE FROM Log

WHERE

Log\_CN IN

(SELECT Log\_CN FROM Log LEFT JOIN Tree USING (Tree\_CN) WHERE Tree.Tree\_CN IS NULL);

# General Commands

## Finding out what tables reference each other

With a large number of tables in the database it can become confusing figuring out where all the references point. Here is a command that will find all the references pointing to the stratum table. Replace Stratum with any other table name, to use for other tables

SELECT \* FROM sqlite\_master WHERE type = 'table' and sql GLOB '\*REFERENCES Stratum[ ,]\*';

Finding out what tables a table references is easy. Use the following command; replace table-name with the name of the desired table.

PRAGMA foreign\_key\_list(‘table-name’);

## Delete all field data

UPDATE CountTree Set TreeCount = 0, SumKPI = 0;

UPDATE SampleGroup Set SampleSelectorState = '';

DELETE FROM Log;

DELETE FROM Stem;

DELETE FROM LogStock;

DELETE FROM TreeCalculatedValues;

DELETE FROM Tree;

DELETE FROM Plot;

DELETE FROM ErrorLog WHERE TableName IN ('Tree', 'Plot', 'Log', 'Stem');

UPDATE CuttingUnit Set TallyHistory = NULL;

## Delete data from whole table columns

There are two methods that can be used when deleting data from the columns in a table. The simplest method, useful when only a few columns need to be cleared, is to use an update command. ( See SQLite ref: <http://sqlite.org/lang_update.html> )

UPDATE Tree Set DBH = 0;

The down side to this is that it doesn’t automatically account for the default value of the column (if one exists) and the command will need to be updated as more columns are added to the table.

The next method uses a combination of Insert/Replace and a Select command. This involves two steps. First we use the select command to read the data we want to keep from the table. If the table has a primary key(PK) then the select command should include the PK. The second step we use an INSERT OR REPLACE command to reinsert the data back into the table. The advantages of using this method is that allows you to erase column data by specifying what data you want to keep rather than what data you don’t want to keep, also it allows you to set the data you don’t want to keep back to its default value without requiring you to know what the default value should be.

INSERT OR REPLACE INTO Tree (Tree\_CN, CuttingUnit\_CN, Stratum\_CN, Plot\_CN, TreeNumber, CreatedBy, CreatedDate)

SELECT Tree\_CN, CuttingUnit\_CN, Stratum\_CN, Plot\_CN, TreeNumber, CreatedBy, CreatedDate FROM Tree;

# Fixes for known issues

## Foreign key error from deleted stratum

Prior to Cruise Manager version 8.05 there was a bug when deleting a stratum, would create foreign key errors.

The table where foreign key errors would be created were: SampleGroup, TreeFieldSetup, and LogFieldSetup. Note that there should not be any foreign key errors in the tree table because Cruise Manager doesn’t allow stratum to be deleted if it contains trees, but it is a possibility.

Before we start deleting references to the deleted stratum, check to see if there are and references to deleted strata in the tree, plot or CountTree tables to make sure we are not going to delete useful data.

SELECT \* FROM Tree LEFT JOIN Stratum USING (Stratum\_CN) WHERE Stratum.Stratum\_CN IS NULL;

SELECT \* FROM Plot LEFT JOIN Stratum USING (Stratum\_CN) WHERE Stratum.Stratum\_CN IS NULL;

SELECT \* FROM CountTree JOIN SampleGroup USING (SampleGroup\_CN) LEFT JOIN Stratum USING (Stratum\_CN) WHERE Stratum.Stratum\_CN IS NULL;

Here are the commands that will delete the records with invalid references to the deleted strata. They are in a specific order because some need to be executed before others, but we will turn on foreign key checking to insure that these commands are executed in the proper order.

PRAGMA foreign\_keys = ON;

DELETE FROM Treefieldsetup WHERE rowid IN (SELECT lft.rowid FROM TreeFieldSetup AS lft LEFT JOIN stratum USING (stratum\_cn) WHERE stratum.Stratum\_CN IS NULL);

DELETE FROM logfieldsetup WHERE rowid IN (SELECT lft.rowid FROM LogFieldSetup AS lft LEFT JOIN stratum USING (stratum\_cn) WHERE stratum.[Stratum\_CN] IS NULL);

--next two commands optional, if we need to delete hanging tree or plot records—

DELETE FROM Plot WHERE Plot\_cn in (SELECT Plot.Plot\_CN FROM Plot LEFT JOIN Stratum USING (Stratum\_CN) WHERE Stratum.Stratum\_cn IS NULL);

