**Primer SQL-database**

(Feb 12, 2014 by Ryan Murphy)

**Contents of the primer**

1. Introduction
2. Database structure
   1. Scenario data
   2. Ensemble data
3. Getting started with SQLite
4. SQL basics
   1. Terminal and SQLite
   2. R and SQLite
   3. Example in R
   4. Premade R functions hiding SQL complexity

**1. Introduction**

The output of the SoilWat R wrapper can quickly become very large, e.g., > 200 GB. This output in a spreadsheet format (e.g., excel or csv files) would prevent any productive use of these data without further splitting. However, further splitting would reduce ones ability to quickly compare different data sets. A sql database solves these issues and would be useful to a wide range of people without a lot of overhead in the learning process. With a few simple commands, a graphical interface, or a plug in anyone can quickly analyze data.

SQLite is portable, requires no persistent process, no configuration, and is a transactional SQL database engine. The actual database is nothing more then a file on the hard drive. This makes the database easy to move and use on different operating systems. To learn more about SQLite database visit the website www.sqlite.org.

**2. Database structure**

**2i. Scenario Data**

All the projects have the same structured output. Scenario data is located in one database. The scenario database, which contains all the output tables, file name is “dbTables.sqlite3”. The tables within are the aggregation tables from the runs. It has 21 unique tables, each table representing an aggregated output or grouped output. The standard deviation to these tables is also in the database with a similar name. These tables are dependent on options set before simulation runs. The tables with daily values are the majority.

This database also has the tables containing the information to the header. The header table is a virtual table containing columns from these different tables. These tables are listed below, and should be excluded when using this database.

Tables within dbTables.db

---Overall Aggregates--- The variable output\_aggregates is used to turn on/off the columns of this table.

1. aggregation\_overall\_mean - There are 51 options available, around 47 was set for these projects. They can be grouped into inputs, climate and weather, climatic dryness, climatic control, yearly water balance, daily extreme values, ecological dryness, mean monthly values, and potential regeneration.

---Daily Tables--- The variable output\_aggregate\_daily is used to turn on or off the following outputs

2. aggregation\_doy\_aet\_mean

3. aggregation\_doy\_deepdrainage\_mean

4. aggregation\_doy\_evaporationsoil\_mean

5. aggregation\_doy\_evaporationsurface\_mean

6. aggregation\_doy\_evaporationtotal\_mean

7. aggregation\_doy\_infiltration\_mean

8. aggregation\_doy\_pet\_mean

9. aggregation\_doy\_rain\_mean

10. aggregation\_doy\_runoff\_mean

11. aggregation\_doy\_snowfall\_mean

12. aggregation\_doy\_snowmelt\_mean

13. aggregation\_doy\_snowpack\_mean

14. aggregation\_doy\_SWAatSWPcrit1500kPa\_Mean

15. aggregation\_doy\_swc\_mean

16. aggregation\_doy\_swp\_mean

17. aggregation\_doy\_temperaturemax\_mean

18. aggregation\_doy\_temperaturemin\_mean

19. aggregation\_doy\_totalprecipitation\_mean

20. aggregation\_doy\_transpiration\_mean

21. aggregation\_doy\_vwc\_mean

----Header Tables--- Tables used to generate the header information.

1. sqlite\_sequence

2. runs

3. header

4. run\_labels

5. scenario\_labels

6. sites

7. experimental\_labels

8. treatments

9. simulation\_years

10. weatherfolders

The daily tables are all formatted the same. The first column is the P\_id column which is a identifier for that row. This links that row up to the header table. The columns following correspond to a day of the year. The tables marked with an asterisk above relate to an output that describes soil layers. These tables have an additional column in the header indicating the layer.

**2ii. Ensemble Data**

The other database files which have a naming schema 'dbEnsemble\_Aggregation \*.sqlite3'. The \* represents a table name from above list (e.g. swa, overall, vwc, etc). The name of the database file corresponds to a table. This means that there are 21 different ensemble database files. The tables in that database correspond to the ensemble name and rank level used to generate that table. There are two ensemble families: SRESA2 and SRESB1. There are three ranks: 2,8,15. Three tables are generated for each ensemble family for each rank. This means there should be 18 tables in each ensemble database.

Ensemble Data is ordered when written. The dbTables are not ordered unless you tell it to. Do not assume that the data is ordered.

