Database and SQL Workshop

Andrew Marx

2016-03-18

Goals

- A brief background of relational databases and SQL
- How to design databases
- How to access and manipulate data using SQL
- How to use your database with R

Background

Spreadsheets

- Data is stored in tables
- Little/no enforcement of structure or integrity
- Not designed to relate data between tables
- Difficult to directly access data from other applications
- Single user; no user management

(Relational) Databases

- Provide a way to organize and store data in tables
- Allow you to define relationships between your data
- Allow you to insert, modify, and delete data
- Allow you to retrieve information for processing in other applications
- Allow you to manage data access and integrity

Importance of Databases

- Your data is the first step in any process; everything else depends on it
- Properly designed databases reduce potential data issues dramatically
- Makes managing your data easier
- The first step to automating your work (analyses, GIS, generating reports, etc)

Implementations

Single User/Embedded

- Microsoft Access
- SQLite

Multiple User/Backend

- Microsoft SQL Server
- Oracle
- MySQL
- PostgreSQL
- IBM DB2

Who uses these?

- SQLite: https://www.sqlite.org/famous.html
- MySQL: https://www.mysql.com/customers/
- PostgreSQL: http://www.postgresql.org/about/users/

Simply put, there's a reason major businesses, corporations, government agencies, non-profits, etc do not use csv or excel files for storing data.

Why don't more people use databases?

Awareness

- They don't know they exist
- They aren't aware of the advantages

Motivation

- They're easy to learn, but not quite as easy as spreadsheets
- Other people might not be using them
- Transferring existing data

Structured Query Language (SQL)

- Typically pronounced “sequel”
- An international standard: ISO/IEC 9075
- Multiple revisions
- Specifically designed for accessing and manipulating relational databases

Caveats

- Different implementations (SQLite, MySQL, etc) might support different versions of the SQL standard
- The standards are not enforced
- Certain parts of the standard leave it open to vendors to decide implementation details
- Some parts of the standard are optional
- Vendors may add their own extensions

Generally speaking, what you are going to learn here is broadly applicable to any SQL implementation. However, being aware that there can be differences is important, especially if searching for help or answers online

Database Design

Tables

- Tables are used to store and organize data
- Each table represents a category or 'entity type' (e.g., capture data)
- Each row, or **record**, represents a unique instance of that entity (e.g., a single capture)
- Each column, or **field**, represents a unique attribute of that entity (e.g., date, species, sample id, etc).
- Other examples:

Table	Record (row)	Fields (columns)
People	A single person	Name, age, height, phone number
Products	A single product	Product name, serial number, price
States	A single state	Name, year of statehood, governor, state flower

Table Design

- Planning ahead is important, and will save time later
- Consistent and clear naming of the table fields is important
- Following database normalization rules

Normalization Rules

- https://en.wikipedia.org/wiki/Database_normalization
- Process for organizing fields and tables
- Eliminates redundancy
- Eliminates data anomalies
- Makes future changes to table design easier

Normalization Rules

- Every row should be unique (no duplicates)
- There should be no top-to-bottom ordering of rows
- There should be no left-to-right ordering of columns
- Columns in a table should not be calculated from other columns in that table

Normalization Rules

- Every value entered into a column should be atomic (indivisible)
- Whether a value is atomic or not can depend on the situation

Name
Shonda Bendel
Matt Apple

VS

FirstName	LastName
Shonda	Bendel
Matt	Apple

Date
2016-06-19
2014-12-21

VS

Year	Month	Day
2016	06	19
2014	12	21

Normalization Rules

- For each row, only a single value should be entered into a column
- E.g., no comma separated values

Bad

Student	Classes
Chu Turco	History
Kip Belle	Math, English
Winnie Gully	English, History

Good

Student	Classes
Chu Turco	History
Kip Belle	Math
Kip Belle	English
Winnie Gully	English
Winnie Gully	English

Normalization Rules

- Subsets of your columns should not contain corresponding values
- Split into multiple tables

Bad (1 table)

Student	StudentID	Age	Course
Marlon Story	0912	14	Math
Marlon Story	0912	14	English
Shonda Bendel	4325	13	Math

Good (2 tables)

Student	StudentID	Age
Marlon Story	0912	14
Shonda Bendel	4325	13

Student	Course
Marlon Story	Math
Marlon Story	English
Shonda Bendel	Math

Other Table Design Rules

- You should not have multiple columns with the same type of data.

Bad

Student	Class1	Class2	Class3	Class4
Deanna	Math	History	English	Biology
Gordon	Biology			
Lisa	Math	English	History	

Good

Student	Class
Deanna	Math
Deanna	History
Deanna	English
Deanna	Biology
Gordon	Biology
Lisa	Math
Lisa	English
Lisa	History

Keys

- Keys refer to a set of zero or more columns
- Technically, you may be able to make your database work without keys, but you then lose some potentially major advantages of using databases
- Several types of keys:
  - Super key
  - Candidate key
  - Primary key
  - Alternate key
  - Composite key
  - Compound key
  - Unique key
  - Foreign key
- Most of these are esoteric. You only really need to understand Primary and Foreign Keys.

Primary Key

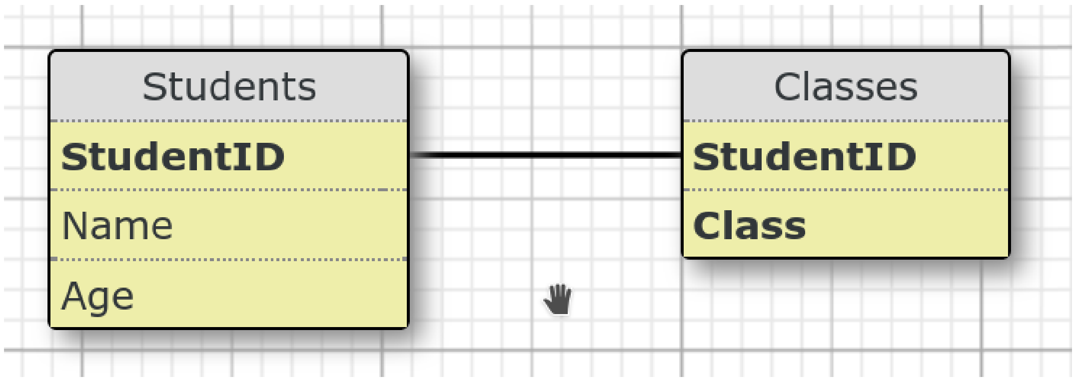
- Can be a single column, or combination of columns (composite primary key)
- Every value or combination of values in a primary key must be unique and not null
- Every table should have a primary key

Foreign Key

- Column that references a column (typically a primary key) of another table
- Used to create a parent-child relationship between tables
- Enforces referential integrity
- Propagates changes throughout the database

Visualizing Keys

- Two tables (Students, Classes)
- Bold = primary key
- Line = foreign key relationship
- Students has a single column primary key
- Classes has a composite primary key, but a single column foreign key
- In this case, Students is a parent table, and Classes is a child table. In other words, Classes depends on the Students table



StudentID	Name	Age
1924	Marlon Story	14
6784	Shonda Bendel	13

StudentID	Course
1924	Math
1924	English
6784	Math

Design Practice 1

- Let's design a database that contains data about people and their pets
- Examples of data that we might be interested in:
  - The person's name
  - Date of birth
  - Sex
  - Address(s)
  - Phone number(s)
  - Email(s)
  - Pet(s)

Design Practice 2

- Let's design a database that contains data about schools, courses, employees/faculty, and students
- Examples of data that we might be interested in:
  - School name
  - Address
  - Course title
  - Course numbers
  - Names
  - IDs
  - etc

Structured Query Language

Components of SQL

- Data Definition Language
- Data Manipulation Language
- Data Control Language

Data Control Language

- Used to control how users can access and modify different aspects of the database and its data
- Basic Commands:
  - GRANT
  - REVOKE

Included for reference. It's very unlikely you will ever actually be in a situation where you have to use these commands

Data Definition Language

- Used to define the database schema, or structure
- How things like tables, views, etc are created and modified
- Basic Commands:
  - CREATE
  - ALTER
  - RENAME
  - DROP
  - TRUNCATE

Data Manipulation Language

- Used to add and modify data in the database
- Commands:
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - MERGE

Data Types

- Different implementations specify slightly different data types
- Generally, data types will fall along these lines:
  - BOOLEAN
  - INT, INTEGER, UNSIGNED INT, BIGINT, etc
  - DECIMAL, FLOAT, REAL, NUMERIC, DOUBLE, etc
  - STRING, CHARACTER, VARCHAR, TEXT, etc
  - BINARY, BLOB, RAW, etc
- Most of these are just specialized versions of a couple core types

NULL

- NULL is a special marker to indicate the absence of a value in a field.
- It is ***not*** a value. I.e., it is not equivalent to 0 or an empty string
- Missing or empty values should be represented with NULL, not dummy values
- This concept is important for comparing values or doing calculations

SQL Syntax

- SQL keywords (SELECT, UPDATE, WHERE, etc) are case insensitive
- Table and column names may or may not be case sensitive, depending on the implementation. Typically not
- Technically, commands should end with a semicolon (;), but some implementations do not require it
- Order of commands matter

Making SQLite Databases

SQLite Manager

- Just one possible option for graphically managing sqlite databases
- As a plugin for Mozilla Firefox, it runs on all platforms with feature parity
- Instructions here apply specifically to SQLite Manager, but the concepts are broadly applicable

Creating Tables

- To create a table, right click on Tables in the left pane, and select the Create Table option
- For SQLite, we can just stick to INTEGER, REAL, and TEXT data types
- If not specified, columns in SQLite default to the TEXT data type

Importing Data

- Data can be imported from CSV files
- Can be imported into either an existing table, or a new table
- To import data, go to *Database* → *Import*
- By default, the table name will be set to the file name
- If the table doesn't exist, it will be created. If it does exist, the data will be added to it
- If an error occurs while trying to import the data, none of the data in the file will be added

Enabling Foreign Keys

- Foreign keys are not enabled in SQLite by default
- To enable them permanently in SQLite Manager:
  - Go to *Tools* > *Open On-Connect SQL Tab*
  - Enter *PRAGMA foreign_keys=ON;* into the On-Connect SQL tab
- Foreign keys can be added to either new or existing tables
- https://www.sqlite.org/foreignkeys.html

Adding Foreign Keys To A New Table

- To add a foreign key to a new table:
  - Right click “Tables” in the side pane, then click “Create Table”
  - Append REFERENCES *table*(*column*) to the data type
  - Include ON UPDATE CASCADE to allow changes to the data
- This is the recommend way to add keys to a table in SQLite Manager
- This is the only way to create multiple foreign keys in SQLite Manager (This is an issue with SQLite Manager itself, not SQLite in general)

Adding Foreign Keys To Existing Tables

- To add a foreign key to an existing table:
  - Select the table in the side pane, then click the structure tab
  - Select the field of interest, right click, edit
  - Append REFERENCES *table*(*column*) after the data type and clear the default value field
  - Include ON UPDATE CASCADE to allow changes to the data
- Because of the way SQLite manager works, only a single foreign key can be added to an existing table

Enabling Automatic Backups

- SQLite Manager lets you enable automatic, timestamped backups
- The backups are created in the same directory as the original, and uses a timestamp that allows chronological sorting
- Can be set to occur automatically or provide a prompt when the database is opened
- To enable:
  - Go to *Tools* → *Options*
  - Under the 'Main' tab, change the value of “Create a timestamped backup...”

SQL Queries

Queries

- Used to retrieve a result set of data from your database
- The results are stored in a virtual table in memory
  - They are not stored or saved physically on the hard drive
- Basic template of a single table query:

  SELECT *column(s)*

  FROM *table*

  WHERE *condition(s)*

  GROUP BY *column(s)*

  ORDER BY *columns(s)*

SELECT

- SELECT and FROM form the core of all SQL data queries

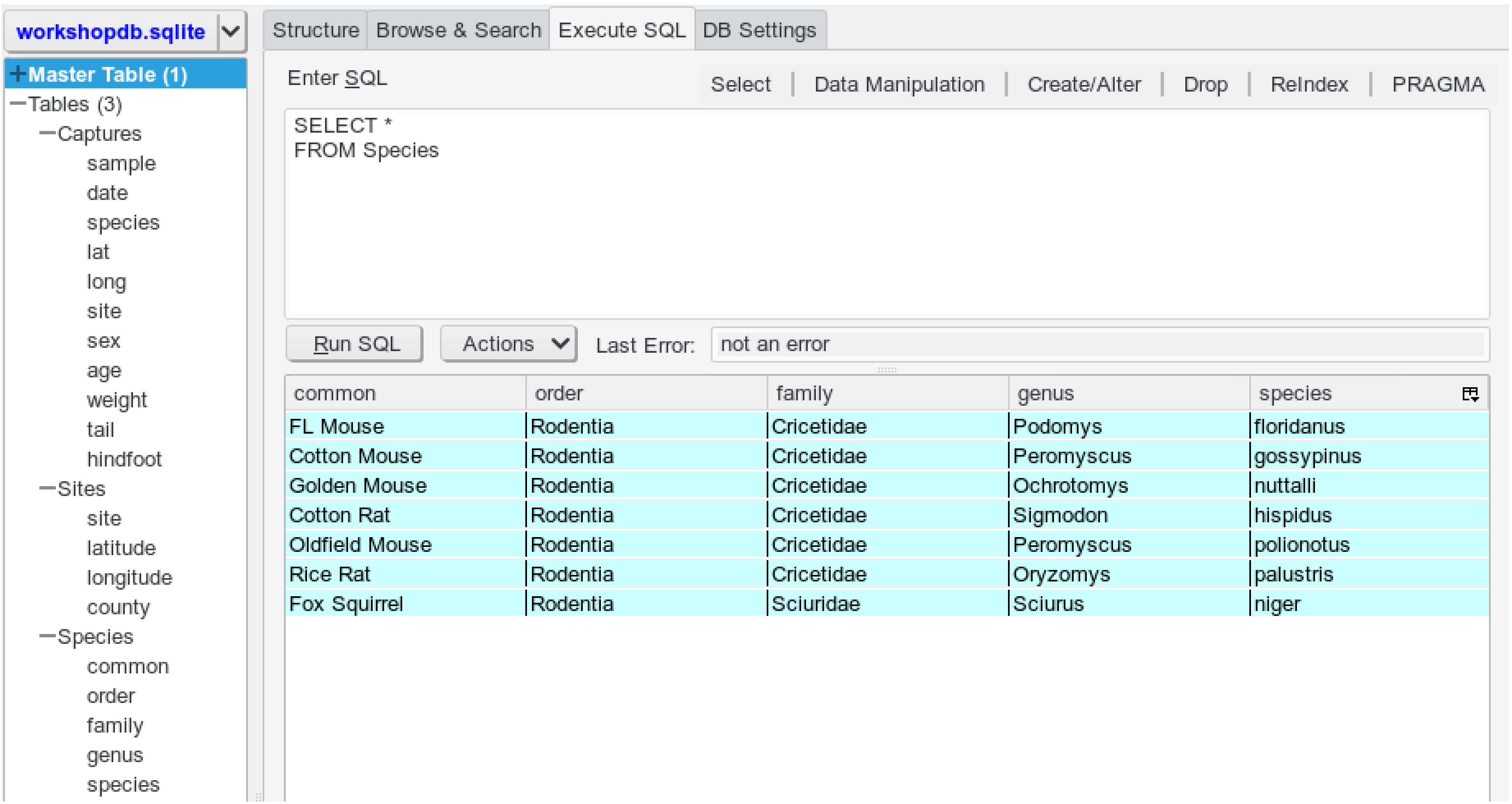
    SELECT *column(s)*

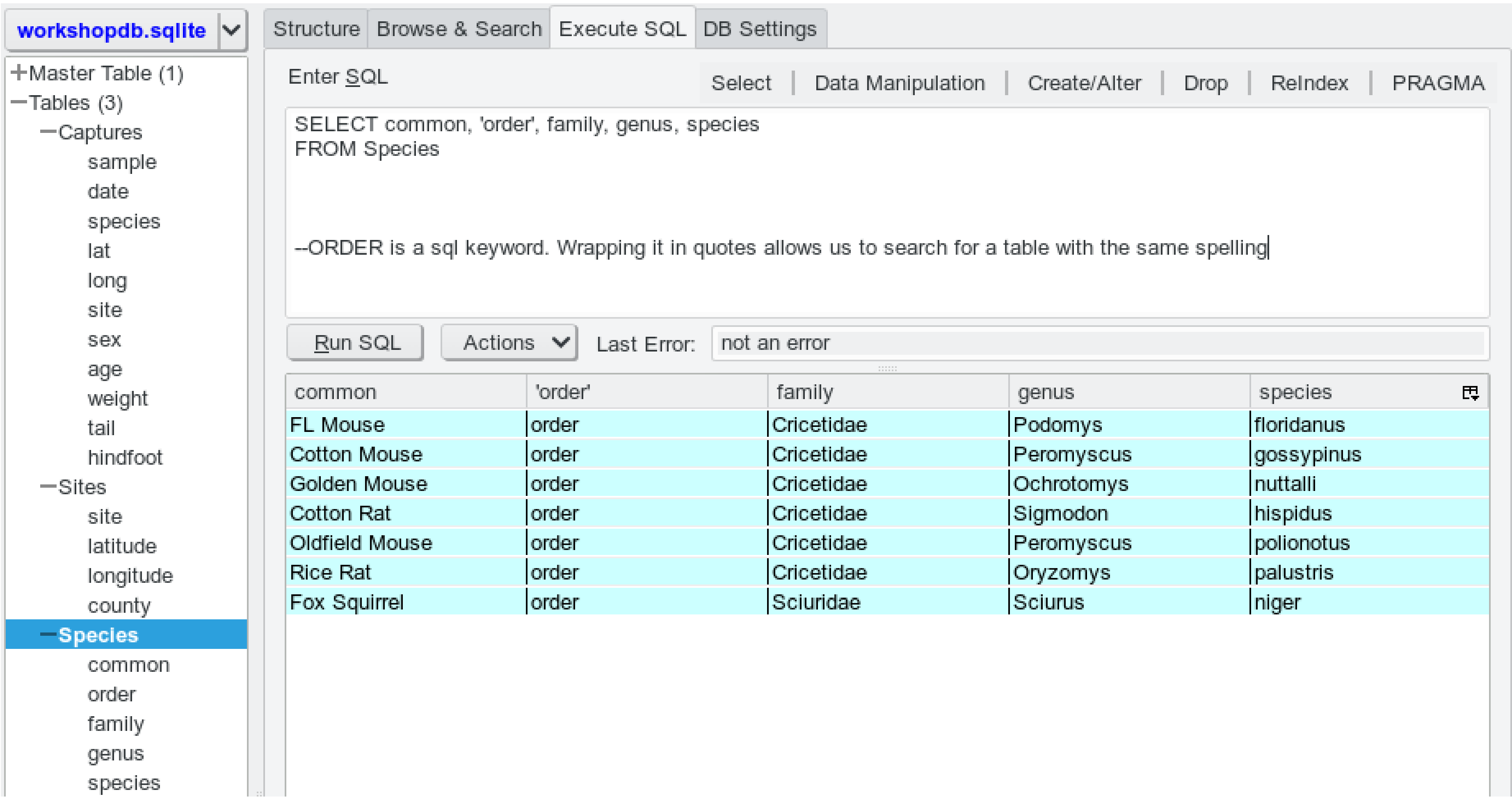
    FROM *table*

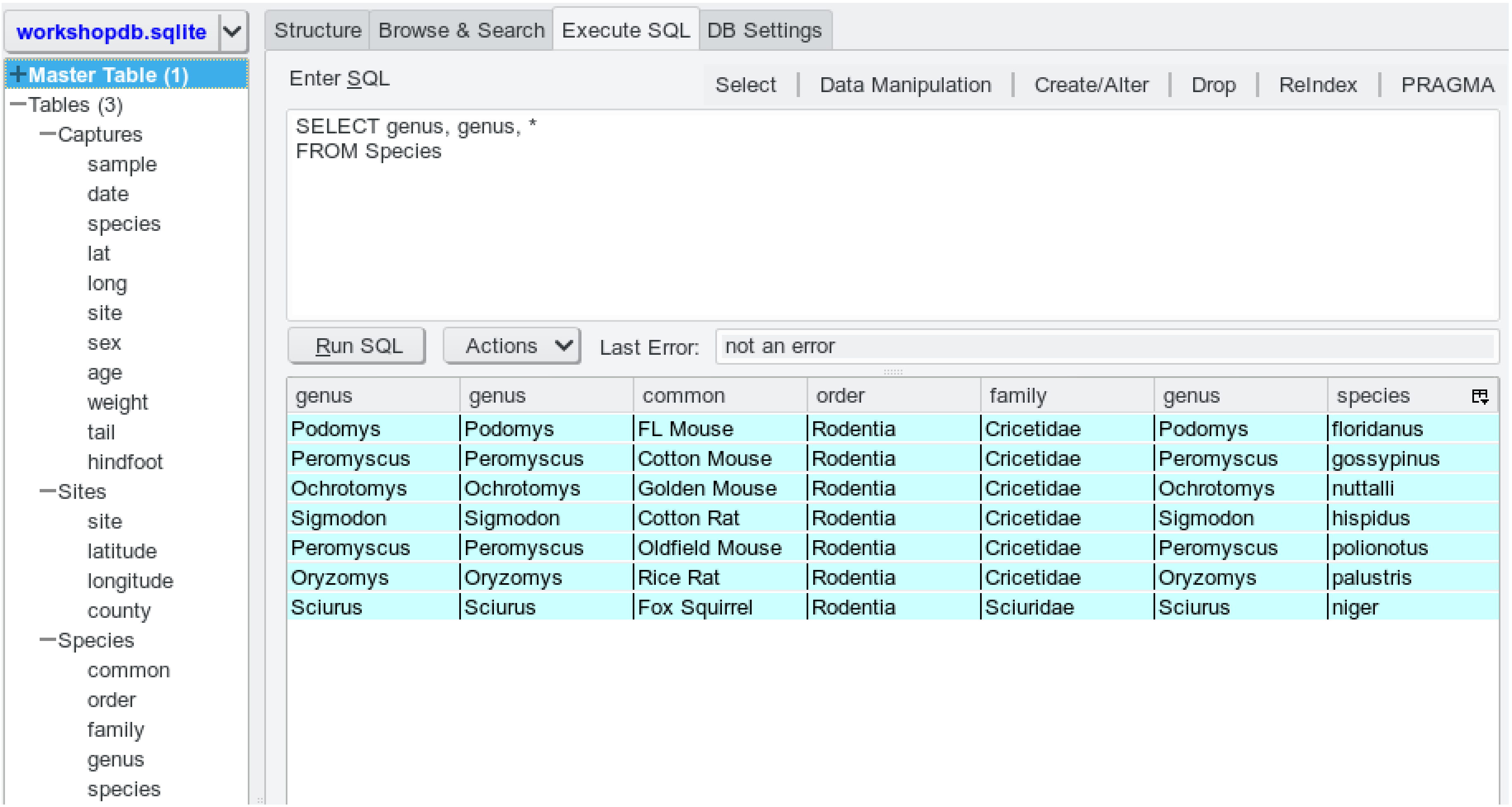
- *columns(s)*: a comma separated list of the fields to include in the results

    -Using an asterisk (*) in place of a column name returns all columns

- *table*: specifies the table to retrieve the fields from


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT *
FROM Species
Run SQL
Actions
Last Error:
not an error
common order family genus species
FL Mouse Rodentia Cricetidae Podomys floridanus
Cotton Mouse Rodentia Cricetidae Peromyscus gossypinus
Golden Mouse Rodentia Cricetidae Ochrotomys nuttalli
Cotton Rat Rodentia Cricetidae Sigmodon hispidus
Oldfield Mouse Rodentia Cricetidae Peromyscus polionotus
Rice Rat Rodentia Cricetidae Oryzomys palustris
Fox Squirrel Rodentia Sciuridae Sciurus niger


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT common, 'order', family, genus, species
FROM Species
--ORDER is a sql keyword. Wrapping it in quotes allows us to search for a table with the same spelling
Run SQL
Actions
Last Error:
not an error
common
'order'
family
genus
species
FL Mouse
order
Cricetidae
Podomys
floridanus
Cotton Mouse
order
Cricetidae
Peromyscus
gossypinus
Golden Mouse
order
Cricetidae
Ochrotomys
nuttalli
Cotton Rat
order
Cricetidae
Sigmodon
hispidus
Oldfield Mouse
order
Cricetidae
Peromyscus
polionotus
Rice Rat
order
Cricetidae
Oryzomys
palustris
Fox Squirrel
order
Sciuridae
Sciurus
niger


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT genus, genus, *
FROM Species
Run SQL
Actions
Last Error:
not an error
genus genus common order family genus species
Podomys Podomys FL Mouse Rodentia Cricetidae Podomys floridanus
Peromyscus Peromyscus Cotton Mouse Rodentia Cricetidae Peromyscus gossypinus
Ochrotomys Ochrotomys Golden Mouse Rodentia Cricetidae Ochrotomys nuttalli
Sigmodon Sigmodon Cotton Rat Rodentia Cricetidae Sigmodon hispidus
Peromyscus Peromyscus Oldfield Mouse Rodentia Cricetidae Peromyscus polionotus
Oryzomys Oryzomys Rice Rat Rodentia Cricetidae Oryzomys palustris
Sciurus Sciurus Fox Squirrel Rodentia Sciuridae Sciurus niger

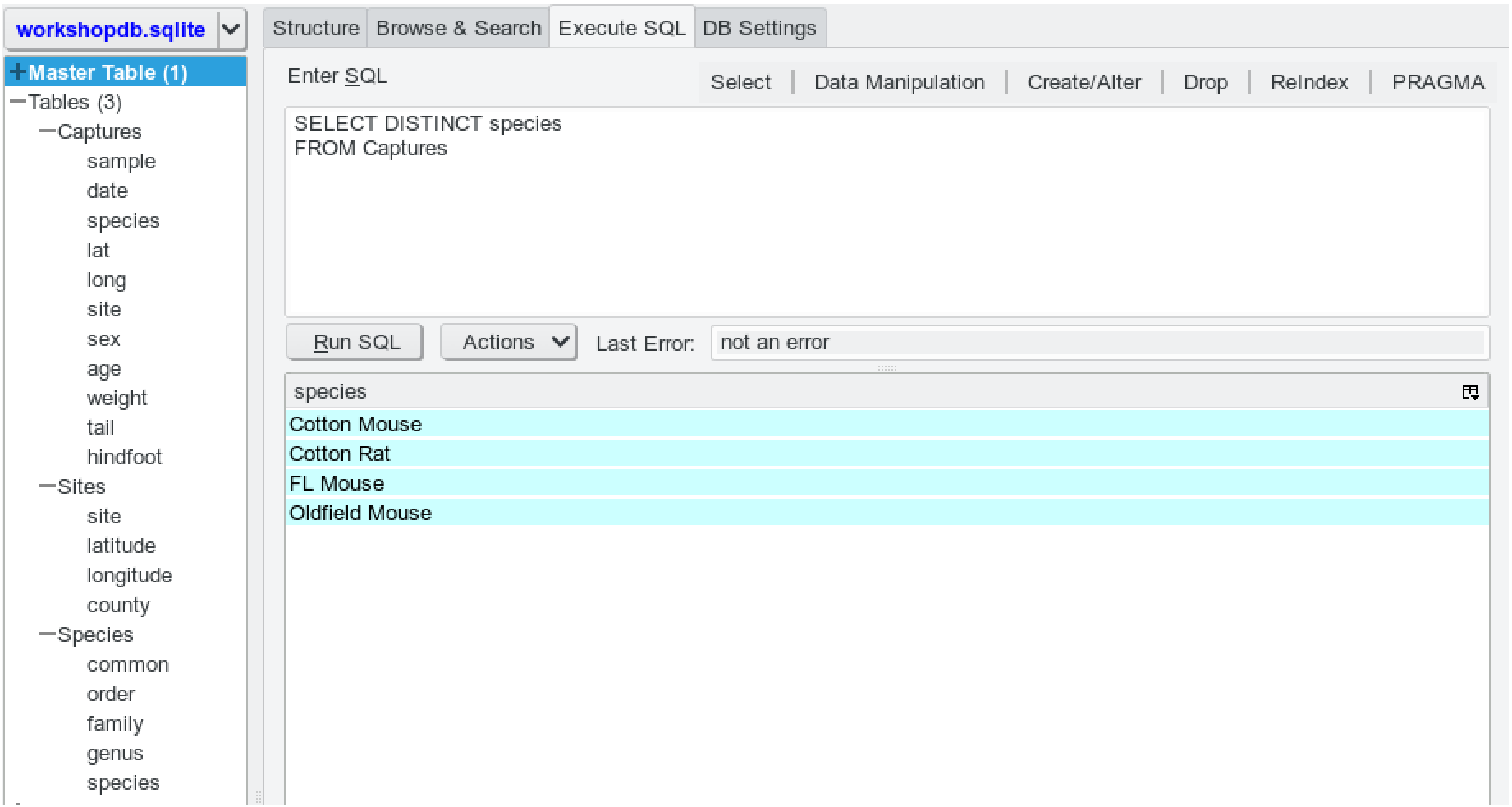
Views

- A virtual table
- The data isn't actually stored physically on the hard drive
- You can't directly modify the data in a view
- Think of it as a way to store a query
- You can query data from a view just like you would from a table
- To create a view in SQLite Manager, right click on 'Views' in the left pane, and select 'Create View'