DELETE FROM Tree WHERE Tree\_cn in (SELECT Tree.Tree\_CN FROM Tree LEFT JOIN Stratum USING (Stratum\_CN) WHERE Stratum.Stratum\_cn IS NULL);

DELETE FROM SampleGroupTreeDefaultValue WHERE SampleGroup\_cn IN (SELECT samplegroup\_cn FROM sampleGroup LEFT JOIN stratum USING (stratum\_cn) WHERE stratum.stratum\_cn IS NULL);

DELETE FROM samplegroup WHERE samplegroup\_cn IN (SELECT samplegroup\_cn FROM sampleGroup LEFT JOIN stratum USING (stratum\_cn) WHERE stratum.stratum\_cn IS NULL);

PRAGMA foreign\_key\_check;--optional, will tell us if all foreign key errors were fixed--

# Cutting Unit Table

## Delete Cutting Unit Records

Delete From CuttingUnitStratum WHERE CuttingUnit\_CN = {0};

DELETE FROM Log WHERE Tree\_CN IN (SELECT Tree\_CN FROM Tree WHERE Tree.CuttingUnit\_CN = {0});

DELETE FROM LogStock WHERE EXISTS (SELECT 1 FROM Tree WHERE Tree.Tree\_CN = LogStock.Tree\_CN AND Tree.CuttingUnit\_CN = {0});

DELETE FROM TreeCalculatedValues WHERE EXISTS (SELECT 1 FROM Tree WHERE Tree.Tree\_CN = TreeCalculatedValues.Tree\_CN AND Tree.CuttingUnit\_CN = {0});

DELETE FROM Tree WHERE CuttingUnit\_CN = {0};

DELETE FROM Plot WHERE CuttingUnit\_CN = {0};

DELETE FROM CountTree WHERE CuttingUnit\_CN = {0};

DELETE FROM CuttingUnit WHERE CuttingUnit\_CN = {0};

# Stratum Table

## Delete Stratum Records

DELETE From CuttingUnitStratum WHERE Stratum\_CN = {0};

DELETE FROM Log WHERE Tree\_CN IN (SELECT Tree\_CN FROM Tree WHERE Tree.Stratum\_CN = {0});

DELETE FROM LogStock WHERE Tree\_CN IN (SELECT Tree\_CN FROM Tree WHERE Tree.Stratum\_CN = {0});

DELETE FROM TreeCalculatedValues WHERE Tree\_CN IN (SELECT 1 FROM Tree WHERE Tree.Stratum\_CN = {0});

DELETE FROM Tree WHERE Stratum\_CN = {0};

DELETE FROM Plot WHERE Stratum\_CN = {0};

DELETE FROM TreeEstimate WHERE CountTree\_CN IN (SELECT CountTree\_CN FROM CountTree JOIN SampleGroup USING (SampleGroup\_CN) WHERE Stratum\_CN = {0});

DELETE FROM CountTree WHERE SampleGroup\_CN IN (SELECT SampleGroup\_CN FROM SampleGroup WHERE SampleGroup.Stratum\_CN = {0});

DELETE FROM SampleGroupTreeDefaultValue WHERE SampleGroup\_CN IN (SELECT SampleGroup\_CN FROM SampleGroup WHERE SampleGroup.Stratum\_CN = {0});

DELETE FROM SampleGroup WHERE Stratum\_CN = {0};

DELETE FROM TreeFieldSetup WHERE Stratum\_CN = {0};

DELETE FROM LogFieldSetup WHERE Stratum\_CN = {0};

DELETE FROM Stratum WHERE Stratum\_CN = {0};

# Sample Group Table

## Delete Sample Group Records

DELETE FROM Log WHERE Tree\_CN IN (SELECT Tree\_CN FROM Tree WHERE Tree.SampleGroup\_CN = {0});

DELETE FROM LogStock WHERE Tree\_CN IN (SELECT Tree\_CN FROM Tree WHERE Tree.SampleGroup\_CN = {0});

DELETE FROM TreeCalculatedValues WHERE Tree\_CN IN (SELECT Tree\_CN FROM Tree WHERE Tree.SampleGroup\_CN = {0});

DELETE FROM Tree WHERE SampleGroup\_CN = {0};

DELETE FROM TreeEstimate WHERE CountTree\_CN IN (SELECT CountTree\_CN FROM CountTree WHERE SampleGroup\_CN = {0});

DELETE FROM CountTree WHERE SampleGroup\_CN = {0};