**3. Getting Started with SQLite**

Download and install the binaries for your OS. Visit www.sqlite.org/download.html to download binaries. Only required if viewing data from terminal.

If you are planning on using R to view the data, you can install the package RSQLite from the package database. From R the command would be, install.packages(“RSQLite”).

Python also has module to interact with a sqlite database. See http://docs.python.org/2/library/sqlite3.html

Ubuntu has a graphical application called sqliteman.

OSX has multiple applications, a comparative list can be found at http://www.barefeetware.com/sqlite/compare/?mlp

Windows multiple options as well for a graphical interface. Some are listed below.

http://sqlitebrowser.sourceforge.net/

http://sqliteadmin.orbmu2k.de/

http://sqlitestudio.one.pl/

**4. SQL basics**

Structured Query Language is an international standard for database manipulation. The following will give a brief overview of getting data out of the database.

SQL is used to Select rows and columns from the database and return them to you.

A basic syntax for selecting all the rows in a table daily.aet is

SELECT \* FROM daily.aet;

“\*” says select all the columns in the table. Using a list of column names in the form (column\_1, column\_2, …..), one could select only the columns needed from the table.

Using WHERE will let you select only the rows that you require.

SELECT \* FROM daily.aet WHERE RunID = 5;

This would select all the rows where the column RunID is equal to five. You can also use these operators, =, <>,>,<,>=,<=, BETWEEN, LIKE, IN.

Select rows between a value.

SELECT \* FROM daily.aet WHERE RunID BETWEEN 5 AND 10;

Select a ensemble family from the table.

SELECT \* FROM daily.aet WHERE Scenario LIKE '%srbsa2%';

Multiple row conditions can be applied by using AND and OR statements.

SELECT P\_id, Label FROM daily.aet WHERE RunID = 10 AND Scenario LIKE '%SRESA2';

Using ORDER BY at the end will sort the data you requested.

SELECT P\_id FROM daily.aet ORDER BY P\_id;

**4i. Terminal and SQLite**

After installing sqlite, sqlite3 command will be available in the terminal. Viewing rows on the terminal is not the best way to look at large sets of data. One can use sqlite3 to export the data to a csv file. Once content with the sql output simple enter this into the sqlite3 session.

> sqlite3 'databasename.db'

.mode csv

.header on

.out file.csv

SELECT STATMENT;

If you need the data from multiple database tables, use sqlite's ATTACH statement;

> sqlite3

sqlite> ATTACH 'dbname1.db' AS X;

sqlite> ATTACH 'dbname2.db' AS Y;

SELECT \* FROM X.daily.aet;

SELECT \* FROM Y.daily.swp;

**4ii. R and SQLite**

R has a package called RSQlite that makes dealing with SQLite database easy. To install type the following into the R session.

install.packages("RSQLite")

To use the package use the following lines

library(RSQLite)

drv <- dbDriver("SQLite")

con <- dbConnect(drv, "path/to/database/file")

List Tables in the database file

dbListTables(con)

To send a query

res <- dbSendQuery(con, "SQL STATEMENT GOES HERE")

To fetch results, n is the number of rows you want, -1 will give you all selected rows.

data <- fetch(res,-1)

Clear results

dbClearResult(res)

The last three steps can be replaced by dbGetQuery(con, “SQL STATMENT”)

This function combines those steps into one function, but this function lacks the ability to keep results and the ability to return only a certain amount of rows.

**4iii. Example in R**

Compare the output for 'TtoAET\_mean' of all sites in region 1 between current and ensemble scenarios

#First lets load RSQLite package and Connect to the database

library(RSQLite)

drv <- dbDriver("SQLite")

con <- dbConnect(drv, "/home/ryan/Documents/Prj06\_VegetationBoundary/1\_PC\_TempDry\_Simulations\_Prj06\_r2mini/4\_Data\_SWOutputAggregated/dbTables\_current.sqlite3")

#Set a list of unwanted tables – header tables.

headerTables <- c("runs","sqlite\_sequence","header","run\_labels","scenario\_labels","sites","experimental\_labels","treatments","simulation\_years","weatherfolders")

#Let see what tables are in the database

Tables <- dbListTables(con)

Tables <- Tables[-which(Tables %in% headerTables)]

Tables

[1] "aggregation\_doy\_AET\_Mean"

[2] "aggregation\_doy\_AET\_SD"

[3] "aggregation\_doy\_DeepDrainage\_Mean"