SELECT – DISTINCT

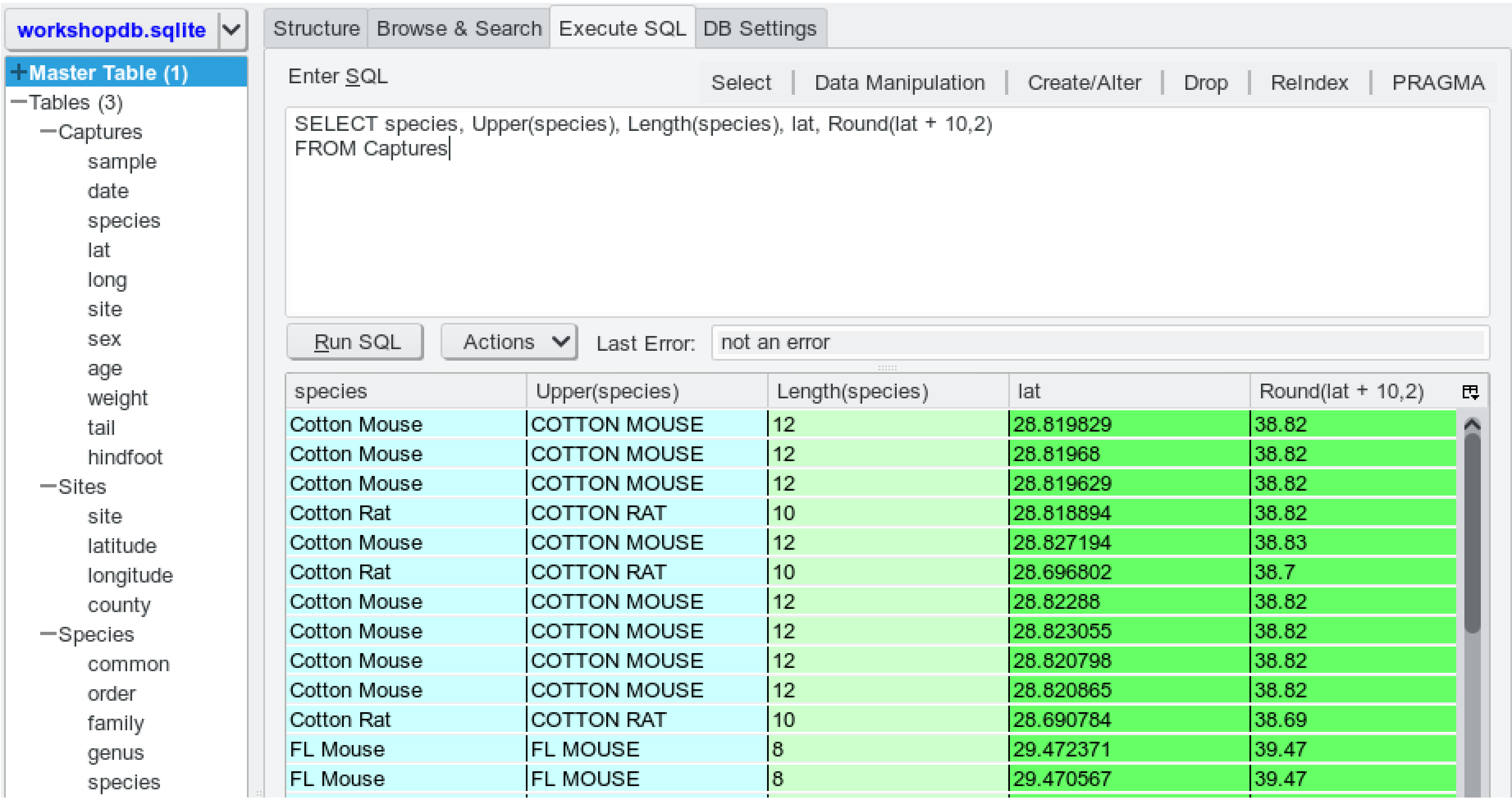
- The DISTINCT keyword eliminates duplicate results

```
SELECT DISTINCT column(s)
FROM tableName
```


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT DISTINCT species
FROM Captures
Run SQL
Actions
Last Error:
not an error
species
Cotton Mouse
Cotton Rat
FL Mouse
Oldfield Mouse

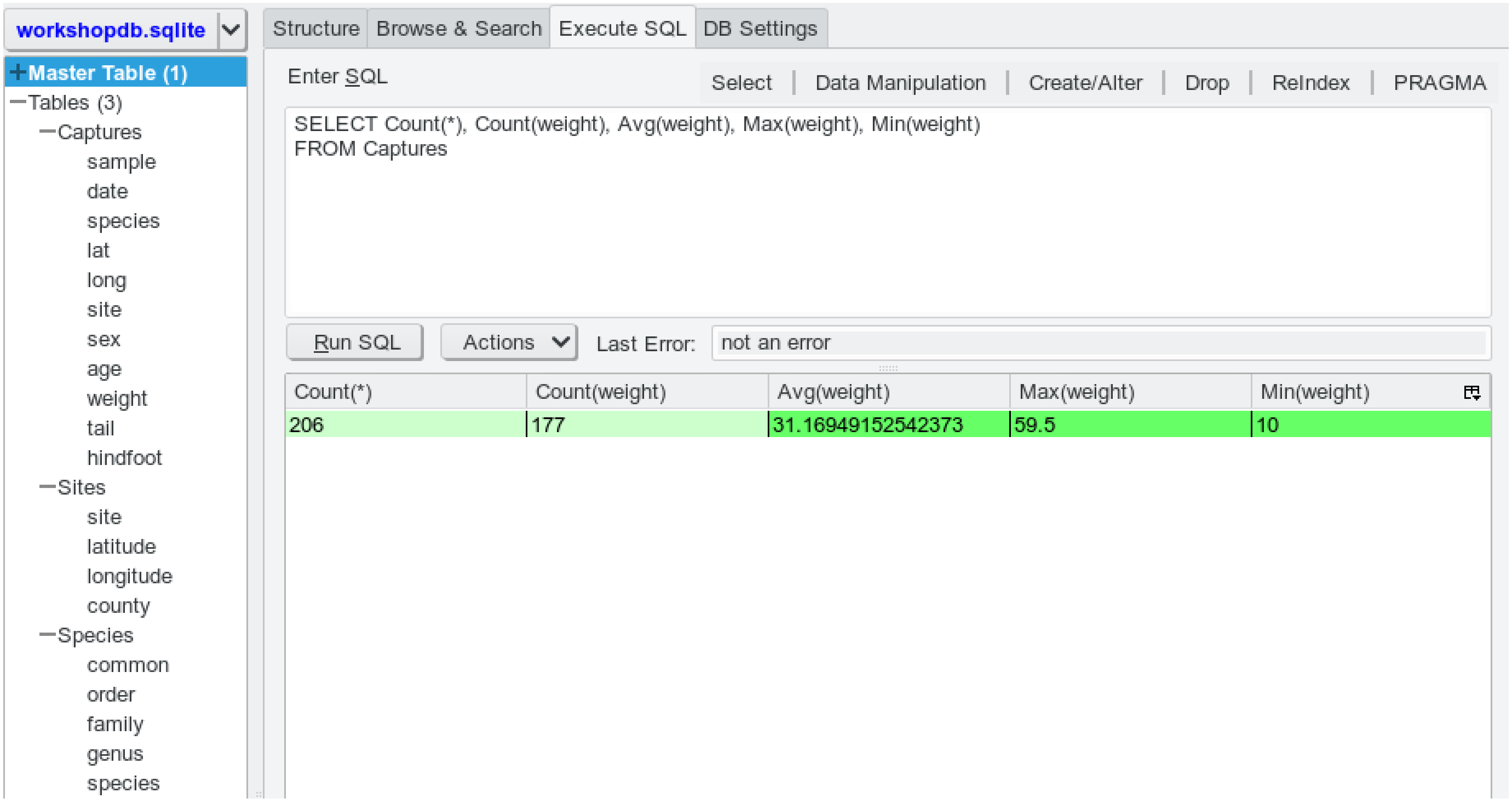
SELECT – Scalar Functions

- Used to modify individual values
- Common scalar functions:
  - Abs(X) returns absolute value of a number
  - Length(X) returns the length of a string
  - Lower(X) converts a text value to all lower case
  - Upper(X) converts a text value to all upper case
  - Min(X,Y,..) returns the argument with the lowest value
  - Max(X,Y,..) returns the argument with the highest value
  - Round(X,Y) rounds a number to Y digits after the decimal place
- Arithmetic operators can also be performed (+,-,*,/,%)
- Other implementations may spell these differently (e.g., upper() vs ucase())
- A full list of SQLite scalar functions: https://www.sqlite.org/lang_corefunc.html
- Note that Min() and Max() are aggregate functions when only passed a single argument


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT species, Upper(species), Length(species), lat, Round(lat + 10,2)
FROM Captures
Run SQL
Actions
Last Error:
not an error
species | Upper(species) | Length(species) | lat | Round(lat + 10,2)
Cotton Mouse | COTTON MOUSE | 12 | 28.819829 | 38.82
Cotton Mouse | COTTON MOUSE | 12 | 28.81968 | 38.82
Cotton Mouse | COTTON MOUSE | 12 | 28.819629 | 38.82
Cotton Rat | COTTON RAT | 10 | 28.818894 | 38.82
Cotton Mouse | COTTON MOUSE | 12 | 28.827194 | 38.83
Cotton Rat | COTTON RAT | 10 | 28.696802 | 38.7
Cotton Mouse | COTTON MOUSE | 12 | 28.82288 | 38.82
Cotton Mouse | COTTON MOUSE | 12 | 28.823055 | 38.82
Cotton Mouse | COTTON MOUSE | 12 | 28.820798 | 38.82
Cotton Mouse | COTTON MOUSE | 12 | 28.820865 | 38.82
Cotton Rat | COTTON RAT | 10 | 28.690784 | 38.69
FL Mouse | FL MOUSE | 8 | 29.472371 | 39.47
FL Mouse | FL MOUSE | 8 | 29.470567 | 39.47

SELECT – Aggregate Functions

- Used to return a single value that is calculated from a column
- Common aggregate functions:
  - Count(*column*) returns the number of rows (see below)
  - Max(*column*) returns the largest value in a column
  - Min(*column*) returns the smallest value in a column
  - Sum(*column*) returns the sum of all values in a column
  - Avg(*column*) returns the average of all values in a column
- Be aware of how NULL values are used in each function
  - E.g., Count(*) returns the number of rows in the table and Count(*column*) returns the number of rows with non null-values in a column


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT Count(*), Count(weight), Avg(weight), Max(weight), Min(weight)
FROM Captures
Run SQL
Actions
Last Error:
not an error
Count(*)
Count(weight)
Avg(weight)
Max(weight)
Min(weight)
206
177
31.16949152542373
59.5
10

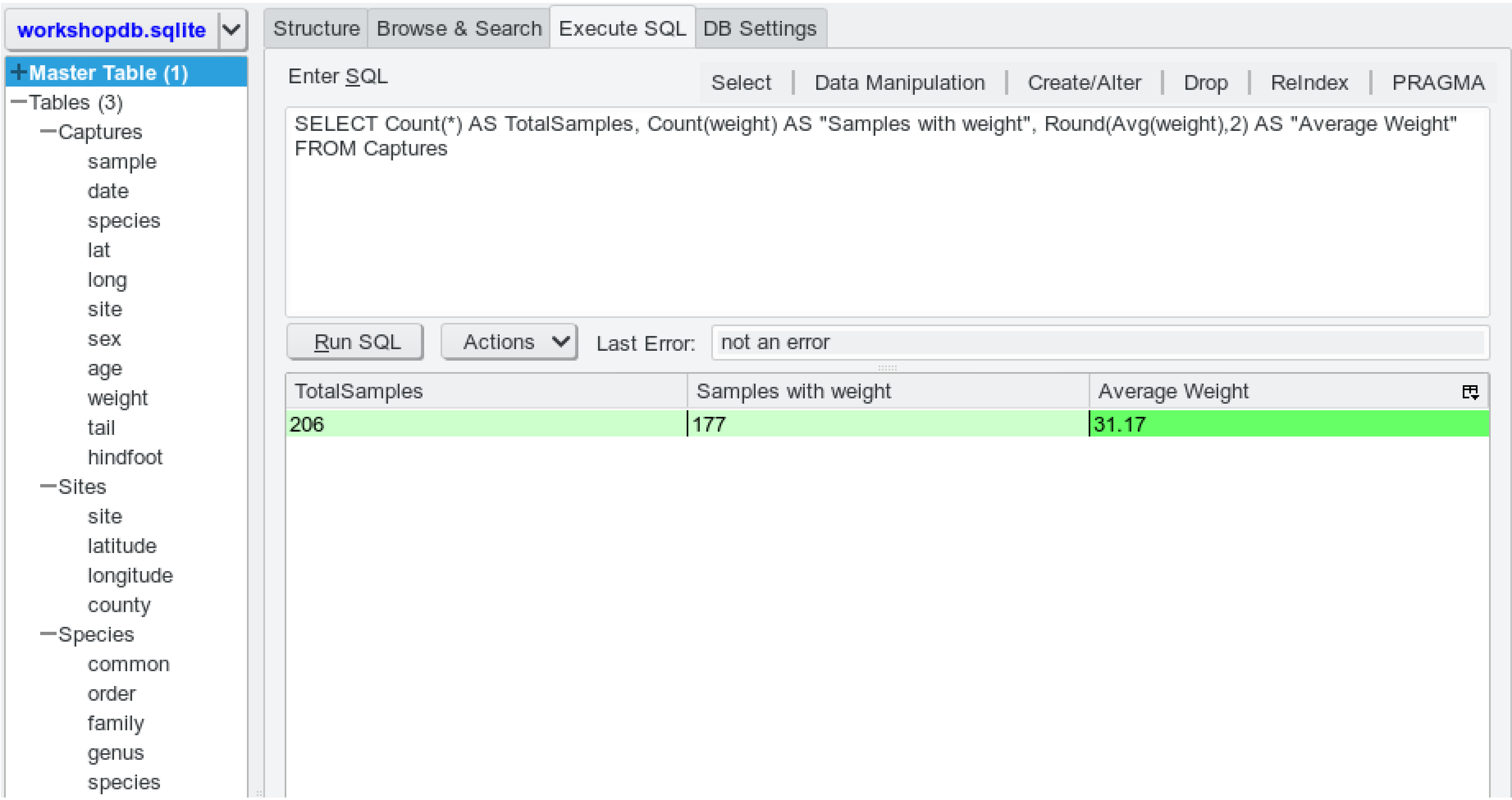
SELECT – AS

- The AS keyword allows you to rename tables and/or fields in your query

    SELECT *column1* AS *newColumn1, …*

    FROM *table* AS *newTable*

- When renaming fields, quotes are only necessary if the new name has spaces


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT Count(*) AS TotalSamples, Count(weight) AS "Samples with weight", Round(Avg(weight),2) AS "Average Weight"
FROM Captures
Run SQL
Actions
Last Error:
not an error
TotalSamples
Samples with weight
Average Weight
206
177
31.17