DELETE FROM SampleGroupTreeDefaultValue WHERE SampleGroup\_CN = {0};

DELETE FROM SampleGroup WHERE SampleGroup\_CN = {0};

# Tree Default Value

## Find Tree Default Values that belong to a Sample Group with a different primary product

SELECT Stratum.Code as StCode,

sg.Code as SGcode,

sg.[PrimaryProduct] as sgPP,

tdv.[PrimaryProduct] as tdvPP,

treedefaultvalue.[Species] FROM SampleGroupTreeDefaultValue

JOIN TreeDefaultValue AS tdv USING (TreeDefaultValue\_CN)

JOIN SampleGroup USING (SampleGroup\_CN)

JOIN stratum USING (Stratum\_CN)

WHERE sampleGroup.[PrimaryProduct] != treedefaultValue.[PrimaryProduct];

# Count Tree

## Create Count Records For newly added Cutting Units

The fallowing command looks for cutting units that don’t have any records in the count tree table, then cross references that with the strata in those units, then finds preexisting count tree records for sample groups in those strata, then creates new count tree records using the preexisting ones as a guide.

To test the output of the command before running it, run the SELECT portion of the command first:

SELECT CuttingUnitStratum.CuttingUnit\_CN, CountTree.SampleGroup\_CN, CountTree.TreeDefaultValue\_CN, CountTree.Tally\_CN, 'script' as cb

FROM CountTree

JOIN SampleGroup USING (SampleGroup\_CN)

JOIN CuttingUnitStratum USING (Stratum\_CN)

WHERE CuttingUnitStratum.CuttingUnit\_CN IN

(SELECT CuttingUnit\_CN FROM CuttingUnit EXCEPT SELECT CuttingUnit\_CN FROM CountTree)

GROUP BY CountTree.SampleGroup\_CN, CountTree.TreeDefaultValue\_CN;

Then run the whole command if the SELECT command output looks correct:

INSERT OR IGNORE INTO CountTree (CuttingUnit\_CN, SampleGroup\_CN, TreeDefaultValue\_CN, Tally\_CN, CreatedBy) SELECT CuttingUnitStratum.CuttingUnit\_CN, CountTree.SampleGroup\_CN, CountTree.TreeDefaultValue\_CN, CountTree.Tally\_CN, 'script' as cb FROM CountTree JOIN SampleGroup USING (SampleGroup\_CN) JOIN CuttingUnitStratum USING (Stratum\_CN) WHERE CuttingUnitStratum.CuttingUnit\_CN IN (SELECT CuttingUnit\_CN FROM CuttingUnit EXCEPT SELECT CuttingUnit\_CN FROM CountTree) GROUP BY CountTree.SampleGroup\_CN, CountTree.TreeDefaultValue\_CN;

## Create Count Records: Tally by Sample Group

note: replace {1} with the SampleGroup\_CN value

INSERT OR Fail INTO CountTree (CuttingUnit\_CN, SampleGroup\_CN, CreatedBy)

Select CuttingUnitStratum.CuttingUnit\_CN, SampleGroup.SampleGroup\_CN, 'NA' AS CreatedBy

From SampleGroup

INNER JOIN CuttingUnitStratum

ON SampleGroup.Stratum\_CN = CuttingUnitStratum.Stratum\_CN

WHERE SampleGroup.SampleGroup\_CN = {1};

## Create Count Records: Tally by Species

note: replace {1} with the SampleGroup\_CN value

INSERT OR IGNORE INTO CountTree (CuttingUnit\_CN, SampleGroup\_CN, TreeDefaultValue\_CN, CreatedBy)

Select CuttingUnitStratum.CuttingUnit\_CN, SampleGroup.SampleGroup\_CN, SampleGroupTreeDefaultValue.TreeDefaultValue\_CN, 'NA' AS CreatedBy

From SampleGroup

INNER JOIN CuttingUnitStratum

ON SampleGroup.Stratum\_CN = CuttingUnitStratum.Stratum\_CN

INNER JOIN SampleGroupTreeDefaultValue

ON SampleGroupTreeDefaultValue.SampleGroup\_CN = SampleGroup.SampleGroup\_CN

WHERE SampleGroup.SampleGroup\_CN = {1};

# Combining Sales

When a cruise is not selling, it may become necessary to combine two or more cruises together to make them more interesting to buyers. Although the script I will show you further on will greatly reduce the amount of work it takes to combine the cruising data from multiple files; it may not be a trivial thing to do overall. Before preparing to combine cruise data you should plan out what the design of the final cruise file will look like and ensure that the new design is compatible with the design of the original cruises. If you are going to combine strata from two files into one you must make sure that the sample groups are compatible.