[4] "aggregation\_doy\_DeepDrainage\_SD"

[5] "aggregation\_doy\_EvaporationSoil\_Mean"

[6] "aggregation\_doy\_EvaporationSoil\_SD"

[7] "aggregation\_doy\_EvaporationSurface\_Mean"

[8] "aggregation\_doy\_EvaporationSurface\_SD"

[9] "aggregation\_doy\_EvaporationTotal\_Mean"

[10] "aggregation\_doy\_EvaporationTotal\_SD"

[11] "aggregation\_doy\_Infiltration\_Mean"

[12] "aggregation\_doy\_Infiltration\_SD"

[13] "aggregation\_doy\_PET\_Mean"

[14] "aggregation\_doy\_PET\_SD"

[15] "aggregation\_doy\_Rain\_Mean"

[16] "aggregation\_doy\_Rain\_SD"

[17] "aggregation\_doy\_Runoff\_Mean"

[18] "aggregation\_doy\_Runoff\_SD"

[19] "aggregation\_doy\_SWAatSWPcrit1500kPa\_Mean"

[20] "aggregation\_doy\_SWAatSWPcrit1500kPa\_SD"

[21] "aggregation\_doy\_SWAatSWPcrit3000kPa\_Mean"

[22] "aggregation\_doy\_SWAatSWPcrit3000kPa\_SD"

[23] "aggregation\_doy\_SWAatSWPcrit3500kPa\_Mean"

[24] "aggregation\_doy\_SWAatSWPcrit3500kPa\_SD"

[25] "aggregation\_doy\_SWAatSWPcrit3900kPa\_Mean"

[26] "aggregation\_doy\_SWAatSWPcrit3900kPa\_SD"

[27] "aggregation\_doy\_SWC\_Mean"

[28] "aggregation\_doy\_SWC\_SD"

[29] "aggregation\_doy\_SWP\_Mean"

[30] "aggregation\_doy\_SWP\_SD"

[31] "aggregation\_doy\_SnowLoss\_Mean"

[32] "aggregation\_doy\_SnowLoss\_SD"

[33] "aggregation\_doy\_Snowfall\_Mean"

[34] "aggregation\_doy\_Snowfall\_SD"

[35] "aggregation\_doy\_Snowmelt\_Mean"

[36] "aggregation\_doy\_Snowmelt\_SD"

[37] "aggregation\_doy\_Snowpack\_Mean"

[38] "aggregation\_doy\_Snowpack\_SD"

[39] "aggregation\_doy\_TemperatureMax\_Mean"

[40] "aggregation\_doy\_TemperatureMax\_SD"

[41] "aggregation\_doy\_TemperatureMin\_Mean"

[42] "aggregation\_doy\_TemperatureMin\_SD"

[43] "aggregation\_doy\_TotalPrecipitation\_Mean"

[44] "aggregation\_doy\_TotalPrecipitation\_SD"

[45] "aggregation\_doy\_Transpiration\_Mean"

[46] "aggregation\_doy\_Transpiration\_SD"

[47] "aggregation\_doy\_VWC\_Mean"

[48] "aggregation\_doy\_VWC\_SD"

[49] "aggregation\_overall\_mean"

[50] "aggregation\_overall\_sd"

#Let us find out the column names in table Aggregation\_Overall\_Mean

colNames <- dbListFields(con,name=Tables[49]) #index for table is 1 in Tables

colNames

[1] "P\_id"

[2] "SWinput\_Soil\_maxDepth\_cm"

[3] "SWinput\_Soil\_soilLayers\_N"

[4] "SWinput\_Soil\_topLayers\_Sand\_fraction"

[5] "SWinput\_Soil\_bottomLayers\_Sand\_fraction"

[6] "SWinput\_Soil\_topLayers\_Clay\_fraction"

…............