SELECT – JOIN

- The JOIN keyword is used to combine rows from two or more tables
- There are different types:
  - CROSS JOIN
  - INNER JOIN
  - OUTER JOIN
- INNER JOIN is by far the most common, and is the default when a type of JOIN isn't specified in SQLite
- When working with multiple tables, it may be necessary to specify which table a column belongs to, e.g.:

  SELECT *table.column, ...*

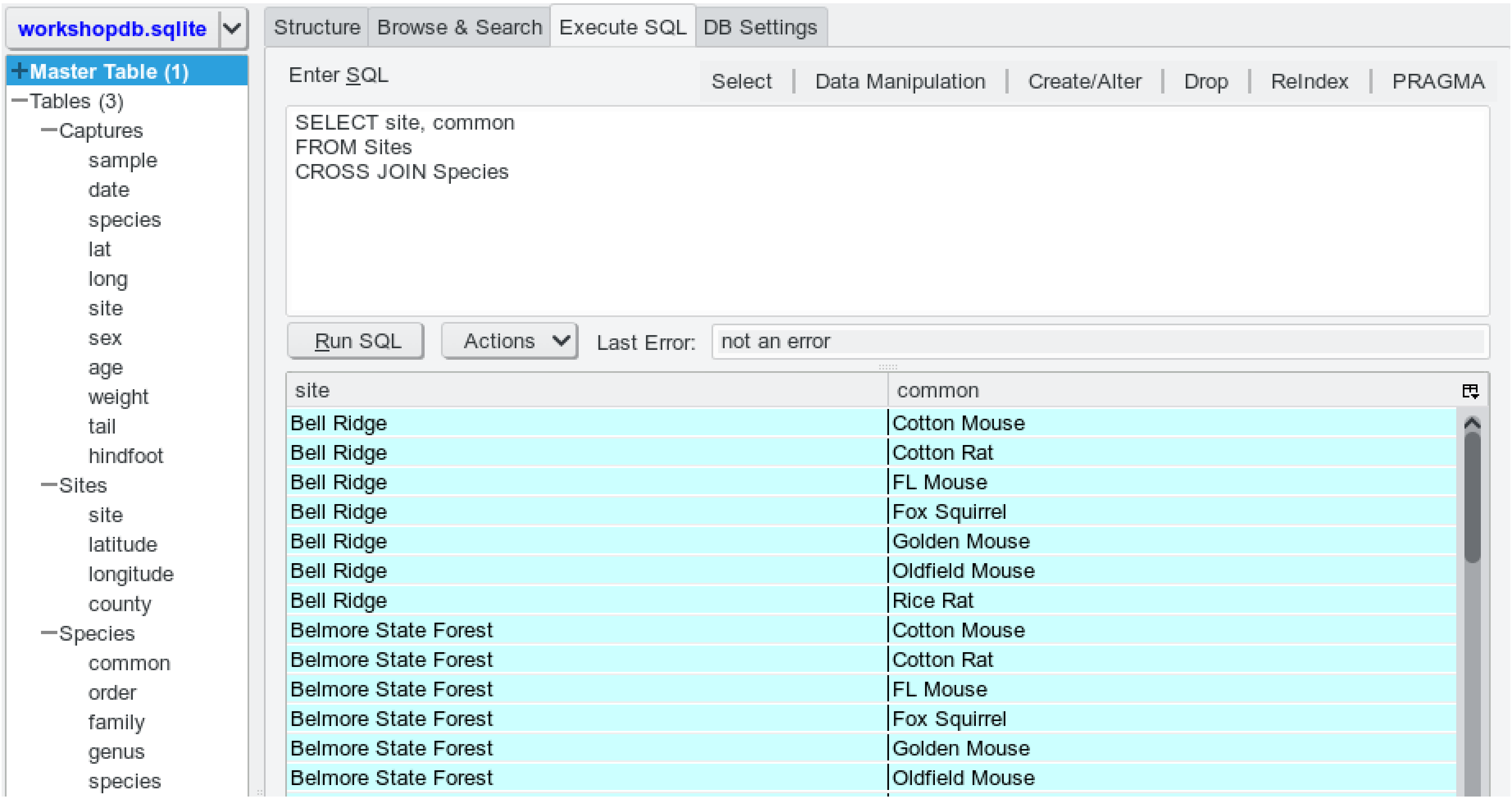
SELECT – CROSS JOIN

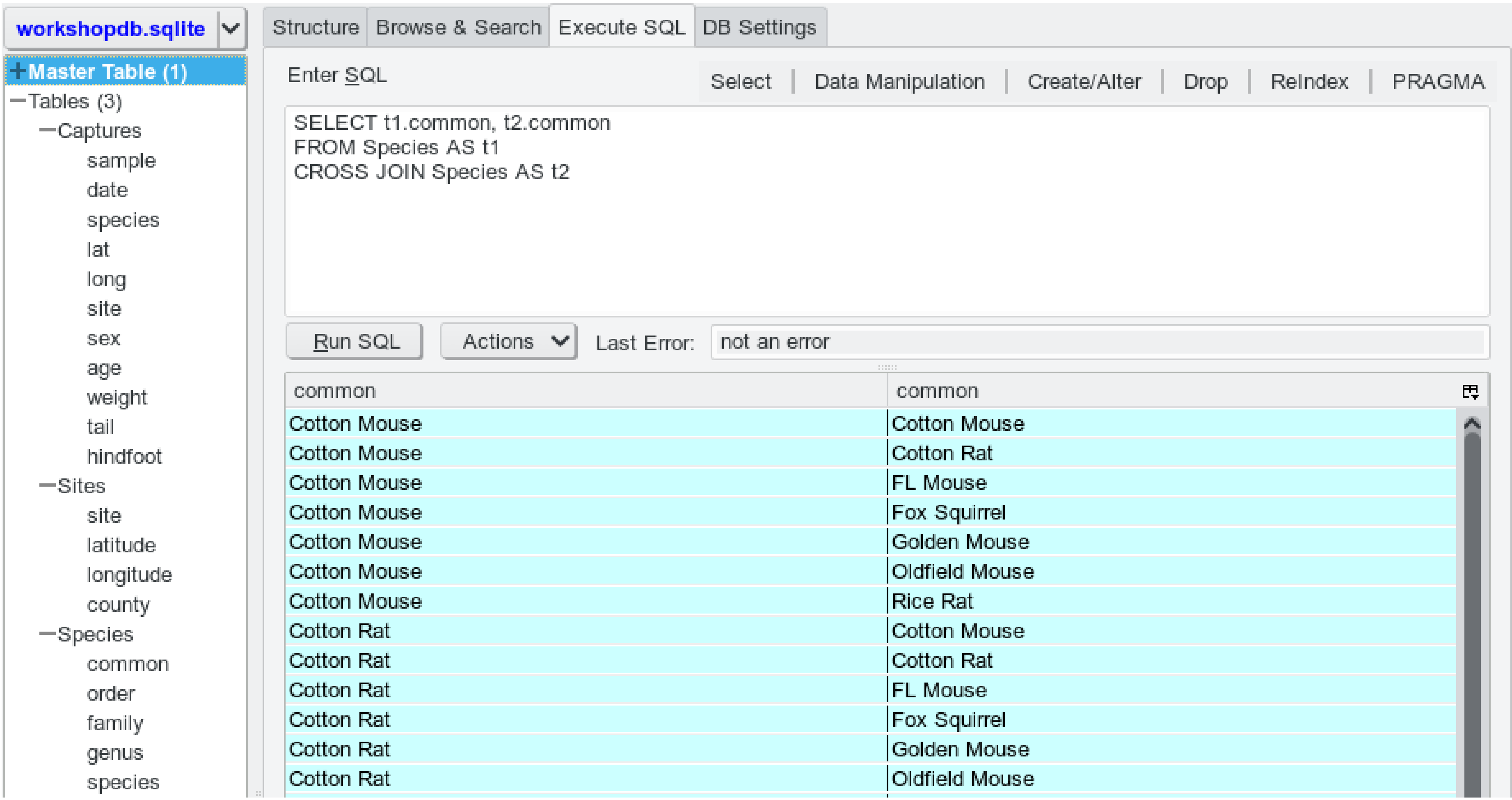
- A CROSS JOIN matches every row of one table with every row of another table
- Can generate extremely large tables if not careful

SELECT *table.column,…*

FROM *table1*

CROSS JOIN *table2*


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT site, common
FROM Sites
CROSS JOIN Species
Run SQL
Actions
Last Error:
not an error
site
common
Bell Ridge
Cotton Mouse
Bell Ridge
Cotton Rat
Bell Ridge
FL Mouse
Bell Ridge
Fox Squirrel
Bell Ridge
Golden Mouse
Bell Ridge
Oldfield Mouse
Bell Ridge
Rice Rat
Belmore State Forest
Cotton Mouse
Belmore State Forest
Cotton Rat
Belmore State Forest
FL Mouse
Belmore State Forest
Fox Squirrel
Belmore State Forest
Golden Mouse
Belmore State Forest
Oldfield Mouse


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT t1.common, t2.common
FROM Species AS t1
CROSS JOIN Species AS t2
Run SQL
Actions
Last Error:
not an error
common
common
Cotton Mouse
Cotton Mouse
Cotton Mouse
Cotton Rat
Cotton Mouse
FL Mouse
Cotton Mouse
Fox Squirrel
Cotton Mouse
Golden Mouse
Cotton Mouse
Oldfield Mouse
Cotton Mouse
Rice Rat
Cotton Rat
Cotton Mouse
Cotton Rat
Cotton Rat
Cotton Rat
FL Mouse
Cotton Rat
Fox Squirrel
Cotton Rat
Golden Mouse
Cotton Rat
Oldfield Mouse

