Introductory Tutorial to Databases and sql / PostgreSQL

Enquist Lab Tutorial by Brad Boyle & Brian Enquist

INTRODUCTION

There are two increasing trends in biological data. First, Biological data are growing faster than exponentially. Further, increasingly, pressing questions in biology, ecology, evolutionary biology demand synthesis and integration across different data sources, resource. Second, it is increasingly possible to not only generate and download a dataset that will 'crash' excel but also crash R.

Many of you who work with "big" data, have probably begun to discover the limitations of spreadsheets or even R for manipulating large or complex data sets. While desktop applications such as Micorsoft Access provide a glimpse of the power of relational databases, they suffer from the usual limitations of point-and-click graphical user interfaces (GUIs) and proprietary software (memory caps, poor performance, $$$, etc.).

For really large datasets, you need the power and flexibility of large, open-source databases such as MySQL (http://www.mysql.com/) and PostgreSQL (http://www.postgresql.org/). It is increasingly important to become aware not only of the basics of databases but also the theory of building databases.

Some of you already interact with these databases using various GUIs, in particular the browser-based phpMyAdmin (http://www.phpmyadmin.net/home_page/index.php). Which is great! But for this meeting, I would like to show you some of the advantages of rolling up your sleeves, getting under the hood, and learning command-line SQL.

Background

SQL (Select Query Language) is the universal language used by all nearly relational databases. Why learn SQL? While many dialects of SQL exist, their syntax and commands are nearly identical; by learning one variant you can quickly learn them all. Furthermore, writing SQL yourself lets you extract exactly what you need from your data; no GUI is perfect, and inevitably there comes a time when you need to tweak the SQL yourself. And there are all the usual advantages of scripting: you get a reusable data-extraction and manipulation pipeline that you can run again and again on the same data, tweak to improve performance or ask new questions, or adapt to completely new projects. Ever tried to reconstruct some data set you built using mouse clicks and spreadsheets? Frustrating no?

This tutorial emphasizes the version of SQL used by the one of the most popular *free* open source database, MySQL. This tutorial cannot cover all aspects – however the introduction is intended to help you get started and keep going on your own.

RELATIONAL DATABASES

SQL is used to enter data into and extract information from relational databases. To use SQL properly, you'll need to learn a bit about relational databases.

Database: an organized body of related information

Relational database: a database organized as tables having formally-described relationships

- Enables data to be accessed or reassembled in many different ways without having to

  reorganize the tables.

- Based on relational theory (E.F. Codd, 1970).

Advantages of relational databases

- Minimize error
  - due to data entry
  - due to data manipulation

- Minimize memory use
- Simpler long term maintenance
- Greater portability to other applications (data integrity lies in structure of database)

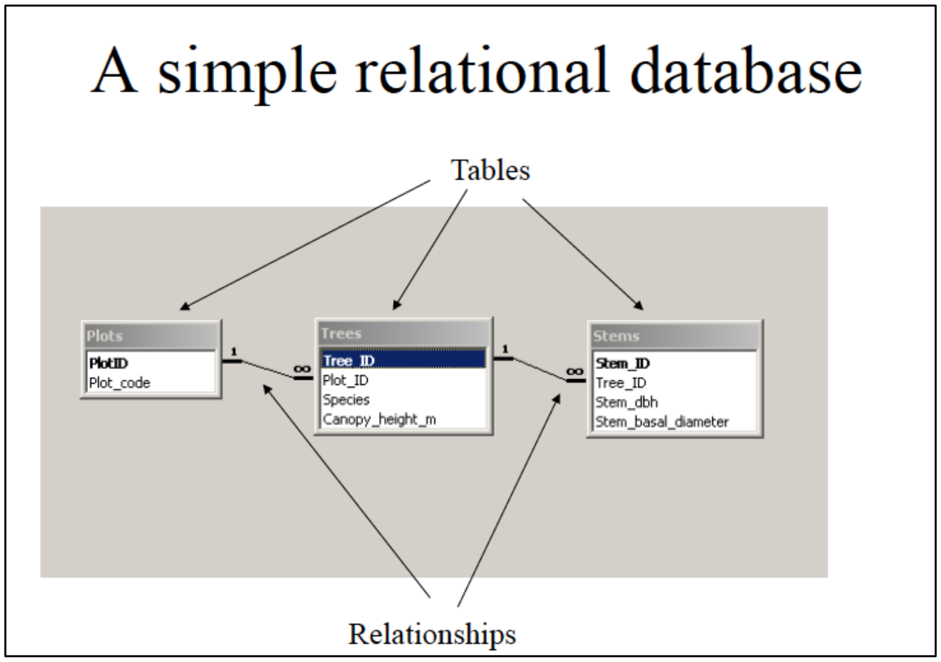
When to use relational database?

- • Large datasets (> 100,000 records)
- • Use over long periods of time or for multiple studies
- • Compiled over time, will be editing/adding records on on-going basis
- • Many people working on same dataset, esp. those doing data entry not the same as those

  doing analysis

- • Low error tolerance (think: business, banking)

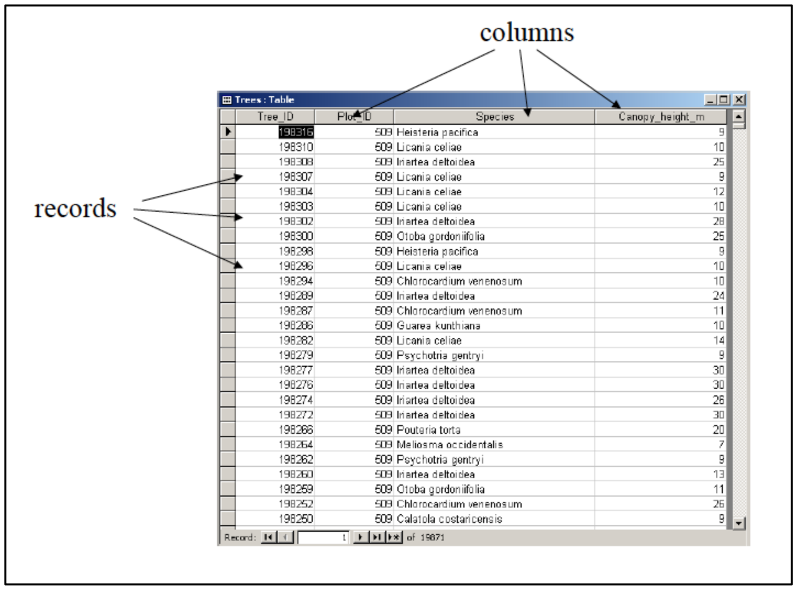
Anatomy of a database



Tables, rows, columns

- A table contains replicate observations about entities
- Observations are in rows called records (or tuples)
- Entities are described in columns (or fields)
- A table is not a spreadsheet!

A database table:



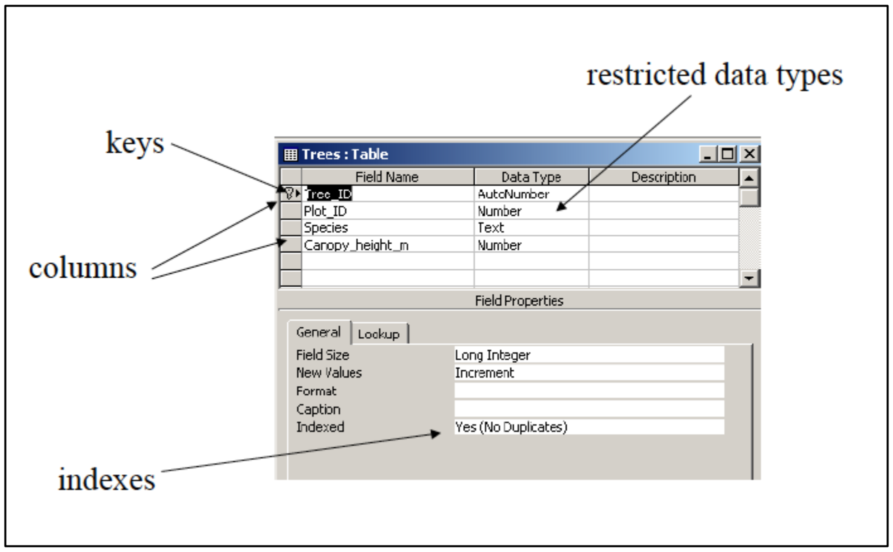
Tree_ID	Plot_ID	Species	Canopy_height_m
198316	509	Heisteria pacifica	9
198310	509	Licania celiae	10
198308	509	Iriartea deltoidea	25
198307	509	Licania celiae	9
198304	509	Licania celiae	12
198303	509	Licania celiae	10
198302	509	Iriartea deltoidea	28
198300	509	Otoba gordoniifolia	25
198298	509	Heisteria pacifica	9
198296	509	Licania celiae	10
198294	509	Chlorocardium venenosum	10
198289	509	Iriartea deltoidea	24
198287	509	Chlorocardium venenosum	11
198286	509	Guarea kunthiana	10
198282	509	Licania celiae	14
198279	509	Psychotria gentryi	9
198277	509	Iriartea deltoidea	30
198276	509	Iriartea deltoidea	30
198274	509	Iriartea deltoidea	26
198272	509	Iriartea deltoidea	30
198266	509	Pouteria torta	20
198264	509	Meliosma occidentalis	7
198262	509	Psychotria gentryi	9
198260	509	Iriartea deltoidea	13
198259	509	Otoba gordoniifolia	11
198252	509	Chlorocardium venenosum	26
198250	509	Calatola costaricensis	9