#Let get the header columns names minus the P\_id column

headerColumns <- dbListFields(conn=con, "header")[-1]

#Lets add the header and the value we are going to look at

fetchColumns\_wHeader <- c(headerColumns, colNames[which(colNames=="TtoAET\_mean")])

fetchColumns\_noHeader <- c(colNames[which(colNames=="TtoAET\_mean")])

#Lets build our SQL string

SQL<-paste("SELECT ",paste(fetchColumns\_wHeader,sep="",collapse=",")," FROM ",Tables[49]," INNER JOIN header ON ",Tables[49],".P\_id=header.P\_id WHERE Region=1 AND Scenario='Current' ORDER BY header.P\_id;",sep="")

SQL

[1] "SELECT P\_id,RunID,Labels,ID,Region,X\_WGS84,YearStart,SimStartYear,YearEnd,Y\_WGS84,ELEV\_m,Mask\_Current,Mask\_Future,Experimental\_Label,LookupWeatherFolder,PotentialNaturalVegetation\_CompositionShrubsC3C4\_Paruelo1996,PotentialNaturalVegetation\_CompositionShrubs\_Fraction,PotentialNaturalVegetation\_CompositionC3\_Fraction,PotentialNaturalVegetation\_CompositionC4\_Fraction,PotentialNaturalVegetation\_CompositionAnnuals\_Fraction,AdjMonthlyBioMass\_Precipitation,AdjMonthlyBioMass\_Temperature,AdjRootProfile,RootProfile\_C3,RootProfile\_C4,RootProfile\_Shrubs,Vegetation\_TotalBiomass\_ScalingFactor,Vegetation\_Litter\_ScalingFactor,Scenario,TtoAET\_mean FROM Aggregation\_Overall\_Mean WHERE Region=1 AND Scenario='Current';"

#Let Just grab the P\_id for those rows to select our of ensemble.

row\_ids <- dbGetQuery(con, "SELECT P\_id FROM header WHERE Region=1 AND Scenario='Current' ORDER BY P\_id;")

#Put those in a format to use in SQL

row\_ids <- paste("(", paste(unlist(row\_ids,use.names=FALSE),collapse=","),")", sep="")

#Get the data

ScenarioData <- dbGetQuery(con, SQL)

#Get Ensemble Data

#Create a new connection to the ensemble database

Econ <- dbConnect(drv, "/home/ryan/Documents/Prj06\_VegetationBoundary/1\_PC\_TempDry\_Simulations\_Prj06\_r2mini/4\_Data\_SWOutputAggregated/dbEnsemble\_aggregation\_overall.sqlite3")

#Get tables

eTables<-dbListTables(Econ)

eTables

[1] "SRESA2\_rank\_01\_means" "SRESA2\_rank\_01\_sds" "SRESA2\_rank\_02\_means"

[4] "SRESA2\_rank\_02\_sds" "SRESA2\_rank\_03\_means" "SRESA2\_rank\_03\_sds"

#Look at SRESA2 rank 03. We can reuse the column names we made but replace Scenario with EnsembleName and add Level

eSQL <- paste("SELECT ",paste(fetchColumns\_noHeader,sep="",collapse=",")," FROM ",eTables[5]," WHERE P\_id IN ", row\_ids,";",sep="")

#Now get our Data

EnsembleData <- dbGetQuery(Econ,eSQL)

#You can always add the header to the EnsembleData

EnsembleData <- cbind(ScenarioData[1,1:22],EnsembleData)

#Now we can compare a scenario run to the ensemble data

> ScenarioData$TtoAET\_mean[1:5]

[1] 0.2166366

> EnsembleData$TtoAET\_mean[1:5]

[1] 0.1817833 0.1506032 0.1984648 0.1759483 0.1599423

#Write that that data out to file with header information.

write.csv(ScenarioData,file="ScenarioData\_TtoAET\_mean.csv")

write.csv(EnsembleData,file=”EnsembleData\_TtoAET\_mean.csv”)

**4iv. Premade R functions hiding SQL complexity**

Daniel has written some functions that might be useful. This is an example of using those functions.

The low level functions are written in “5\_Database\_Functions.R”. We will be looking at two functions that use those low level functions. These functions are in the “5b\_GTD\_DataAnalysis\_Functions.R” File.

#The function get.SeveralOverallVariables\_ofStudy will return one or more response variables from the overall table in either ensemble or dbTables\_current.sqlite3 database. Use i\_climCat to select ensemble family and rank.