## Setup

The ideal setup for combining multiple files would be to create a new cruise file with all cutting units and strata you intend to have in your combined sale. Make sure sample group codes matchup between your source files and your new cruise file. The script expects to find a sample group in your destination stratum with the same code as the sample group in your source stratum. When setting up cutting units and stratum, it may be advantageous to create them in numerical order, as you will see when matching up record ID.

## Matching up the record IDs

After you are done setting up your destination cruise file you will need to get the record ID values for the cutting units and strata in both your destination and source files. These values are not visible in Cruise Manager. You will need to open the files in a SQLite editor to find them. Look into the cutting unit and stratum tables and write down the CuttingUnit\_CN and Stratum\_CN values and which Unit/Stratum code they go to. Once you have figured out what record ID values go to what unit or stratum in each of your files, create a map to translate Unit-Stratum record id combinations from each of your source files to your destination file.

## Rubber to the Road

Running this script using *SQLite Expert* (mentioned in the recommended software section) is fairly simple. First open the destination cruise file. Then you will need to attach the source file. When you attach the source file you will need to give it a name. Give it the name “fromDB”. When you execute the script it will pop up a dialog asking you for the values of parameters of the script. Those parameters are @toST, @toUnit, @fromST, and @fromUnit.

## The Beef

INSERT INTO main.Plot

(Plot\_GUID,

Stratum\_CN,

CuttingUnit\_CN,

PlotNumber,

IsEmpty,

Slope,

KPI,

Aspect,

Remarks,

XCoordinate,

YCoordinate,

ZCoordinate,

MetaData,

Blob)

SELECT

Plot\_GUID,

@toST,

@toUnit,

PlotNumber,

IsEmpty,

Slope,

KPI,

Aspect,

Remarks,

XCoordinate,

YCoordinate,

ZCoordinate,

MetaData,

Blob

FROM [fromDB].[Plot] as fromTbl

WHERE CuttingUnit\_CN = @fromUnit AND

Stratum\_CN = @fromST;

INSERT INTO main.Tree

(Tree\_GUID,

TreeDefaultValue\_CN,

Stratum\_CN,

SampleGroup\_CN,

CuttingUnit\_CN,

Plot\_CN,

TreeNumber,

Species,

CountOrMeasure,

TreeCount,

KPI,

STM,

SeenDefectPrimary,

SeenDefectSecondary,

RecoverablePrimary,

HiddenPrimary,

Initials,

LiveDead,

Grade,

HeightToFirstLiveLimb,

PoleLength,

ClearFace,

CrownRatio,

DBH,

DRC,

TotalHeight,

MerchHeightPrimary,

MerchHeightSecondary,

FormClass,

UpperStemDOB,

UpperStemDiameter,

UpperStemHeight,

DBHDoubleBarkThickness,

TopDIBPrimary,

TopDIBSecondary,

DefectCode,

DiameterAtDefect,

VoidPercent,

Slope,

Aspect,

Remarks,

XCoordinate,

YCoordinate,

ZCoordinate,

MetaData,

IsFallBuckScale

)

Select

[fromTbl].Tree\_GUID,

(SELECT [TreeDefaultValue\_CN] FROM [main].[TreeDefaultValue] as t1 WHERE t1.Species = fromTDVTbl.Species AND t1.PrimaryProduct = fromTDVTbl.PrimaryProduct AND t1.LiveDead = fromTDVTbl.LiveDead),

@toST,

(SELECT [SampleGroup\_CN] FROM [main].[SampleGroup] as t2 WHERE t2.[Code] = [fromSGTbl].[Code] AND t2.Stratum\_CN = @toST),

@toUnit,

(Select [Plot\_CN] from [main].[Plot] as t1 where t1.[Plot\_GUID] = [fromPlotTbl].[Plot\_GUID]),