Record: 1 of 19871

Data types and indexes

- Columns have defined data types
- Columns can be indexed to speed up joining tables and searching for specific values

Screenshoot of a table definition from an MS Access database, showing column names, data types and indices:



Field Name	Data Type	Description
Tree_ID	AutoNumber	
Plot_ID	Number	
Species	Text	
Canopy_height_m	Number	

Field Properties

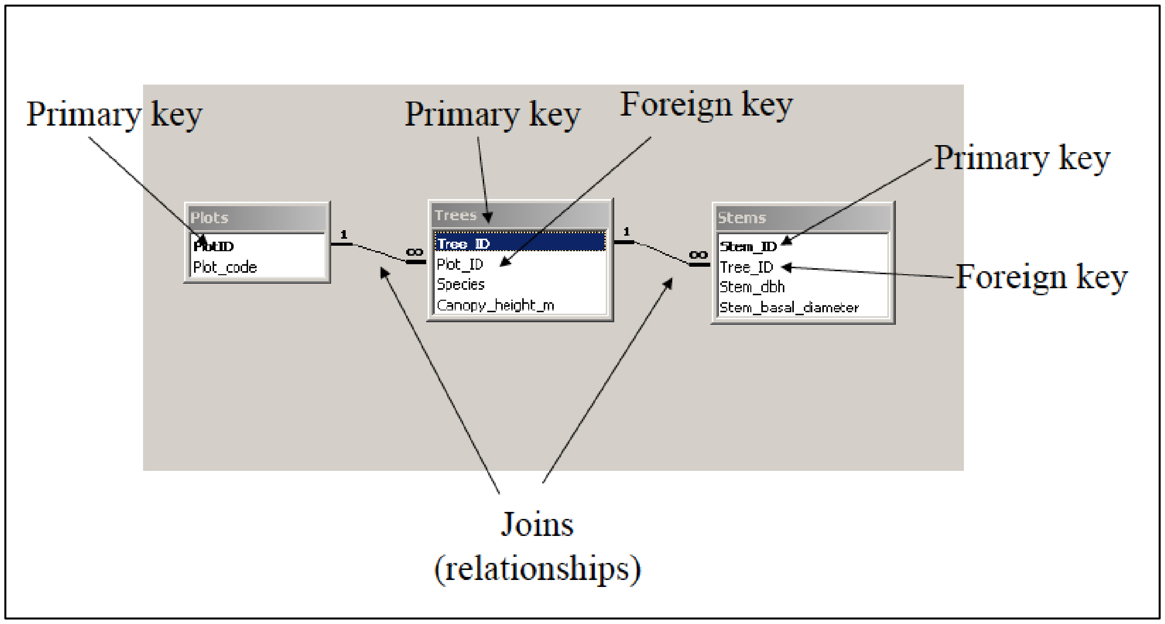
General | Lookup

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (No Duplicates)

Joins and keys

- Tables can be related (joined) to one another by means of special columns called keys
- A primary key is a special column whose values uniquely define each record in a table
- A foreign key is that same value repeated in another table, used to link back to the primary table.

Some examples of joins and keys:



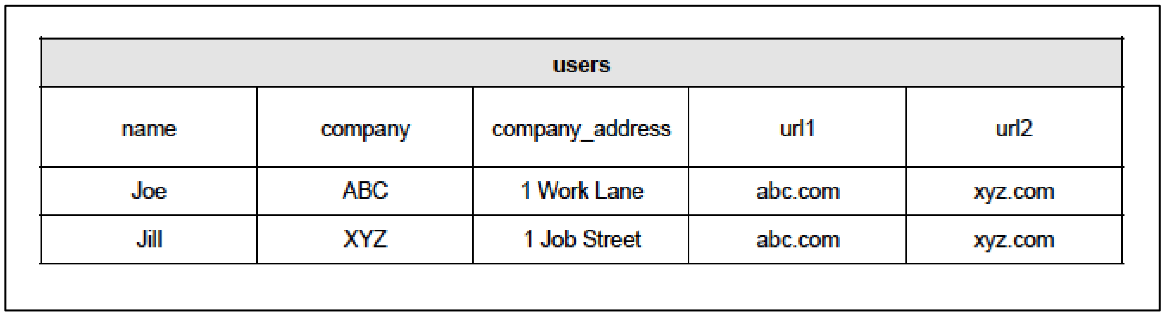
Normalization

- Relational databases are "normalized" to various degrees
- Normalization is the process of decomposing complex datasets into simple, separate, indivisible entities
- Operationally, normalization usually involves breaking down a single spreadsheet-like table into many tables, and relating those tables to one another by mean of primary key-foreign key joins.

Example of the normalization process

(From: Wise, B. 2003. Database Normalization And Design Technique. PHP Builder. http://www.phpbuilder.com/columns/barry20000731.php3)

A simple denormalized spreadsheet:

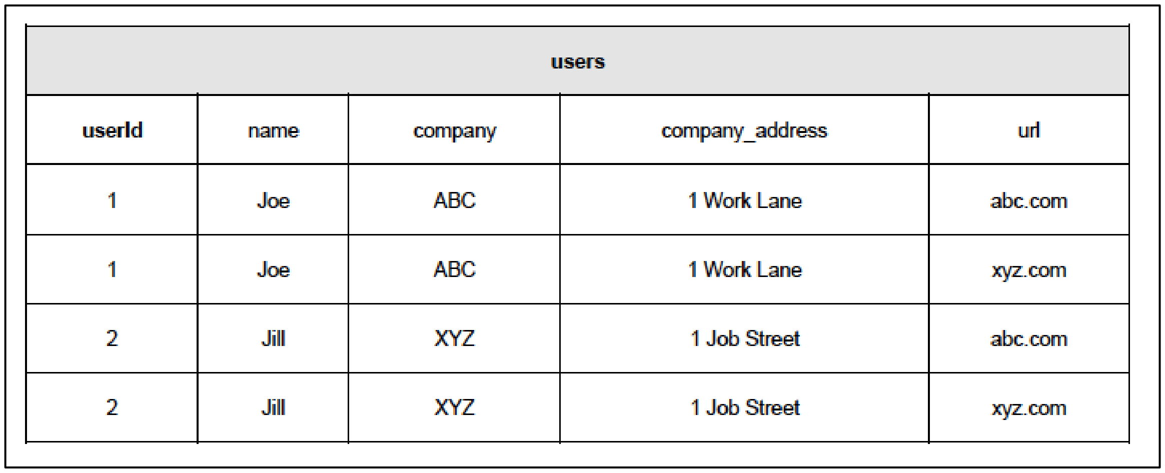


users				
name	company	company_address	url1	url2
Joe	ABC	1 Work Lane	abc.com	xyz.com
Jill	XYZ	1 Job Street	abc.com	xyz.com

First Normal Form:

- Eliminate repeating groups.
- Combine columns referring to same entity into single column
- Identify each set of related data with a primary key.