#

#Set some required variables

dir.sana <- "/home/ryan/Documents/Prj06\_VegetationBoundary/1\_PC\_TempDry\_Simulations\_Prj06\_r2mini"

dir.dat <- "/home/ryan/Documents/Prj06\_VegetationBoundary/1\_PC\_TempDry\_Simulations\_Prj06\_r2mini/4\_Data\_SWOutputAggregated"

dir.gis <- "/home/ryan/Documents/Prj06\_VegetationBoundary/1\_PC\_TempDry\_Simulations\_Prj06\_r2mini/GTD\_SharedAnalysis/1\_GISdata"

source("5b\_GTD\_DataAnalysis\_Functions.R")

#View the different options

climCat

Family Rank

Current Current NA

SRESA2\_rank1 SRESA2 1

SRESA2\_rank2 SRESA2 2

SRESA2\_rank3 SRESA2 3

#Get the overall response variable TtoAET\_mean

Scenario\_data <-get.SeveralOverallVariables\_ofStudy(responseName="TtoAET\_mean", i\_climCat=1, whereClause="Region=1")

Ensemble\_data <- get.SeveralOverallVariables\_ofStudy(responseName="TtoAET\_mean", i\_climCat=3, whereClause="Region=1")

#How to read in a whole table ensemble or current

table\_overall <- get.Table\_ofStudy(responseName="overall", i\_climCat=1, whereClause="Region = 1")

table\_overall\_ensemble <- get.Table\_ofStudy(responseName="overall", i\_climCat=3, whereClause="Region = 1")

**5i. Windows GUI SQLiteStudio**

a.) download from <http://sqlitestudio.pl/?act=download>

<http://sqlitestudio.pl/files/free/stable/windows/sqlitestudio-2.1.5.exe>



b.) Save File



c.) Run Sqlitestudio.exe

d.) Connect to databases. We are going to connect to two databases.

1.) dbTables.sqlite3

2.) dbEnsemble\_aggregation\_overall.sqlite3

e.) open SQL editor

f.) View Header by double clicking header under Views

g.) Select column names and make a list separated by coma

header.P\_id AS P\_id,Labels,Region,X\_WGS84,Experimental\_Label,PotentialNaturalVegetation\_CompositionShrubsC3C4\_Paruelo1996,PotentialNaturalVegetation\_CompositionShrubs\_Fraction,PotentialNaturalVegetation\_CompositionC3\_Fraction,PotentialNaturalVegetation\_CompositionC4\_Fraction,PotentialNaturalVegetation\_CompositionAnnuals\_Fraction,AdjMonthlyBioMass\_Precipitation,AdjMonthlyBioMass\_Temperature,AdjRootProfile,RootProfile\_C3,RootProfile\_C4,RootProfile\_Shrubs,Vegetation\_TotalBiomass\_ScalingFactor,Vegetation\_Litter\_ScalingFactor,Scenario

h.) Combine header list with response name and build sql statement in editor. Make sure dbTables\_current.sqlite3 is selected. Hit the execute button in the top left corner.

SELECT header.P\_id AS P\_id,Labels,Region,X\_WGS84,Experimental\_Label,PotentialNaturalVegetation\_CompositionShrubsC3C4\_Paruelo1996,PotentialNaturalVegetation\_CompositionShrubs\_Fraction,PotentialNaturalVegetation\_CompositionC3\_Fraction,PotentialNaturalVegetation\_CompositionC4\_Fraction,PotentialNaturalVegetation\_CompositionAnnuals\_Fraction,AdjMonthlyBioMass\_Precipitation,AdjMonthlyBioMass\_Temperature,AdjRootProfile,RootProfile\_C3,RootProfile\_C4,RootProfile\_Shrubs,Vegetation\_TotalBiomass\_ScalingFactor,Vegetation\_Litter\_ScalingFactor,Scenario,TtoAET\_mean FROM Aggregation\_Overall\_Mean INNER JOIN header ON Aggregation\_Overall\_Mean.P\_id=header.P\_id WHERE Region=1 AND Scenario='Current' ORDER BY header.P\_id;

I.) View results make adjustments to sql statement if needed. Save results if needed. The export results is to the right of the execute button.

This line will show you just the P\_ids

SELECT header.P\_id AS P\_id FROM header WHERE Region=1 AND Scenario='Current' ORDER BY header.P\_id;

Make the following line with the P\_ids

IN (1,34,67,100,133.........)

j.) #Look at SRESA2 rank 03. from dbEnsemble\_aggregation\_overall.sqlite3 . Switch databases in the selector for the editor. Use the IN we made in previous step to make sql statement.

SELECT \* FROM SRESA2\_rank\_03\_means WHERE P\_id IN(1,34,67,100,133....) ORDER BY P\_id;

Export/Save data and then you can compare like the example above.