[fromTbl].TreeNumber,

[fromTbl].Species,

[fromTbl].CountOrMeasure,

[fromTbl].TreeCount,

[fromTbl].KPI,

[fromTbl].STM,

[fromTbl].SeenDefectPrimary,

[fromTbl].SeenDefectSecondary,

[fromTbl].RecoverablePrimary,

[fromTbl].HiddenPrimary,

[fromTbl].Initials,

[fromTbl].LiveDead,

[fromTbl].Grade,

[fromTbl].HeightToFirstLiveLimb,

[fromTbl].PoleLength,

[fromTbl].ClearFace,

[fromTbl].CrownRatio,

[fromTbl].DBH,

[fromTbl].DRC,

[fromTbl].TotalHeight,

[fromTbl].MerchHeightPrimary,

[fromTbl].MerchHeightSecondary,

[fromTbl].FormClass,

[fromTbl].UpperStemDOB,

[fromTbl].UpperStemDiameter,

[fromTbl].UpperStemHeight,

[fromTbl].DBHDoubleBarkThickness,

[fromTbl].TopDIBPrimary,

[fromTbl].TopDIBSecondary,

[fromTbl].DefectCode,

[fromTbl].DiameterAtDefect,

[fromTbl].VoidPercent,

[fromTbl].Slope,

[fromTbl].Aspect,

[fromTbl].Remarks,

[fromTbl].XCoordinate,

[fromTbl].YCoordinate,

[fromTbl].ZCoordinate,

[fromTbl].MetaData,

[fromTbl].IsFallBuckScale

FROM [fromDB].Tree AS fromTbl

LEFT JOIN [fromDB].Plot as fromPlotTbl USING (Plot\_CN)

JOIN [fromDB].CuttingUnit as fromUnitTbl USING (CuttingUnit\_CN)

JOIN [fromDB].SampleGroup as fromSGTbl USING (SampleGroup\_CN)

JOIN [fromDB].TreeDefaultValue as fromTDVTbl USING (TreeDefaultValue\_CN)

where fromTbl.CuttingUnit\_CN = @fromUnit AND

fromTbl.Stratum\_CN = @fromST;

INSERT INTO main.LOG

(Log\_GUID,

Tree\_CN,

LogNumber,

Grade,

SeenDefect,

PercentRecoverable,

Length,

ExportGrade,

SmallEndDiameter,

LargeEndDiameter,

GrossBoardFoot,

NetBoardFoot,

GrossCubicFoot,

NetCubicFoot,

BoardFootRemoved,

CubicFootRemoved,

DIBClass,

BarkThickness)

SELECT

[fromTbl].Log\_GUID,

(SELECT Tree\_CN FROM main.Tree WHERE Tree\_GUID = fromTreeTbl.Tree\_GUID),

[fromTbl].LogNumber,

[fromTbl].Grade,

[fromTbl].SeenDefect,

[fromTbl].PercentRecoverable,

[fromTbl].Length,

[fromTbl].ExportGrade,

[fromTbl].SmallEndDiameter,

[fromTbl].LargeEndDiameter,

[fromTbl].GrossBoardFoot,

[fromTbl].NetBoardFoot,

[fromTbl].GrossCubicFoot,

[fromTbl].NetCubicFoot,

[fromTbl].BoardFootRemoved,

[fromTbl].CubicFootRemoved,

[fromTbl].DIBClass,

[fromTbl].BarkThickness

FROM [fromDB].[Log] as fromTbl

JOIN [fromDB].Tree AS fromTreeTbl USING (Tree\_CN)

WHERE fromTreeTbl.CuttingUnit\_CN = @fromUnit

AND fromTreeTbl.Stratum\_CN = @fromST;

INSERT INTO main.Tally

(HotKey, Description)

SELECT HotKey, Description

FROM [fromDB].Tally AS S1

WHERE NOT EXISTS (SELECT 1 FROM main.Tally AS T1 WHERE T1.HotKey = S1.HotKey AND T1.Description = S1.Description)

GROUP BY HotKey, Description;

INSERT INTO main.CountTree

( SampleGroup\_CN,

CuttingUnit\_CN,

Tally\_CN,

TreeDefaultValue\_CN,

TreeCount,

SumKPI)

SELECT

(SELECT [SampleGroup\_CN] FROM [main].[SampleGroup] WHERE [Code] = [fromSGTbl].[Code] AND Stratum\_CN = @toST),

@toUnit,

(SELECT Tally\_CN FROM main.Tally WHERE HotKey = fromTally.HotKey and Description = fromTally.Description),

(SELECT [TreeDefaultValue\_CN] FROM [main].[TreeDefaultValue] as t1 WHERE t1.Species = fromTDVTbl.Species AND t1.PrimaryProduct = fromTDVTbl.PrimaryProduct AND t1.LiveDead = fromTDVTbl.LiveDead),

[fromTbl].TreeCount,

[fromTbl].SumKPI

FROM [fromDB].CountTree AS fromTbl

JOIN [fromDB].SampleGroup AS fromSGTbl USING (SampleGroup\_CN)