The same spreadsheet in first normal form:

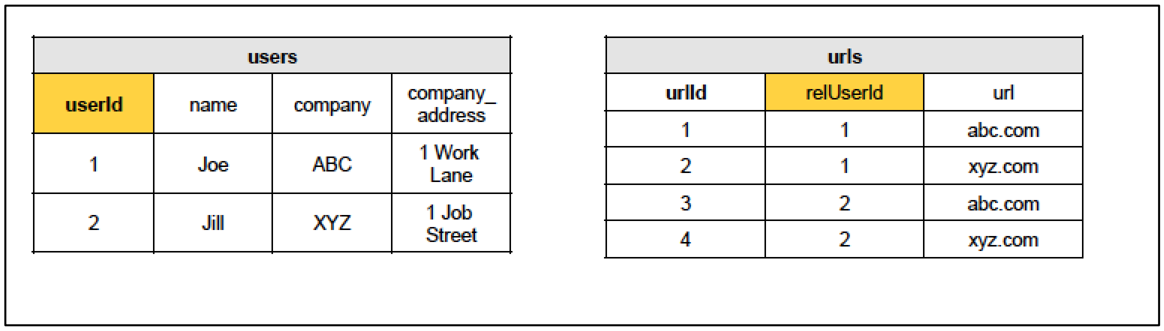


users				
userId	name	company	company_address	url
1	Joe	ABC	1 Work Lane	abc.com
1	Joe	ABC	1 Work Lane	xyz.com
2	Jill	XYZ	1 Job Street	abc.com
2	Jill	XYZ	1 Job Street	xyz.com

Second Normal Form:

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.

Now in second normal form (primary keys in yellow):



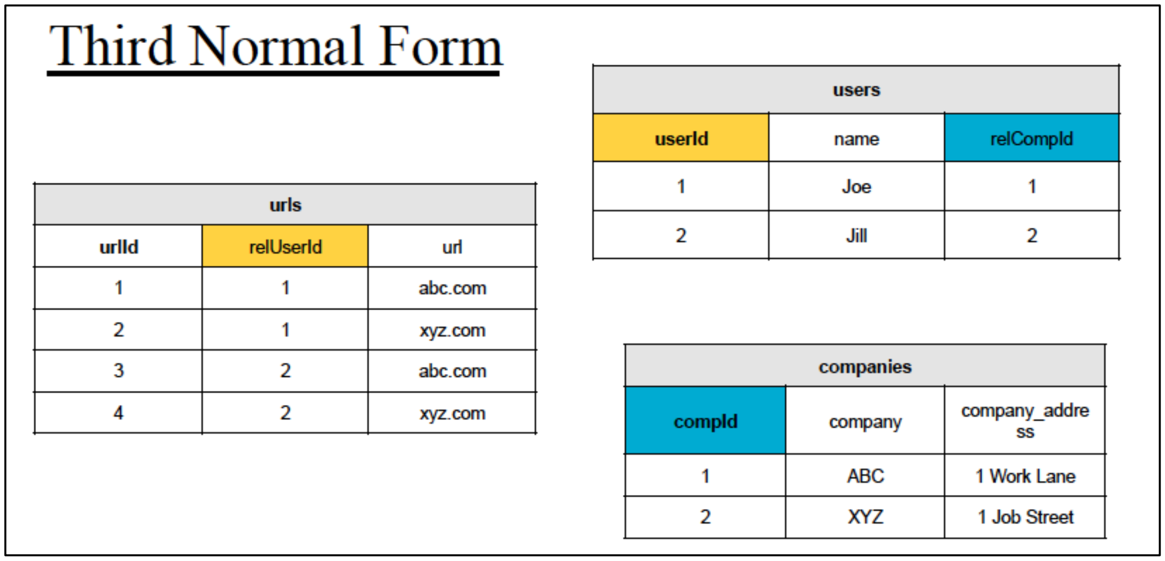
users			
userId	name	company	company_address
1	Joe	ABC	1 Work Lane
2	Jill	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

Third Normal Form

- Eliminate fields that do not depend on the key.

Now in third normal form (primary keys yellow, foreign keys blue):



Third Normal Form

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

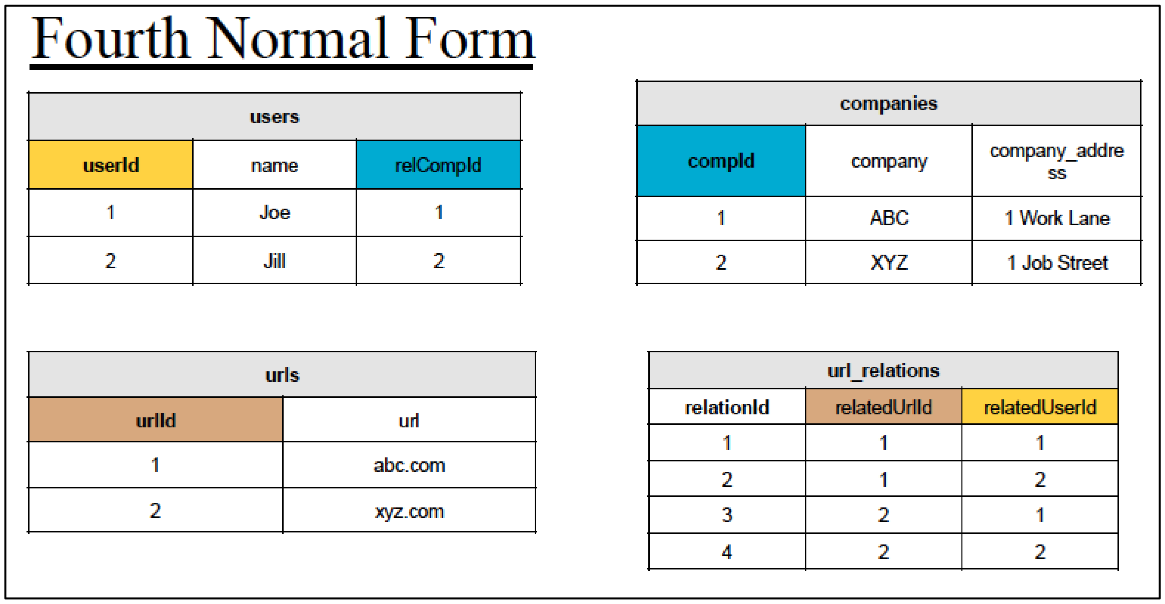
users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

Fourth Normal Form

- Move independent entities to separate tables

Finally, in fourth normal form:



Fourth Normal Form

users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls	
urlId	url
1	abc.com
2	xyz.com

url_relations		
relationId	relatedUrlId	relatedUserId
1	1	1
2	1	2
3	2	1
4	2	2

In the above example, note that the table url_relations links urls to users; that's why it has two foreign keys. Because one user can have more than one url, and one url can be shared by more than one person, the relationship between users and urls is many-to-many. Direct many-to-many relationships are not allowed under strict normalization; they must be decomposed into two one-to-many relationships by means of a linking table (in this case, url_relations).