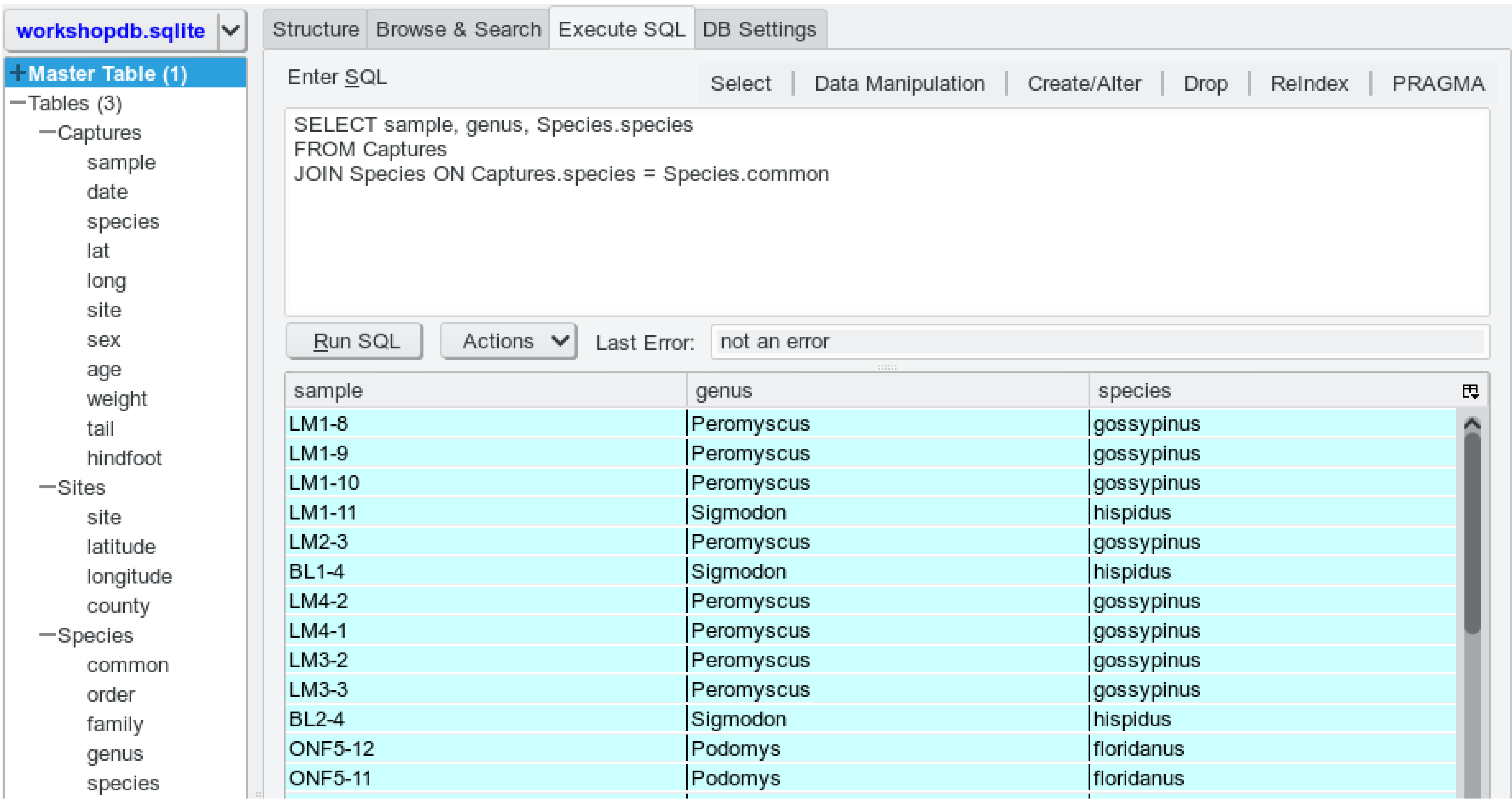
SELECT – INNER JOIN

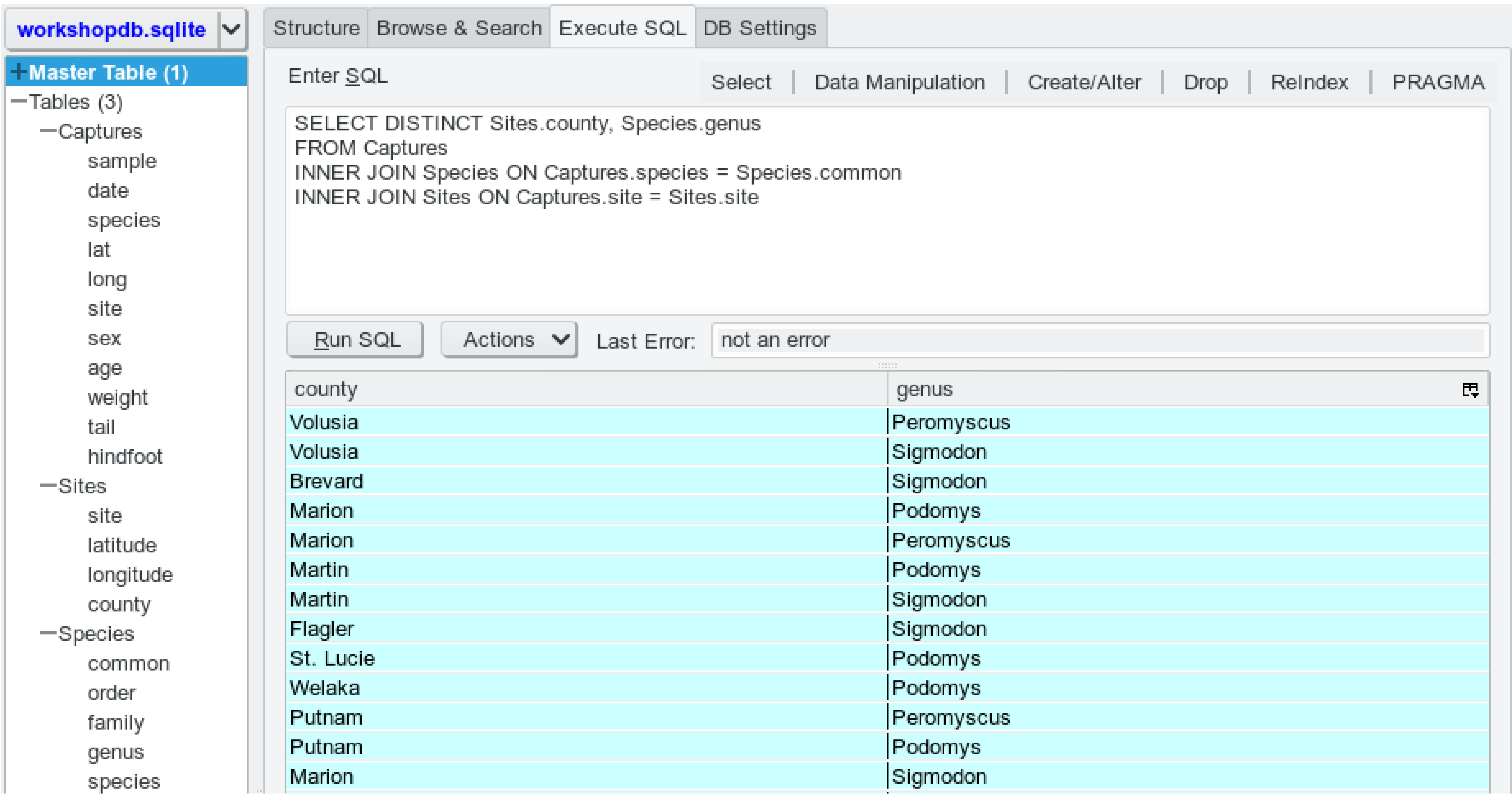
- The most common type of JOIN, an INNER JOIN matches rows together from multiple tables
- Because it is the default type of JOIN in SQLite, the INNER portion can be omitted

SELECT *table.column,…*

FROM *table1*

[INNER] JOIN *table2* ON *table1.column = table2.column*


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, genus, Species.species
FROM Captures
JOIN Species ON Captures.species = Species.common
Run SQL
Actions
Last Error:
not an error
sample
genus
species
LM1-8 Peromyscus gossypinus
LM1-9 Peromyscus gossypinus
LM1-10 Peromyscus gossypinus
LM1-11 Sigmodon hispidus
LM2-3 Peromyscus gossypinus
BL1-4 Sigmodon hispidus
LM4-2 Peromyscus gossypinus
LM4-1 Peromyscus gossypinus
LM3-2 Peromyscus gossypinus
LM3-3 Peromyscus gossypinus
BL2-4 Sigmodon hispidus
ONF5-12 Podomys floridanus
ONF5-11 Podomys floridanus


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT DISTINCT Sites.county, Species.genus
FROM Captures
INNER JOIN Species ON Captures.species = Species.common
INNER JOIN Sites ON Captures.site = Sites.site
Run SQL
Actions
Last Error:
not an error
county
genus
Volusia
Peromyscus
Volusia
Sigmodon
Brevard
Sigmodon
Marion
Podomys
Marion
Peromyscus
Martin
Podomys
Martin
Sigmodon
Flagler
Sigmodon
St. Lucie
Podomys
Welaka
Podomys
Putnam
Peromyscus
Putnam
Podomys
Marion
Sigmodon

SELECT – OUTER JOIN

- OUTER JOIN is an extension of INNER JOIN
- Basically, it includes unmatched rows as well, but pads them with null values
- Technically, there are 3 types (LEFT OUTER JOIN, RIGHT OUTER JOIN, and FULL OUTER JOIN), but SQLite only supports LEFT OUTER JOIN

SELECT *table.column,…*

FROM *table1*

LEFT OUTER JOIN *table2* ON *table1.column* = *table2.column*

WHERE

- WHERE is used to filter your data

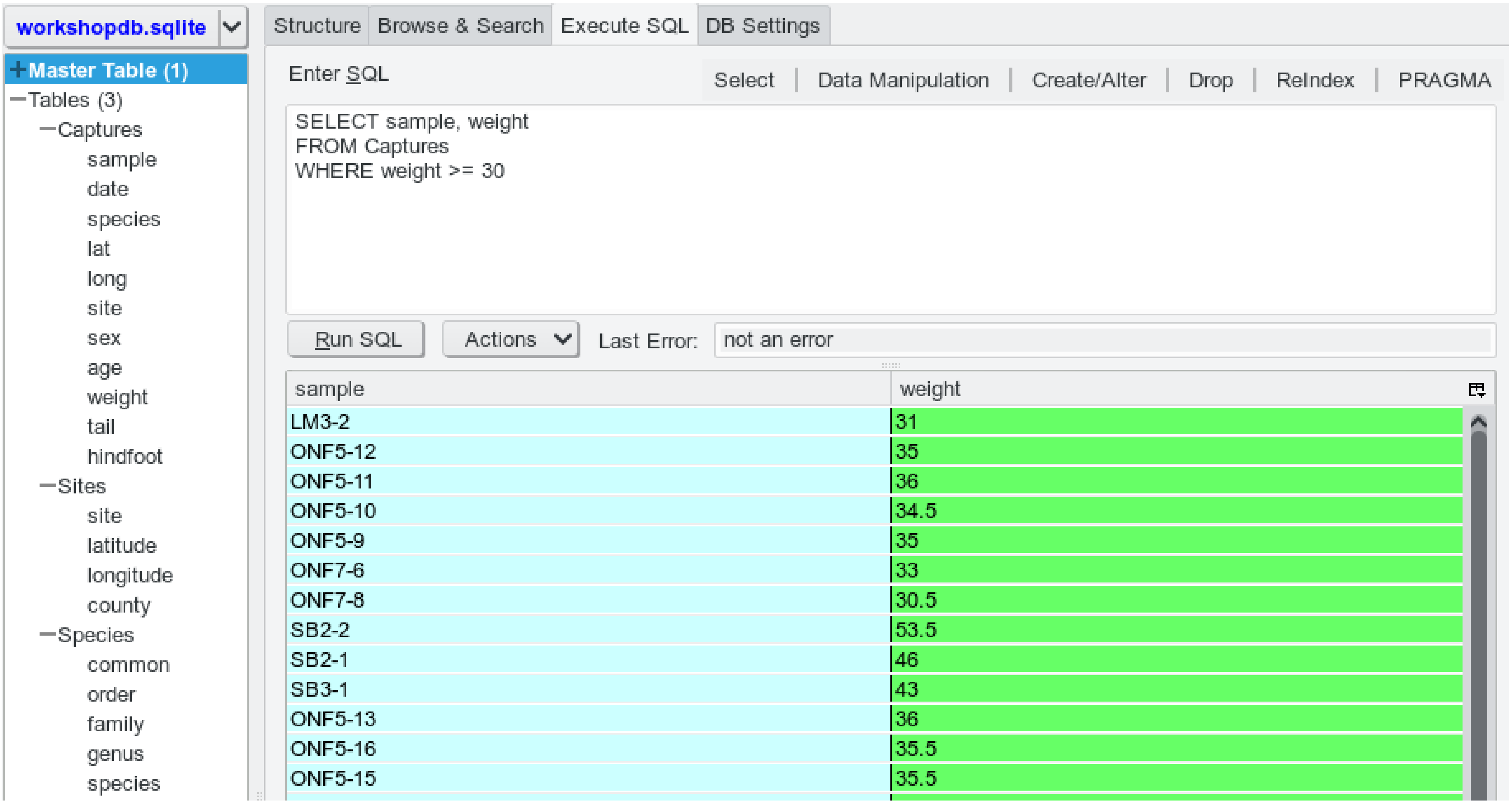
    WHERE *column operator [value]*

- *operator* can be one of the following: =, <>, >, >=, <, <=, IN, BETWEEN, LIKE, IS NULL, IS NOT NULL

WHERE – Comparators

- Comparators: =, <>, >, >=, <, <=
- Used to compare numeric values
- Basically will return false for NULL values

WHERE *column comparator value*


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, weight
FROM Captures
WHERE weight >= 30
Run SQL
Actions
Last Error:
not an error
sample
weight
LM3-2
31
ONF5-12
35
ONF5-11
36
ONF5-10
34.5
ONF5-9
35
ONF7-6
33
ONF7-8
30.5
SB2-2
53.5
SB2-1
46
SB3-1
43
ONF5-13
36
ONF5-16
35.5
ONF5-15
35.5