LEFT JOIN [fromDB].TreeDefaultValue AS fromTDVTbl USING (TreeDefaultValue\_CN)

JOIN [fromDB].Tally AS fromTally USING (Tally\_CN)

WHERE fromTbl.CuttingUnit\_CN = @fromUnit

AND fromSGTbl.Stratum\_CN = @fromST;

# Appendix i: Recommended Software

## SQLite Shell/Command line interface

The SQLite shell is a command line application that provides very basic interaction with a SQLite database, but it offers some very useful functionality. Most notably it allows the ability to dump table data, can export data in various text formats, such as, CSV. More information on the command line interface can be found here: <http://sqlite.org/cli.html>

Download: <http://sqlite.org/download.html>

## SQLite Admin

A freeware sqlite editor. Has been in beta for long time and my experience with it has been a buggy and may not be able to run all commands.

Download: <http://sqliteadmin.orbmu2k.de>

## SQLiteman

SQLiteman is an open source GUI application with simple interface, but offers some useful features out of the box. It offers the ability export data to CSV, XML, SQL, and Clipboard, as well it has the ability to generate random data. For non-programmers this is probably the best option.

Download: <http://sqliteman.yarpen.cz>

## SQLite Expert

SQLite Expert is another GUI application for editing databases. It offers a paid and free version with some features disabled in the latter. Has many customization options; useful for more experienced users, but may confuse novice users. Allows you to select SQLite library dll, and allows importing from any other ADO database.

Recommended options: Disable “Application is single instance” and “Remember open database” under Options->General->Startup. Add “.cruise” file association under Options->File Associations

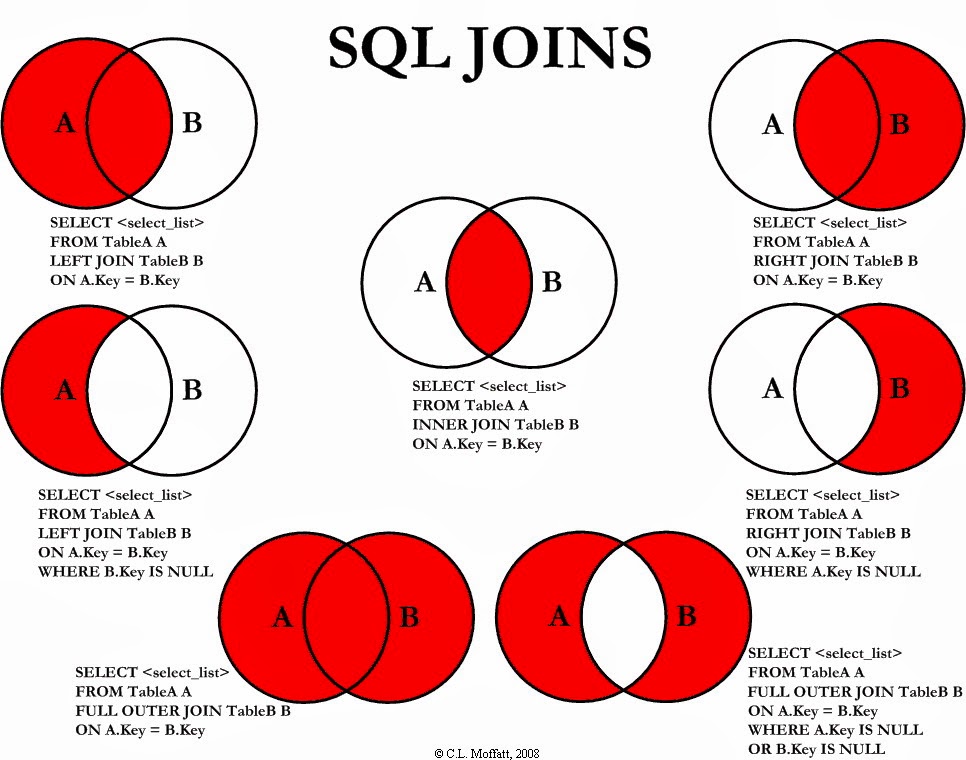
Download: <http://www.sqliteexpert.com>

## SQLite Compare

SQLite Compare is a utility that allows deep comparison between files. It can also be used to move records between files.

Download: <http://www.codeproject.com/Articles/220018/SQLite-Compare-Utility>

# Appendix ii: SQL JOIN



Note: SQLite doesn’t support right or outer joins