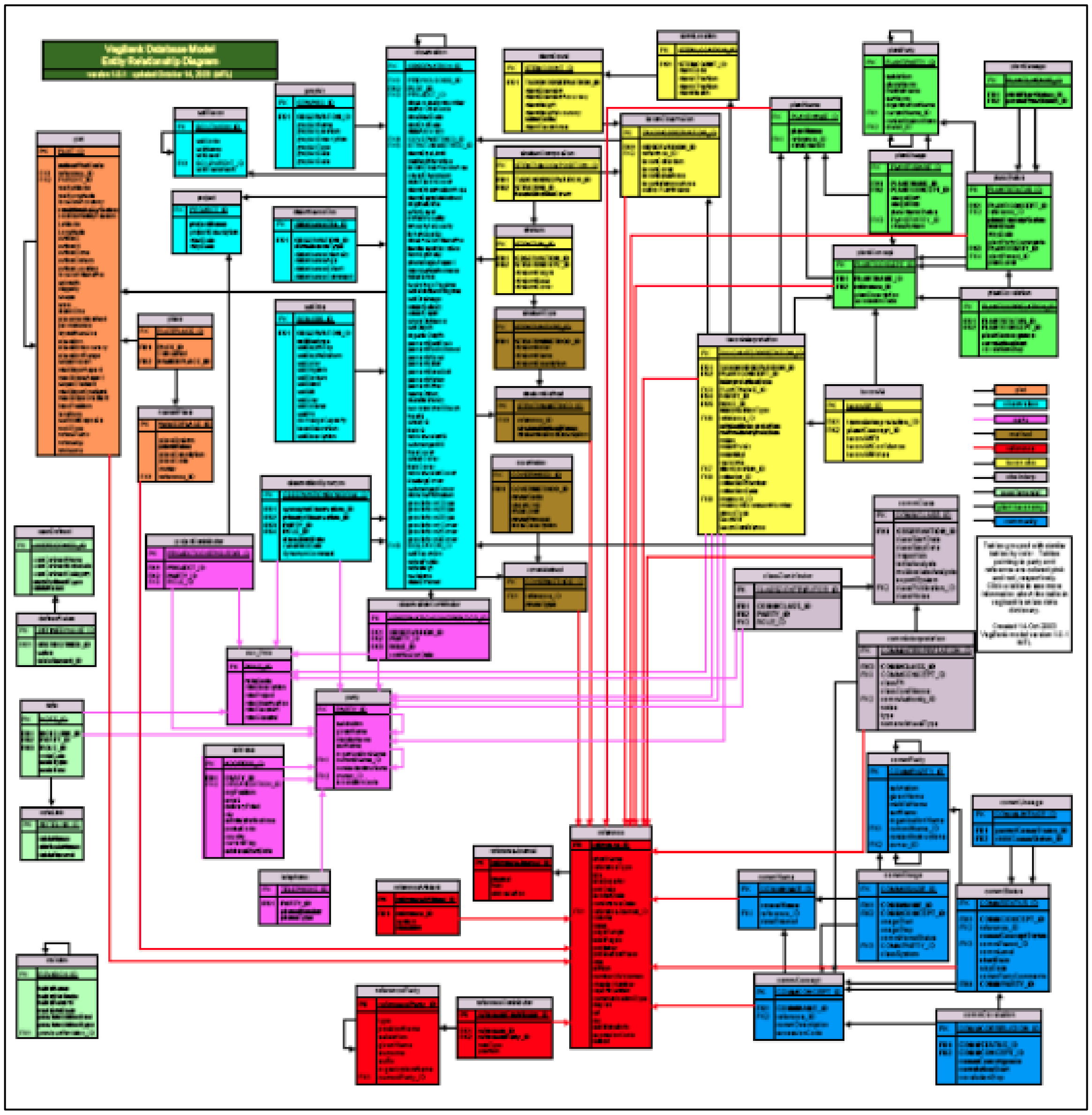
Normalization vs. denomalization

Why normalize?

- Preserve data integrity, prevent anomalies (think: banks generally used highly normalized databases):
  - Database structure itself enforces rules regarding data type, relationships and content
- Simpler to enter and update data:
  - Easier to spot and correct anomalies in data because defining attributes associated with particular entities live in only one place. Some examples:
    - The name, address, email of a person is entered only once, in one table (say, 'users') Elsewhere this person is identified only by a numeric foreign key (say, 'user_id'). Details of this person are not repeated in these tables, but are retrieved by joining to the table 'user'.
    - The name and author of a species appears only once, on a single record in the 'species' table. In other tables it appears only as a numeric foreign key (say, 'species_id'). Misspell the author? No problem! Change it once only in the species table.
  - BUT: harder initial setup, and requires complex interface to make accessble to other users
- Reduced memory usage (fewer NULLS and non-applicable values)
- Faster to WRITE (entry and editing of data)

→ Best for entry and maintenance of data (“Transactional databases”)

Now for the downside: strictly normalized databases can be mind-bogglingly complex! Check out this representation (an ERD or Entity Relationship Diagram) of VegBank (www.vegbank.org):



Why denormalize?

- Simplicity:
  - Simple initial setup
  - Simple to get data out of (if you know the rules)
  - BUT: Hard to get into and maintain, without messing up data
- Faster to READ

→ Best for databases used for read-only select operations (e.g., non-dynamic website) and data restructured in various ways for analysis (“Analytical databases)

Bottom line? You do not need to normalize you data to work with it using SQL. SQL doesn't care. In particular, if you are building a database for purely analytical purposes—you load it once only, then use SQL queries to extract data and run some analyses—you will probably not need to normalize at all (aothough you should at understand normalization so you know how to join tables). On the other hand, if you are building a data for data entry, you should consider applying at least some degree of normalization. If that database is very big, or will be used by many people, or for a long time, or you have zero error tolerance—normalize like crazy!

Finally, when in doubt, remember what do professional database programmers have to say about normalization:

Normalize til it hurts, denormalize til it works.

MYSQL

How to obtain and install MySQL

You don't necessarily need to install MySQL locally on your own machine if you have access to MySQL on a server. Still want your own installation? Here goes:

You want the "MySQL Community Server":

http://dev.mysql.com/downloads/mysql/

- make sure you choose the appropriate download for your operating system and OS version
- Windows:
    - in most cases, you will want "Windows (x86, 32-bit), MSI Installer"
    - the installer should help you to configure a reasonable setup, but if you like to roll your own, google "MySQL installation windows". There are many (sometimes conflicting) instructions
    - apparently there is a fancy new complete MySQL product package and installer at: http://dev.mysql.com/downloads/

Tip for all:

- For fine-tuning, find and edit your mysql configuration file (my.cnf or my.ini) using any plain text editor. Google "MySQL configuration file"

GUIs (Graphical User Interfaces) for working with MySQL

You don't need these if you work strictly command line, but they can be handy, especially for getting data in and out of MySQL, which can be trickly. The following are free, run on both Windows and Mac, and provide interfaces for interacting with MySQL on a remote server. The best I know are:

Navicat:
http://www.navicat.com/

Navicat is $$$ software, but there is a free version, 'Navicat Lite', hard to find; here it is for pc:
http://download.cnet.com/windows/database-software/?tag=404
and for mac:
http://download.cnet.com/Navicat-Lite-Free-Multiple-Databases-GUI/3000-10254_4-75119669.html

MySQL Workbench:
http://www.mysql.com/downloads/workbench/

Generally, Navicat is the best all-round solution. Straightforward interface without all the confusing extras of MySQL Workbench. Furthermore, Navicat and has one big feature not available in MySQL Workbench: an excellent import utility to help you get csv files into MySQL quickly. For some reason MySQL's own application lacks this essential feature. Also, Navicat can connect to other databases, such as PostgreSQL.

If you need to model a new or complex database, I highly recommend MySQL Workbench's modelling and design tool.

Setting up a VPN (Virtual Private Network) connection

If you are connecting to a server at the University of Arizona, you will need to do this extra step first. The University of Arizona provides an *extra* layer of protection that you must get through before you can log on remotely to any computer on campus. Basically, on-campus computers are set up to reject connections from computers with IP addresses indicating they are not on campus. With VPN, after you have identified yourself as an authorized U of A member with your UA login and password, the software assignes your logging-in-from-home-with-Cox-Internet computer a proxy IP address that tricks on campus computers into thinking you're on campus.

If you don't already know how to set up a VPN connection, it is worth learning. Setting up a VPN will also let you download journal articles for free as if you were on campus.

You can establish a VPN login from any remote computer by opening up a browser and going to:

https://vpn.arizona.edu/

Warning: does not work on Safari. Use Internet Explorer or Firefox.

If the browser version doesn't work, try downloading and installing VPN client software on your compute. See:

http://uits.arizona.edu/services/vpn_support

If you are already on campus, and connected to the internet via a network cable, you probably already have an on-campus IP address and should not need to use VPN (although it won't hurt). However, if you have connected via wireless to the UA Public network, you will probably still need to authenticate using VPN.

Connecting to MySQL on a remote server using command line

The following examples assume you have shell access (a user login and password) to the remote server. I can set you up on the Enquist lab server. Ask me! Also, you will be using the Unix operating system. Don't be afraid. Knowledge is power!