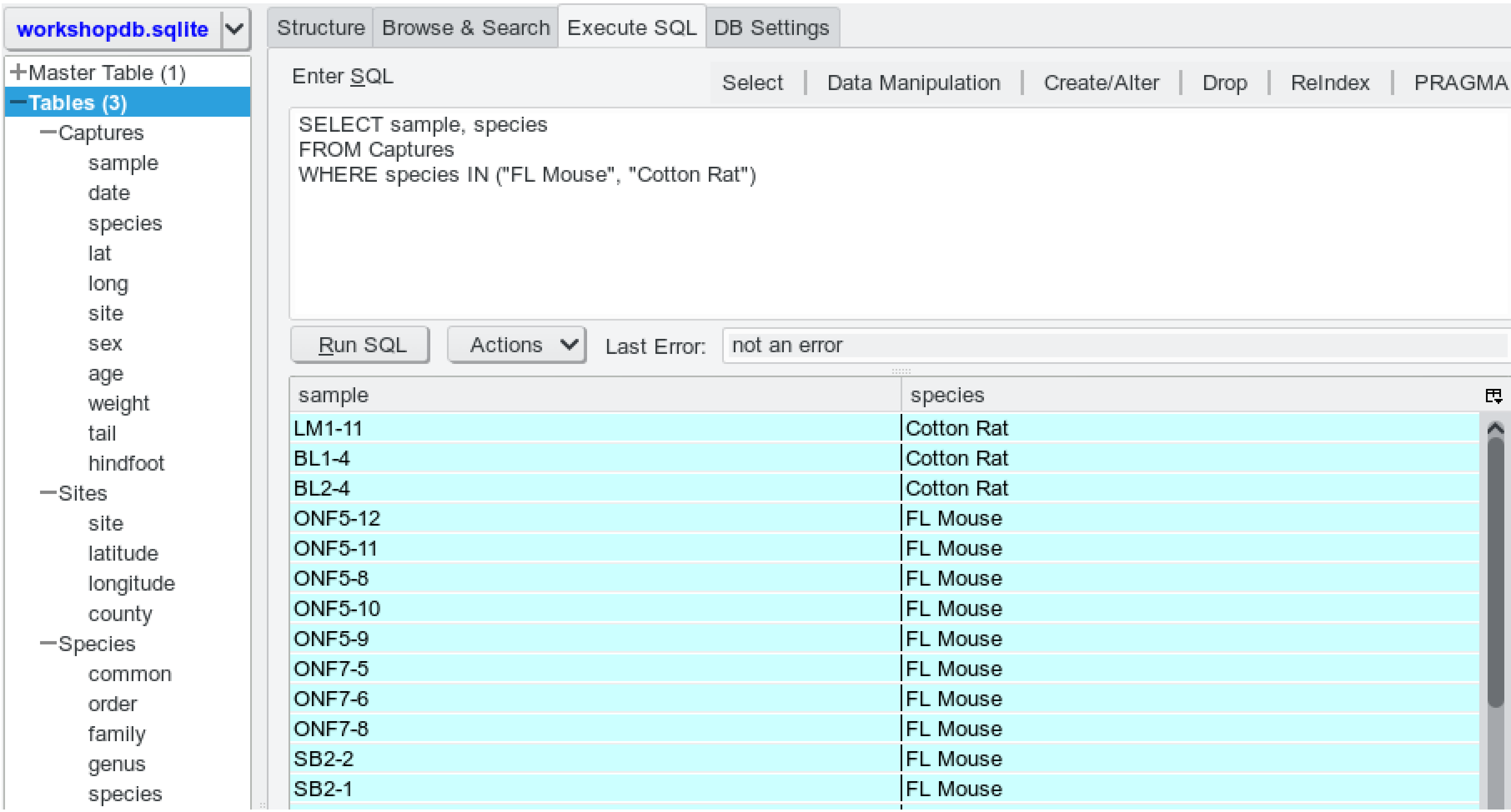
WHERE – IN, BETWEEN

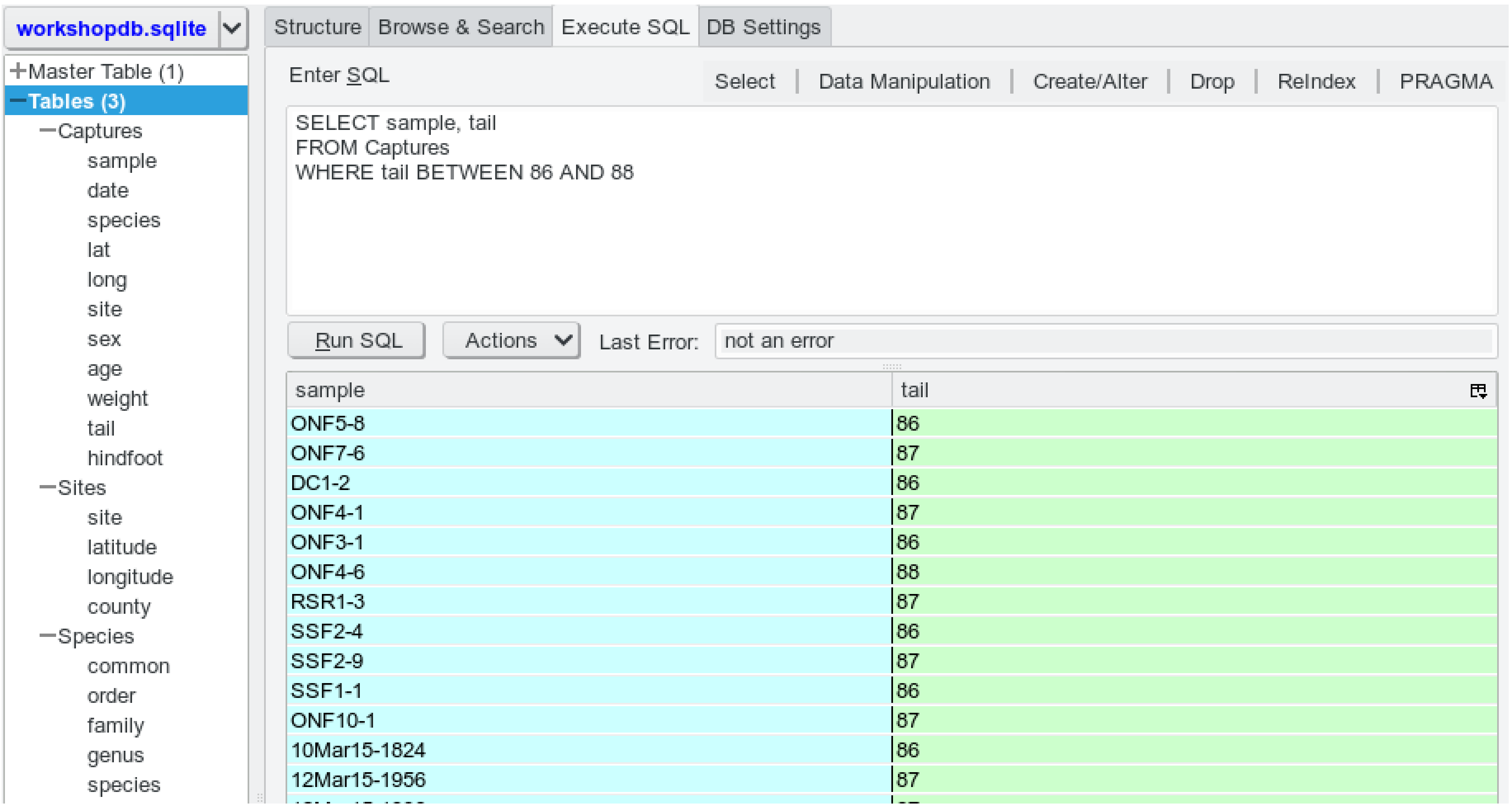
- IN is used to find values in a set (text or numeric)

  WHERE *column* IN (*value1, value2, …*)

- BETWEEN is used to find values in a range

  WHERE *column* BETWEEN *value1* AND *value2*


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, species
FROM Captures
WHERE species IN ("FL Mouse", "Cotton Rat")
Run SQL
Actions
Last Error:
not an error
sample
species
LM1-11
Cotton Rat
BL1-4
Cotton Rat
BL2-4
Cotton Rat
ONF5-12
FL Mouse
ONF5-11
FL Mouse
ONF5-8
FL Mouse
ONF5-10
FL Mouse
ONF5-9
FL Mouse
ONF7-5
FL Mouse
ONF7-6
FL Mouse
ONF7-8
FL Mouse
SB2-2
FL Mouse
SB2-1
FL Mouse


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, tail
FROM Captures
WHERE tail BETWEEN 86 AND 88
Run SQL
Actions
Last Error:
not an error
sample
tail
ONF5-8
86
ONF7-6
87
DC1-2
86
ONF4-1
87
ONF3-1
86
ONF4-6
88
RSR1-3
87
SSF2-4
86
SSF2-9
87
SSF1-1
86
ONF10-1
87
10Mar15-1824
86
12Mar15-1956
87

WHERE – LIKE

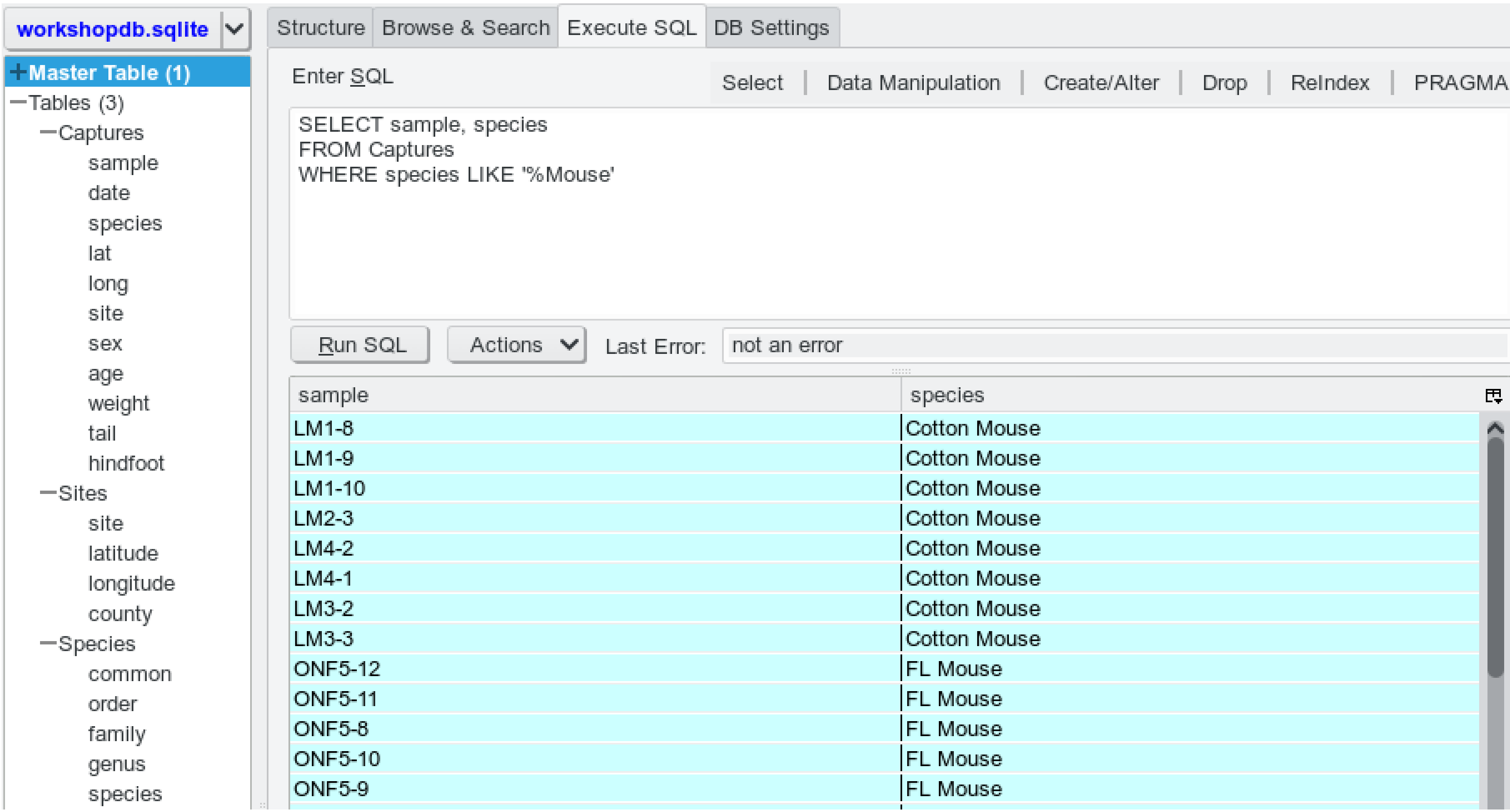
- LIKE is used to search for patterns of text
- Is not case sensitive
- Wildcards essentially allow you to look for substitutes in a string
  - The percent sign (%) wildcard means 0 or more characters
  - The underscore (_) wildcard means exactly 1 character

WHERE *column* LIKE *pattern*

- The '=' operator can also be used to determine if two strings are equal, but can behave differently than LIKE in certain situations
- GLOB is a more advanced and case sensitive alternative to LIKE

Wildcard examples

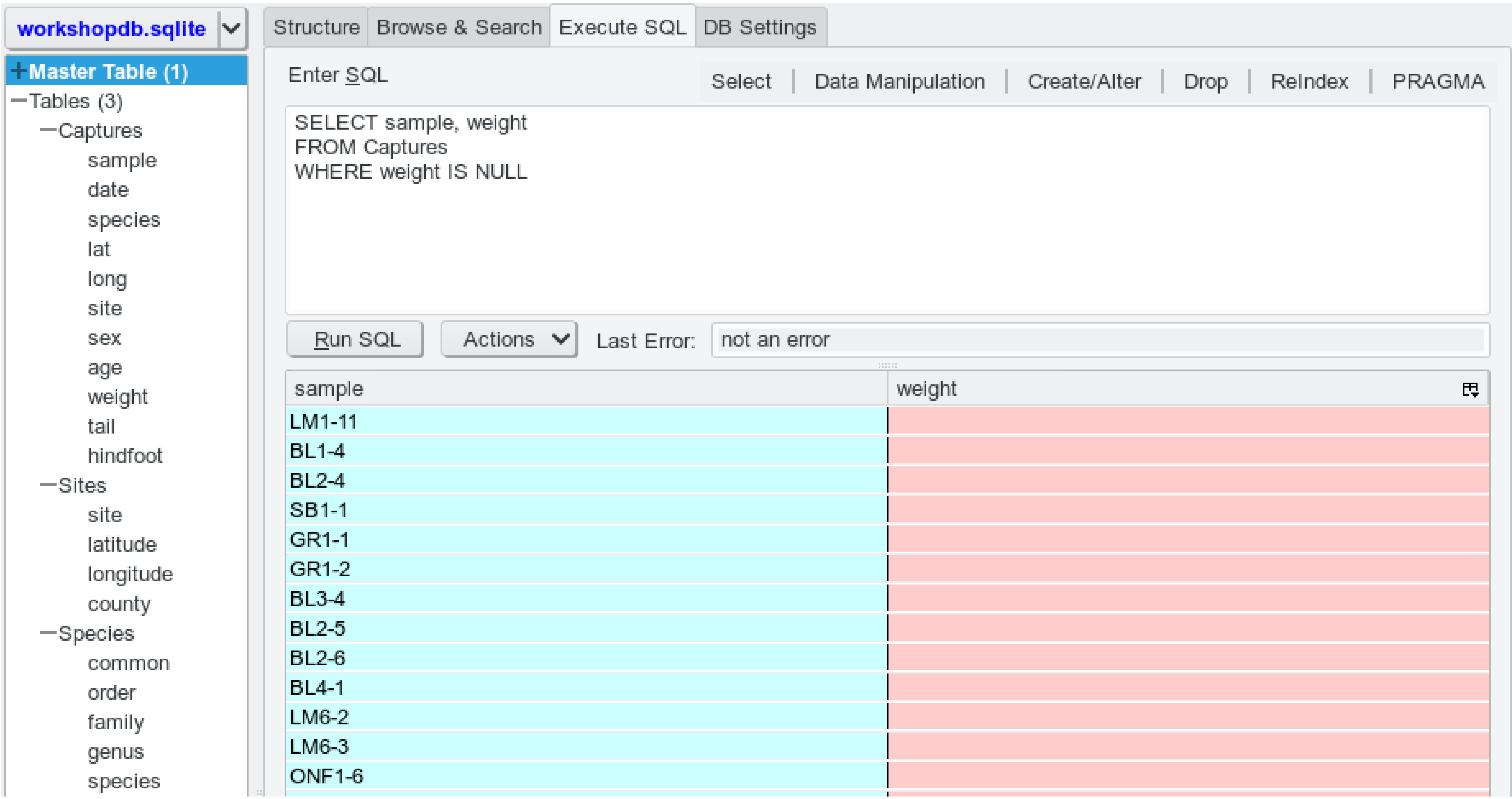
Statement	Description
LIKE 'ace%'	Finds all strings that start with 'ace'
LIKE 'ace_'	Finds all strings that start with 'ace' and are exactly 4 characters long
LIKE '%ace'	Finds all strings that end with 'ace'
LIKE '_ace'	Finds all strings that end with 'ace' and are exactly 4 characters long
LIKE 'ac%e'	Finds all strings that start with 'ac' and end with 'e'
LIKE 'ac_e'	Finds all strings that start with 'ac', end with 'e', and the 3rd character is anything
LIKE '%a_c_e%'	Finds all strings that have 5 characters in them. The 5 characters must start with an 'a', followed by any single character, followed by a 'c', followed by any single character, followed by an 'e'


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, species
FROM Captures
WHERE species LIKE '%Mouse'
Run SQL
Actions
Last Error:
not an error
sample
species
LM1-8
Cotton Mouse
LM1-9
Cotton Mouse
LM1-10
Cotton Mouse
LM2-3
Cotton Mouse
LM4-2
Cotton Mouse
LM4-1
Cotton Mouse
LM3-2
Cotton Mouse
LM3-3
Cotton Mouse
ONF5-12
FL Mouse
ONF5-11
FL Mouse
ONF5-8
FL Mouse
ONF5-10
FL Mouse
ONF5-9
FL Mouse

WHERE – IS NULL

- IS NULL is used to find rows with null values
- Necessary because null values are the absence of a value, and can't be used in comparisons

WHERE *column* IS NULL


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, weight
FROM Captures
WHERE weight IS NULL
Run SQL
Actions
Last Error:
not an error
sample
weight
LM1-11
BL1-4
BL2-4
SB1-1
GR1-1
GR1-2
BL3-4
BL2-5
BL2-6
BL4-1
LM6-2
LM6-3
ONF1-6

WHERE – NOT

- The NOT operator is used to reverse or negate the meaning of a logical operator
  - LIKE → NOT LIKE
  - IN → NOT IN
  - BETWEEN → NOT BETWEEN
  - IS NULL → IS NOT NULL

WHERE – AND, OR, ()

- The use of the AND and OR keywords allows you to combine multiple conditions
- The AND conditions are checked first, followed by the OR conditions
- The use of parentheses allows you to control the order and readability of conditions