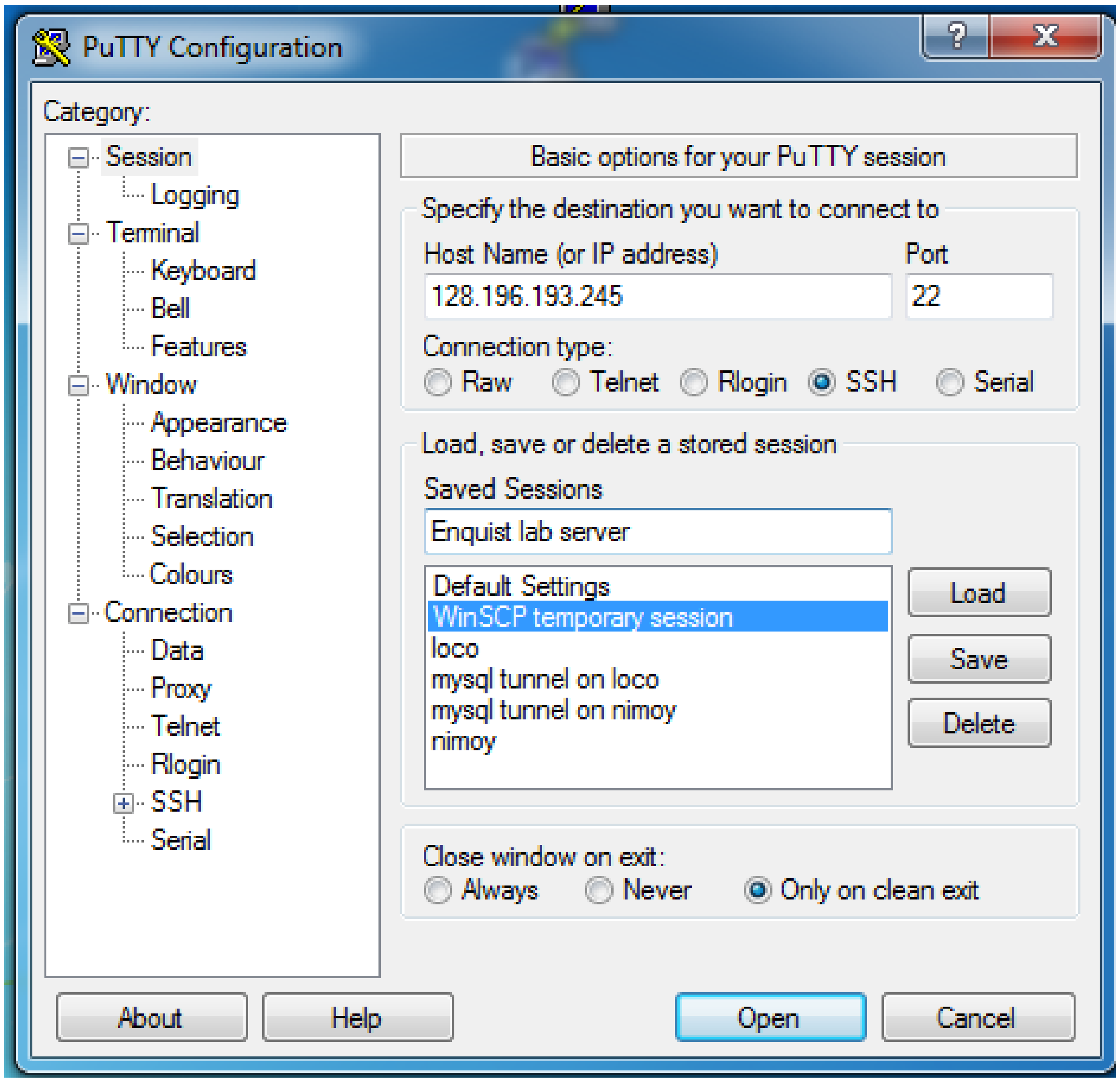
If you get excited about Unix and want to learn more (be afraid: absolute power corrupts absolutely!), please read the appendix "A (very) basic Unix tutorial" at the end of this document.

Mac people

- All you need is "Terminal" which is already installed on your Mac.
- You may also want to install an ftp/scp application, which lets you drag files between finder or your desktop and the remote computer. Try Cyberduck (http://cyberduck.ch/).

Windows users

- You need to install a Terminal-like application.
- Go to https://sitelicense.arizona.edu/ssh/, click on Download the Putty shell client, log in, and install the software



Specify the destination you want to connect to

Host Name (or IP address): 128.196.193.245

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

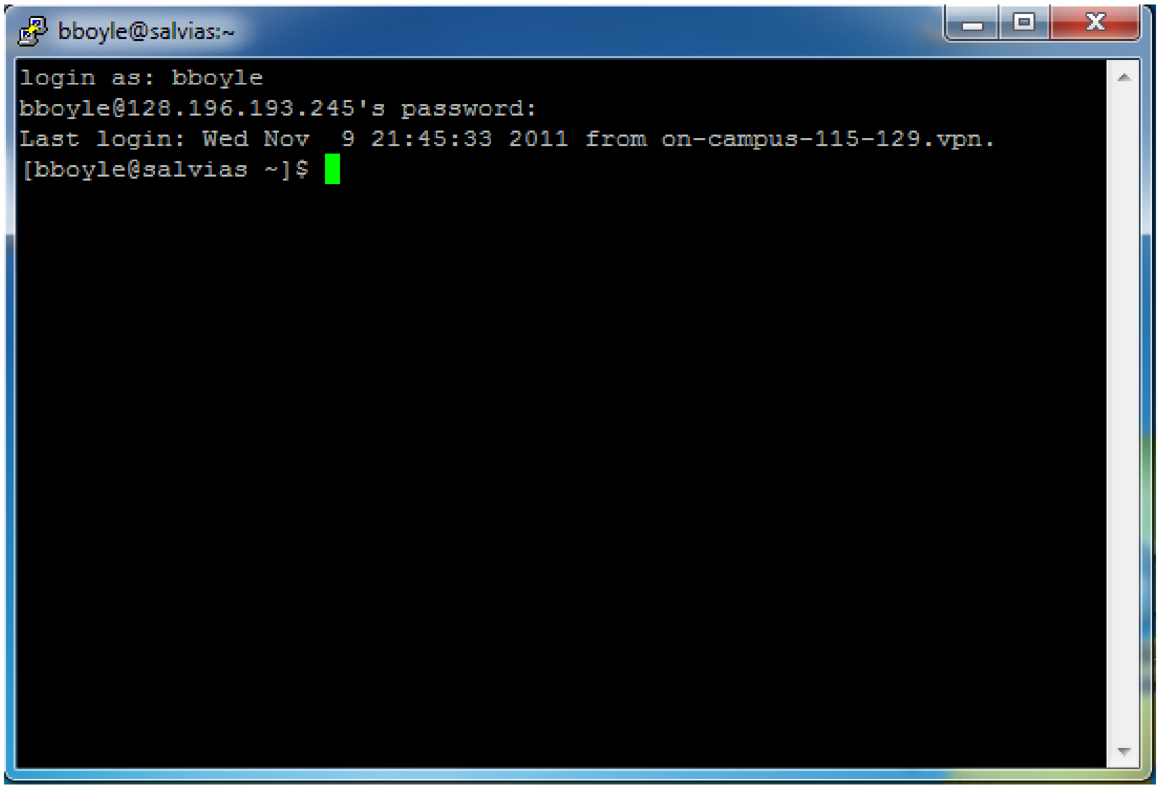
Load, save or delete a stored session

Saved Sessions: Enquist lab server

Default Settings
WinSCP temporary session
loco
mysql tunnel on loco
mysql tunnel on nimoy
nimoy

Close window on exit: Always Never [✓ Only on clean exit]

- The session "Enquist lab server" should now appear as a Saved Session. Click on it, and press 'Open'.
- A terminal window will open (with ugly back background; ask me if you want to switch to black text on white background)
- At the prompt, enter your username, then your password:



```
login as: bboyle
bboyle@128.196.193.245's password:
Last login: Wed Nov  9 21:45:33 2011 from on-campus-115-129.vpn.
[bboyle@salvias ~]$
```

You're in!

<u>Step 2b: Logging out (ending your session)</u>

While you're at it, you'll need to know how to get out. Just enter "exit". That's it, you're done.

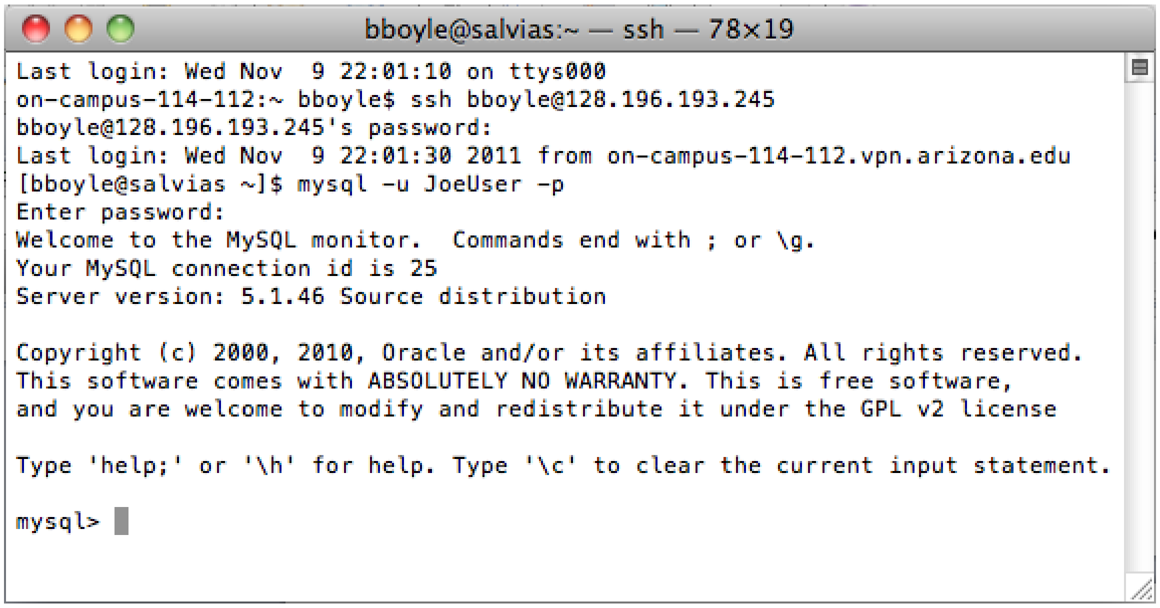
OK, now log back in again. From here on in, the steps are the same for Windows or Mac.

<u>Step 3. Log into MySQL</u>

Enter the command 'mysql' followed by the switch '-u' (user), your user name, and the switch '-p' (using password). Here's an example of the command logging in as 'JoeUser':

```
mysql –u JoeUser –p
```

Here's what an actual session looks like:



```
Last login: Wed Nov  9 22:01:10 on ttys000
on-campus-114-112:~ bboyle$ ssh bboyle@128.196.193.245
bboyle@128.196.193.245's password:
Last login: Wed Nov  9 22:01:30 2011 from on-campus-114-112.vpn.arizona.edu
[bboyle@salvias ~]$ mysql -u JoeUser -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 5.1.46 Source distribution

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

Connecting to MySQL on a remote server using phpMyAdmin

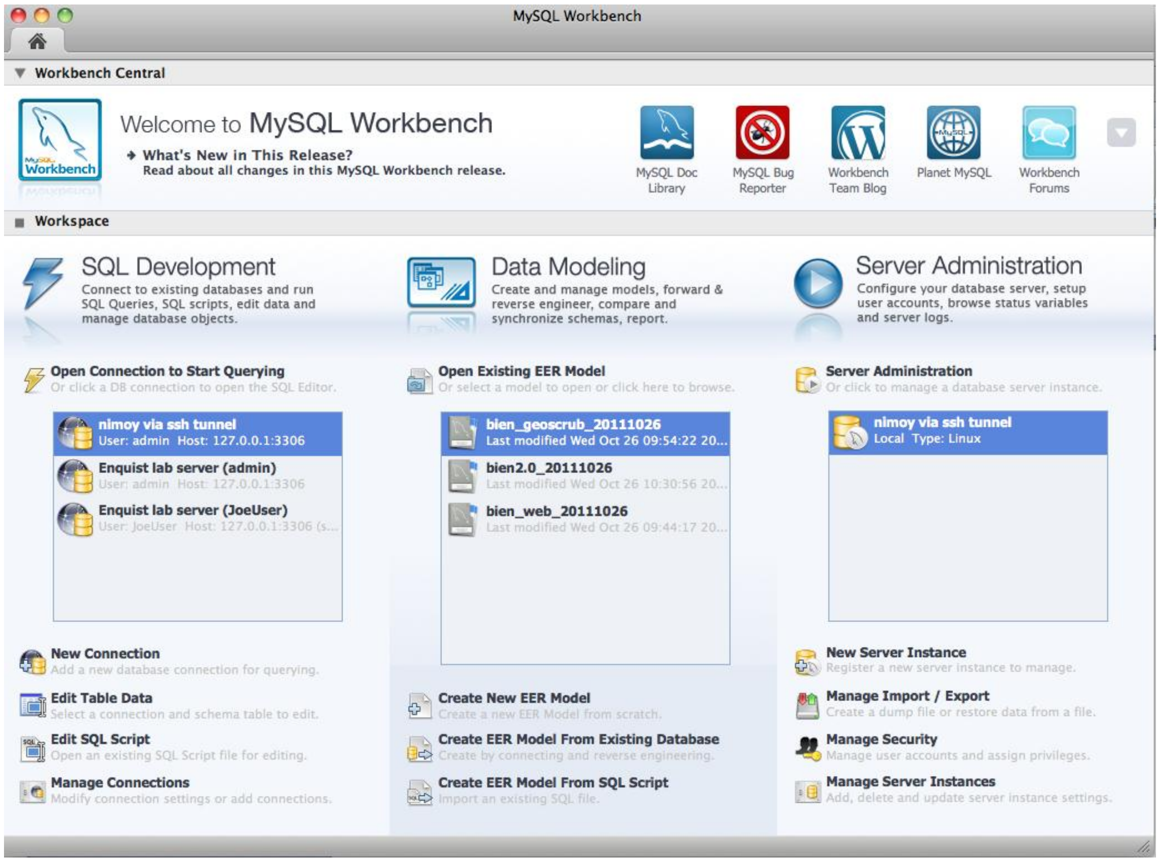
All you need to do is paste the address into a browser (bookmark it!) and enter your username and password. We do not yet have phpMyAdmin set up on the server, but will soon. I'll keep you posted.

Other user interfaces, out of the box, will be blocked from connecting to the server (this is pretty standard for most servers). To get around this block you need to set up something called an 'ssh tunnel'. Basically, you spoof the remote machine into thinking that you are logging in locally. To do this you must have a login and password on the remote machine.

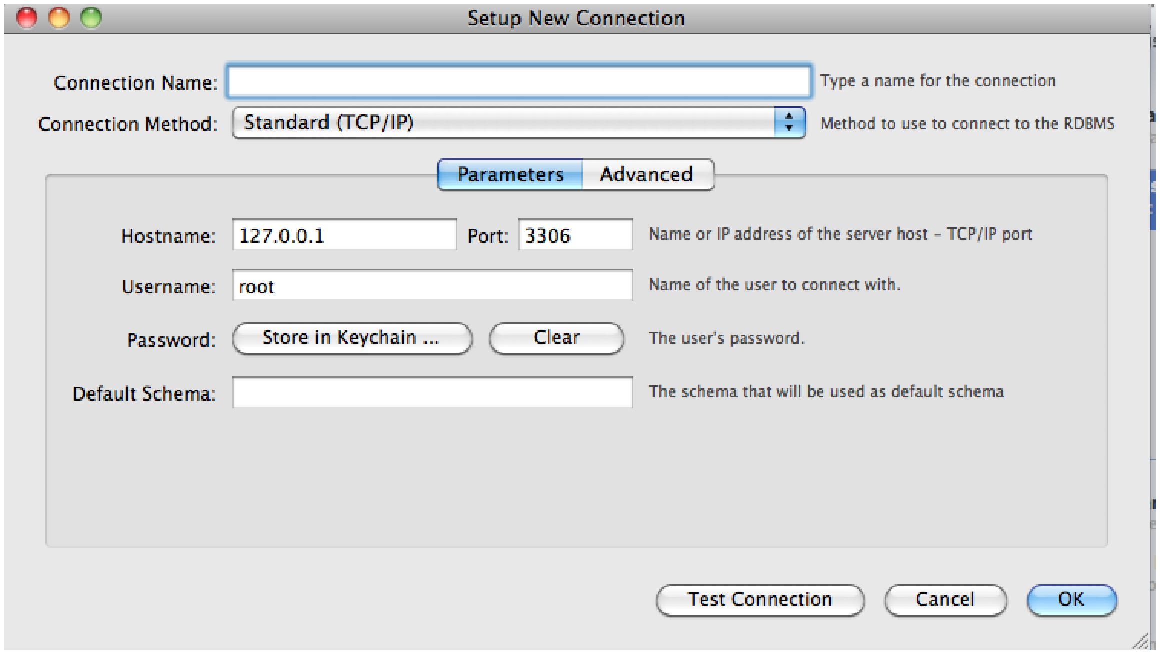
Connecting to MySQL on a remote server using MySQL Workbench

You MUST set up a VPN connection first to connect with MySQL Workbench. Read above if you don't know what this is.

VPN all set? Good. Now, open up MySQL Workbench. Here's what the basic interface looks like on my machine. Ignore all my existing saved connections; you won't have any when you first log on:

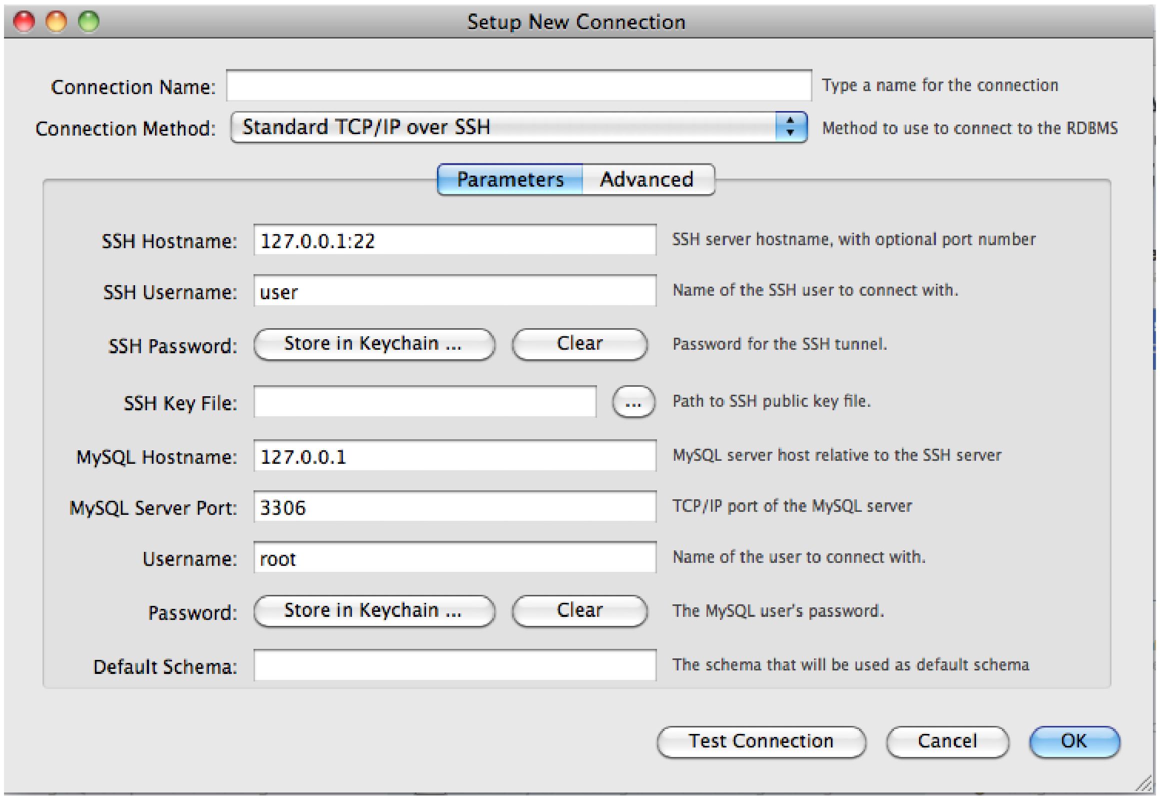

MySQL Workbench
Workbench Central
Welcome to MySQL Workbench
What's New in This Release?
Read about all changes in this MySQL Workbench release.
MySQL Doc Library
MySQL Bug Reporter
Workbench Team Blog
Planet MySQL
Workbench Forums
Workspace
SQL Development
Connect to existing databases and run SQL Queries, SQL scripts, edit data and manage database objects.
Data Modeling
Create and manage models, forward & reverse engineer, compare and synchronize schemas, report.
Server Administration