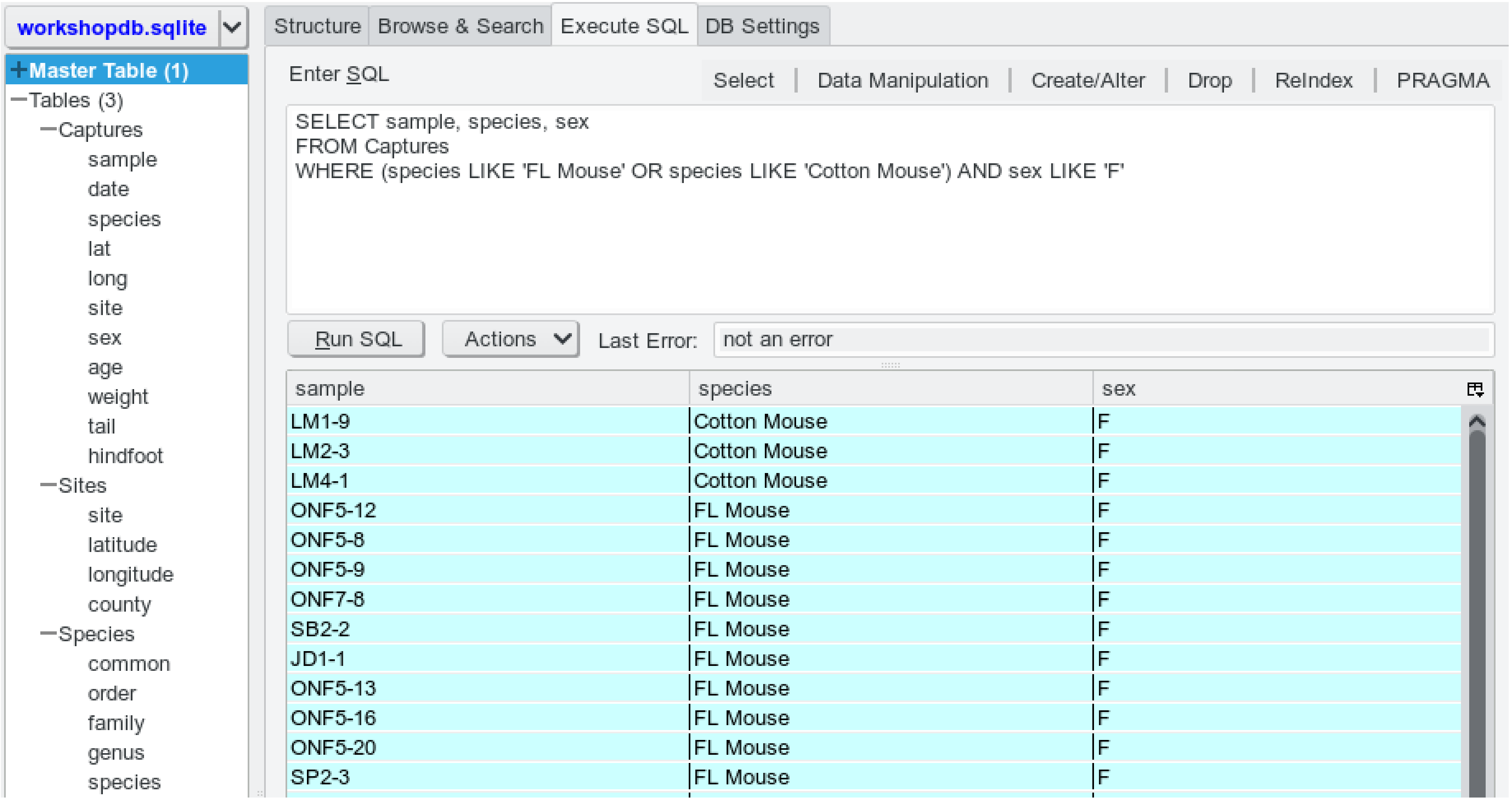
WHERE *condition1* AND *condition2* OR *condition3*

Is equivalent to:

WHERE (*condition1* AND *condition2)* OR *condition3*

And is different from:

WHERE *condition1* AND (*condition2* OR *condition3*)

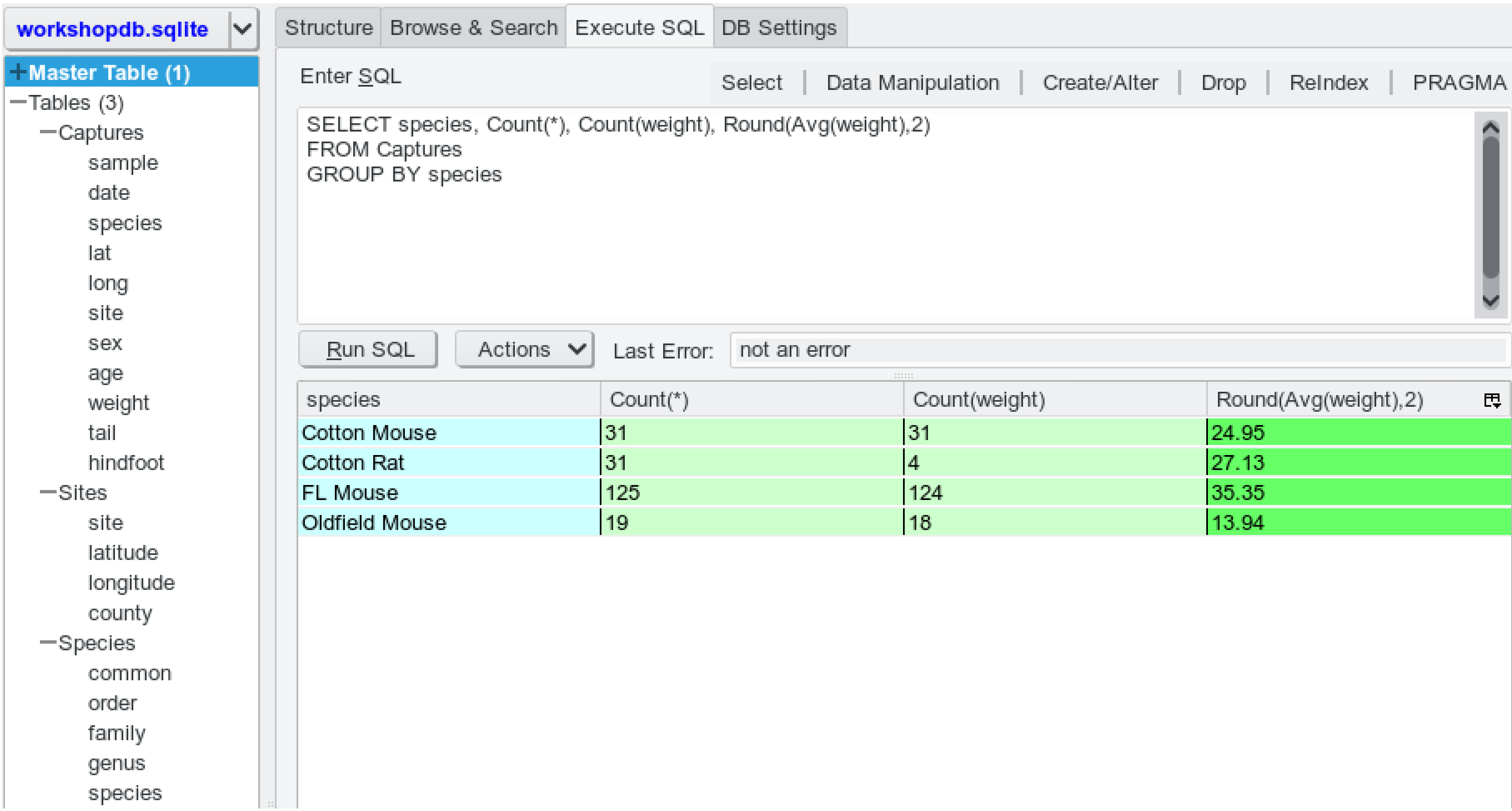

workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT sample, species, sex
FROM Captures
WHERE (species LIKE 'FL Mouse' OR species LIKE 'Cotton Mouse') AND sex LIKE 'F'
Run SQL
Actions
Last Error:
not an error
sample
species
sex
LM1-9
Cotton Mouse
F
LM2-3
Cotton Mouse
F
LM4-1
Cotton Mouse
F
ONF5-12
FL Mouse
F
ONF5-8
FL Mouse
F
ONF5-9
FL Mouse
F
ONF7-8
FL Mouse
F
SB2-2
FL Mouse
F
JD1-1
FL Mouse
F
ONF5-13
FL Mouse
F
ONF5-16
FL Mouse
F
ONF5-20
FL Mouse
F
SP2-3
FL Mouse
F

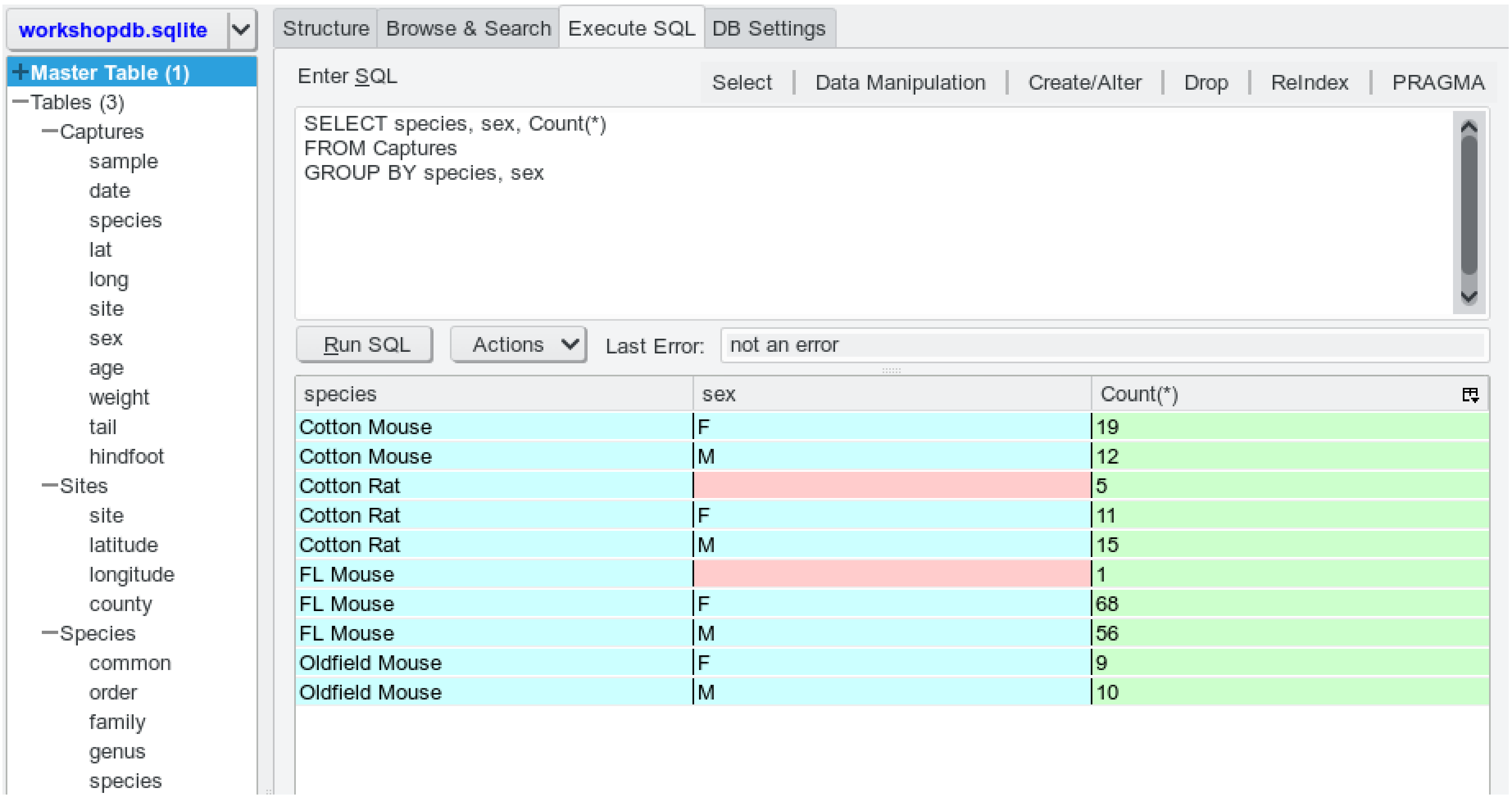
GROUP BY

- Allows you to group your data based on values from one or more columns
- Aggregate functions will be applied to each group instead of the whole table

GROUP BY *column(s)*

- *column(s)* is a comma separated list of columns that you want group your results by


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT species, Count(*), Count(weight), Round(Avg(weight),2)
FROM Captures
GROUP BY species
Run SQL
Actions
Last Error:
not an error
species
Count(*)
Count(weight)
Round(Avg(weight),2)
Cotton Mouse
31
31
24.95
Cotton Rat
31
4
27.13
FL Mouse
125
124
35.35
Oldfield Mouse
19
18
13.94

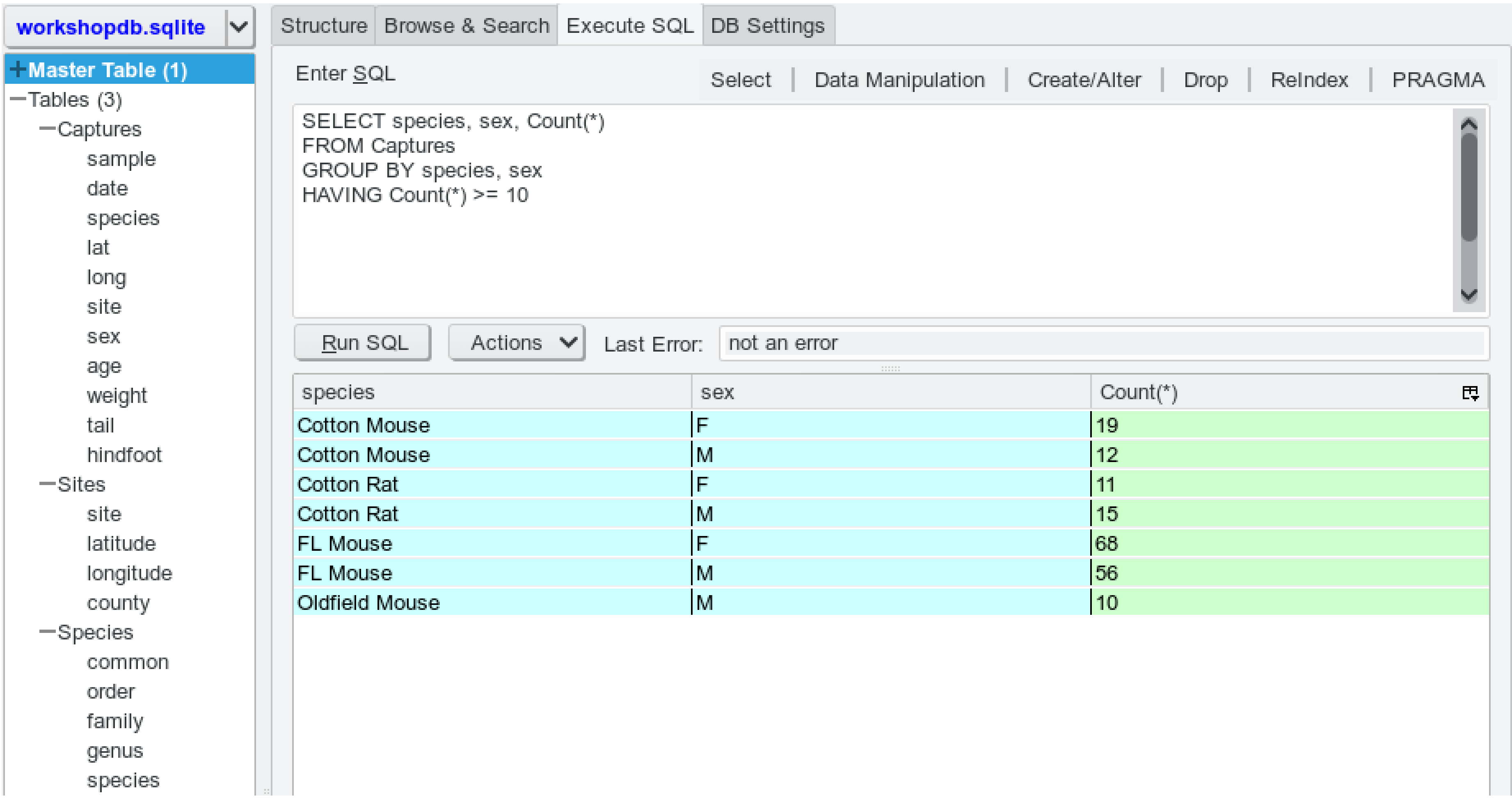

workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT species, sex, Count(*)
FROM Captures
GROUP BY species, sex
Run SQL
Actions
Last Error:
not an error
species
sex
Count(*)
Cotton Mouse
F
19
Cotton Mouse
M
12
Cotton Rat
5
Cotton Rat
F
11
Cotton Rat
M
15
FL Mouse
1
FL Mouse
F
68
FL Mouse
M
56
Oldfield Mouse
F
9
Oldfield Mouse
M
10