Configure your database server, setup user accounts, browse status variables and server logs.
Open Connection to Start Querying
Or click a DB connection to open the SQL Editor.
nimoy via ssh tunnel
User: admin Host: 127.0.0.1:3306
Enquist lab server (admin)
User: admin Host: 127.0.0.1:3306
Enquist lab server (JoeUser)
User: JoeUser Host: 127.0.0.1:3306 (s...
Open Existing EER Model
Or select a model to open or click here to browse.
bien_geoscrub_20111026
Last modified Wed Oct 26 09:54:22 20...
bien2.0_20111026
Last modified Wed Oct 26 10:30:56 20...
bien_web_20111026
Last modified Wed Oct 26 09:44:17 20...
Server Administration
Or click to manage a database server instance.
nimoy via ssh tunnel
Local Type: Linux
New Connection
Add a new database connection for querying.
Edit Table Data
Select a connection and schema table to edit.
Edit SQL Script
Open an existing SQL Script file for editing.
Manage Connections
Modify connection settings or add connections.
Create New EER Model
Create a new EER Model from scratch.
Create EER Model From Existing Database
Create by connecting and reverse engineering.
Create EER Model From SQL Script
Import an existing SQL file.
New Server Instance
Register a new server instance to manage.
Manage Import / Export
Create a dump file or restore data from a file.
Manage Security
Manage user accounts and assign privileges.
Manage Server Instances
Add, delete and update server instance settings.

There's a lot of stuff here you don't need. To set up your first connection, click "New Connection" on the left under "SQL Development". You'll get a panel looking something like this:



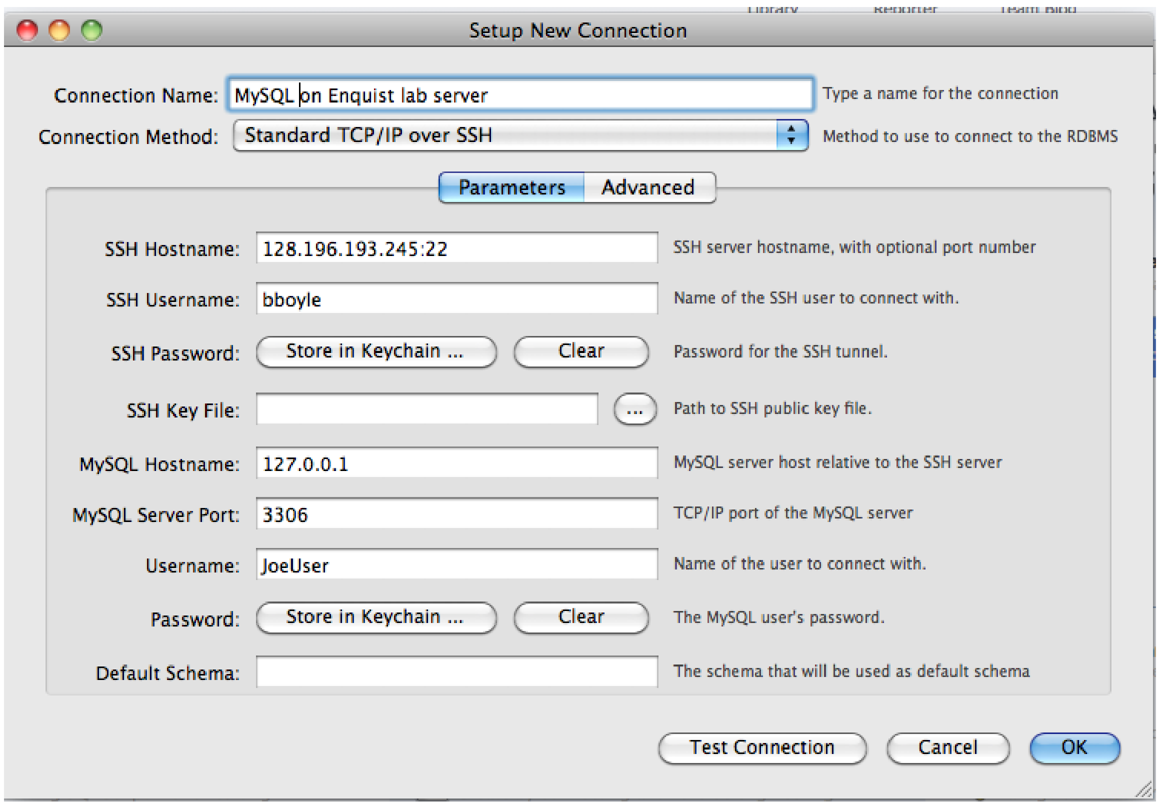
Connection Name: Type a name for the connection
Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS
Hostname: 127.0.0.1 Port: 3306 Name or IP address of the server host - TCP/IP port
Username: root Name of the user to connect with.
Password: The user's password.
Default Schema: The schema that will be used as default schema

Under “Connection Method” choose “Standard TCP/IP over SSH” and you’ll see this:



Connection Name:		Type a name for the connection
Connection Method:	Standard TCP/IP over SSH	Method to use to connect to the RDBMS
SSH Hostname:	127.0.0.1:22	SSH server hostname, with optional port number
SSH Username:	user	Name of the SSH user to connect with.