GROUP BY – HAVING

- WHERE is used to filter each row before it is grouped
- HAVING is used to filter your data after it has been grouped

GROUP BY *column(s)*

HAVING *condition(s)*

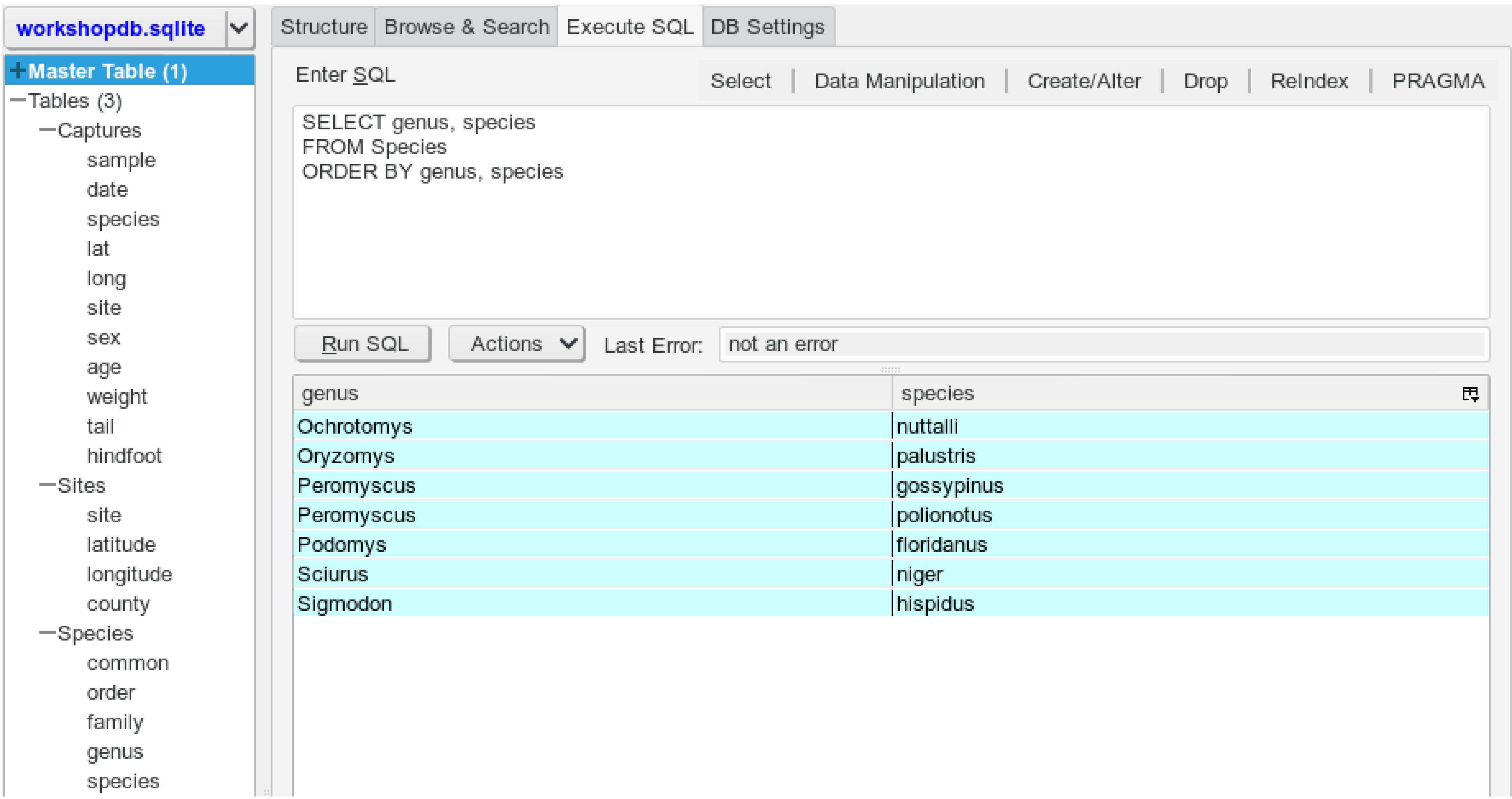

workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT species, sex, Count(*)
FROM Captures
GROUP BY species, sex
HAVING Count(*) >= 10
Run SQL
Actions
Last Error:
not an error
species
sex
Count(*)
Cotton Mouse
F
19
Cotton Mouse
M
12
Cotton Rat
F
11
Cotton Rat
M
15
FL Mouse
F
68
FL Mouse
M
56
Oldfield Mouse
M
10

ORDER BY

- Allows you to order your results
- Without it, the order of your results is undefined, or not guaranteed

ORDER BY *column(s)*

- *column(s)* is a comma separated list of the columns you want to order your results by
- The order of the columns matters


workshopdb.sqlite
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Structure
Browse & Search
Execute SQL
DB Settings
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT genus, species
FROM Species
ORDER BY genus, species
Run SQL
Actions
Last Error:
not an error
genus
species
Ochrotomys nuttalli
Oryzomys palustris
Peromyscus gossypinus
Peromyscus polionotus
Podomys floridanus
Sciurus niger
Sigmodon hispidus

ORDER BY – ASC, DESC

- By default, each column in ORDER BY is sorted in ascending order
- The sort order can be specified by including the ASC or DESC keyword after each column name

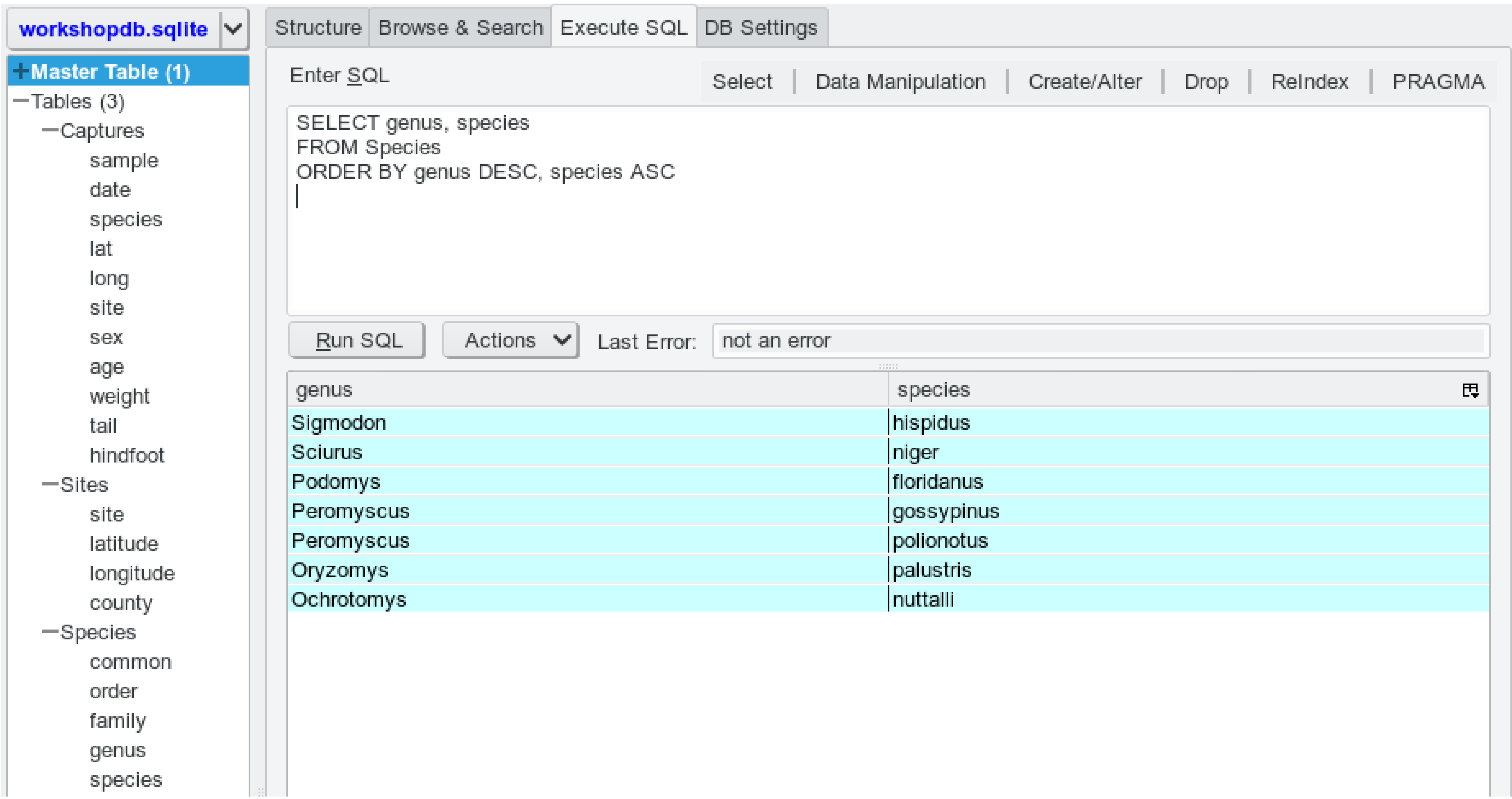
ORDER BY *column*

Is the same as:

ORDER BY *column* ASC

Is the reverse of:

ORDER BY *column* DESC


workshopdb.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (3)
Captures
sample
date
species
lat
long
site
sex
age
weight
tail
hindfoot
Sites
site
latitude
longitude
county
Species
common
order
family
genus
species
Enter SQL
Select
Data Manipulation
Create/Alter
Drop
ReIndex
PRAGMA
SELECT genus, species
FROM Species
ORDER BY genus DESC, species ASC
Run SQL
Actions
Last Error:
not an error
genus
species
Sigmodon
hispidus
Sciurus
niger
Podomys
floridanus
Peromyscus
gossypinus
Peromyscus
polionotus
Oryzomys
palustris
Ochrotomys
nuttalli

Integrating Databases With Other Software

Database Integration

- One of the main features of relational databases is their ability to be easily integrated with other software
- This facilitates automation of workflows, which can save significant amounts of time
- Many programming languages support it, including R and Python
- GIS software (ArcGIS, QuantumGIS, etc) supports it

R Integration

- Using relational databases with R is actually very simple
- The steps we use for SQLite in R is almost identical for other implementations like MySQL, PostgreSQL, etc
- Basic steps:
  - Load the database library
  - Connect to the database
  - Submit SQL command to database
  - Something happens (results)

RSQLite

- RSQLite is an implementation of the DBI library for SQLite
- Useful commands:
  - dbConnect()
  - dbListTables()
  - dbListFields()
  - dbSendQuery()
  - DbGetQuery()
  - dbReadTable()
  - dbWriteTable()
- These apply to other implementations (MySQL, PostgreSQL, etc) as well

RSQLite – dbConnect()

- dbConnect() is used to connect to databases

  connection = dbConnect(*driver*, *dbname*)

- *driver* is used to specify the type of database. For SQLite, *driver* = RSQLite::SQLite()
- *dbname* is a string used to specify the location of the database
- *connection* is a variable to store information about the connection

RSQLite – dbListTables(), dbListFields()

- dbListTables() lists the tables and views in a database

  tables = dbListTables(*connection*)

- dbListFields() lists the fields in a particular table

  fields = dbListFields(*connection, table*)

- *connection* is used to specify the database
- *tables* is a variable to store a vector of the table names in the database
- *table* is a string containing the name of a table in the database
- *fields* is a variable to store a vector of the field names in a table

RSQLite – dbSendQuery(), dbGetQuery()

- dbSendQuery() is used to send SQL commands to the database
- Does not directly return results
- Useful for making changes to the database (e.g, creating/deleting tables, adding data, etc)

  dbSendQuery(*connection, query*)

- dbGetQuery() does the same thing, except it also returns results (if any)

  results = dbGetQuery(*connection, query*)

- *query* is a string containing an sql statement
- *results* is a variable to store the results of the query. Record data will be returned as a data frame

RSQLite – dbReadTable(), dbWriteTable()

- dbReadTable() is a shortcut for loading all the data from a table into a data frame
- Equivalent to running “SELECT * FROM *table*” in dbGetQuery()

  results = dbReadTable(*connection*, *table*)

- dbWriteTable() allows you to write a data frame to a new or existing table in the database

  dbWriteTable(*connection*, *table*, *dataframe*)

Simple Example

```r

library(RSQLite)


# The path to your database will probably be different

con <- dbConnect(RSQLite::SQLite(), dbname="/Database/workshopdb.sqlite")


# We can load data from a table into a data frame
# Lets get all my Florida mouse data
# Note that you have to be careful about mixing quotes when your query involves strings

data <- dbGetQuery(con, "SELECT * FROM Captures WHERE species LIKE 'FL Mouse'")


# Personally, I prefer storing my queries as strings

query <- "SELECT *
          FROM Captures
          WHERE species LIKE 'FL Mouse'"

data <- dbGetQuery(con, query)


```