SSH Password:	Store in Keychain ... Clear	Password for the SSH tunnel.
SSH Key File:		Path to SSH public key file.
MySQL Hostname:	127.0.0.1	MySQL server host relative to the SSH server
MySQL Server Port:	3306	TCP/IP port of the MySQL server
Username:	root	Name of the user to connect with.
Password:	Store in Keychain ... Clear	The MySQL user's password.
Default Schema:		The schema that will be used as default schema

Fill it out as follows, user your server login name (instead of ‘bboyle’) for “SSH Username” and your MySQL user name (instead of JoeUser) for MySQL. Note the actuall server address under “SSH Hostname”. The IP address “125.0.0.1” under MySQL Hostname is “localhost”, meaning, the server itself. This is how you spoof the server into thinking you are connecting locally. Don’t forget the :22 (forwarding port) after the SSH Hostname, and leave the MySQL Server Port as is (should be 3306).


Setup New Connection

Connection Name:	MySQL on Enquist lab server	Type a name for the connection
Connection Method:	Standard TCP/IP over SSH	Method to use to connect to the RDBMS
SSH Hostname:	128.196.193.245:22	SSH server hostname, with optional port number
SSH Username:	bboyle	Name of the SSH user to connect with.
SSH Password:		Password for the SSH tunnel.
SSH Key File:		Path to SSH public key file.
MySQL Hostname:	127.0.0.1	MySQL server host relative to the SSH server
MySQL Server Port:	3306	TCP/IP port of the MySQL server
Username:	JoeUser	Name of the user to connect with.
Password:		The MySQL user's password.
Default Schema:		The schema that will be used as default schema

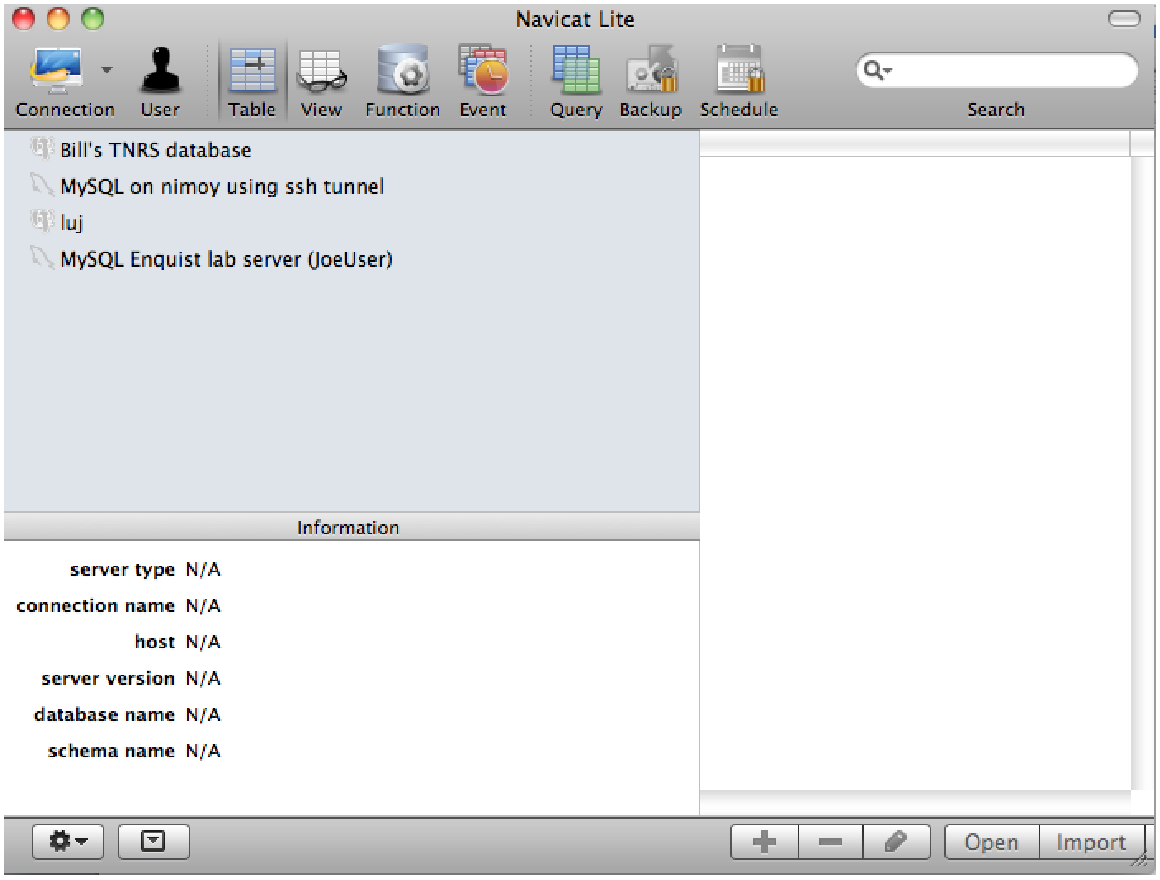
Click OK.

Back in the MySQL Workbench admin panel, click on the new connection you've just created. Enter your two passwords when prompted, and…you're in, ready to query!

Connecting to MySQL on a remote server using Navicat Lite

You MUST set up a VPN connection first to connect with MySQL Workbench. Read above if you don't know what this is.

Open Navicat Lite. You'll see something like this, minus my existing connections:



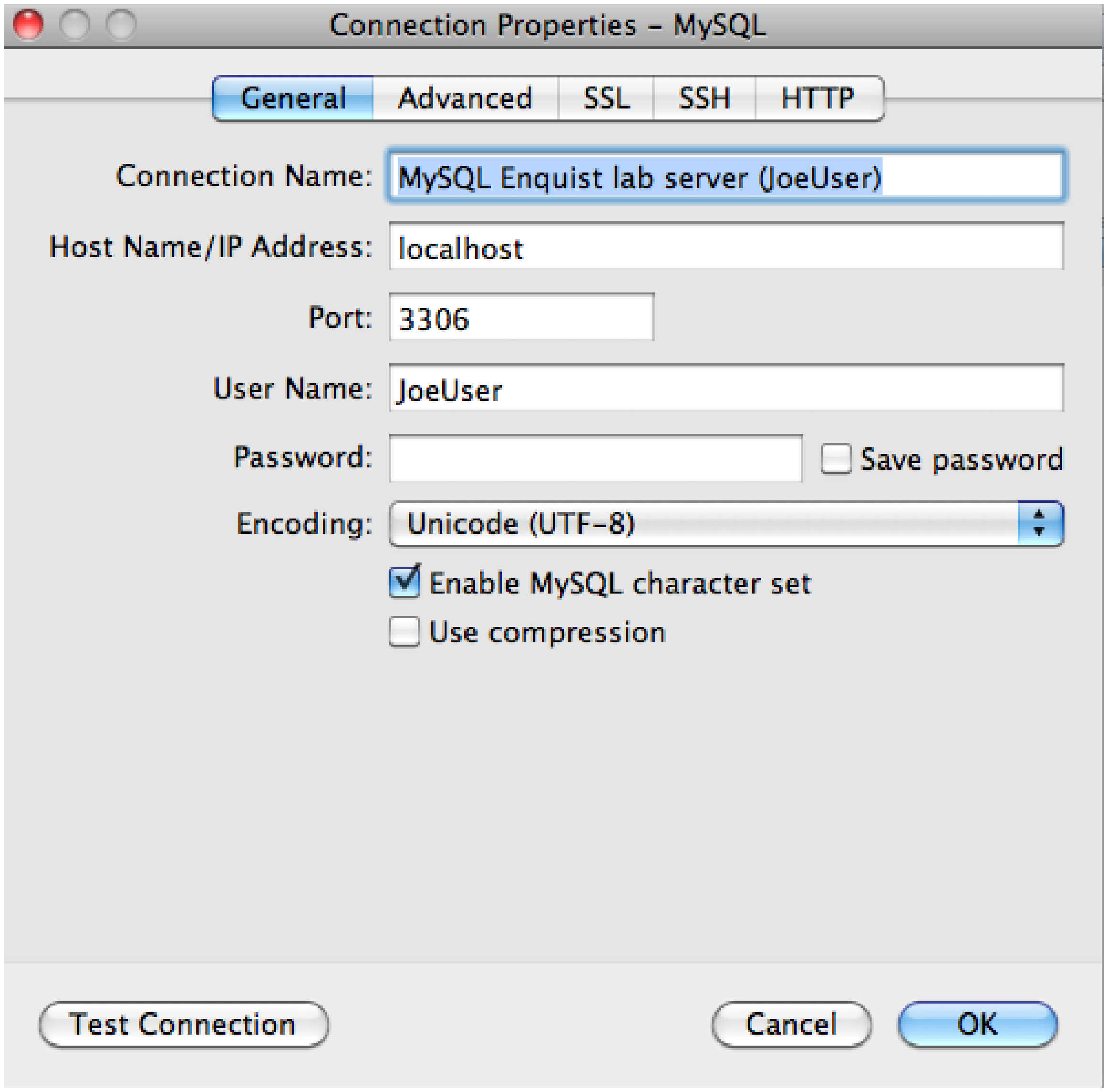
Bill's TNRS database
MySQL on nimoy using ssh tunnel
luj
MySQL Enquist lab server (JoeUser)

Information

server type	N/A
connection name	N/A
host	N/A
server version	N/A
database name	N/A
schema name	N/A

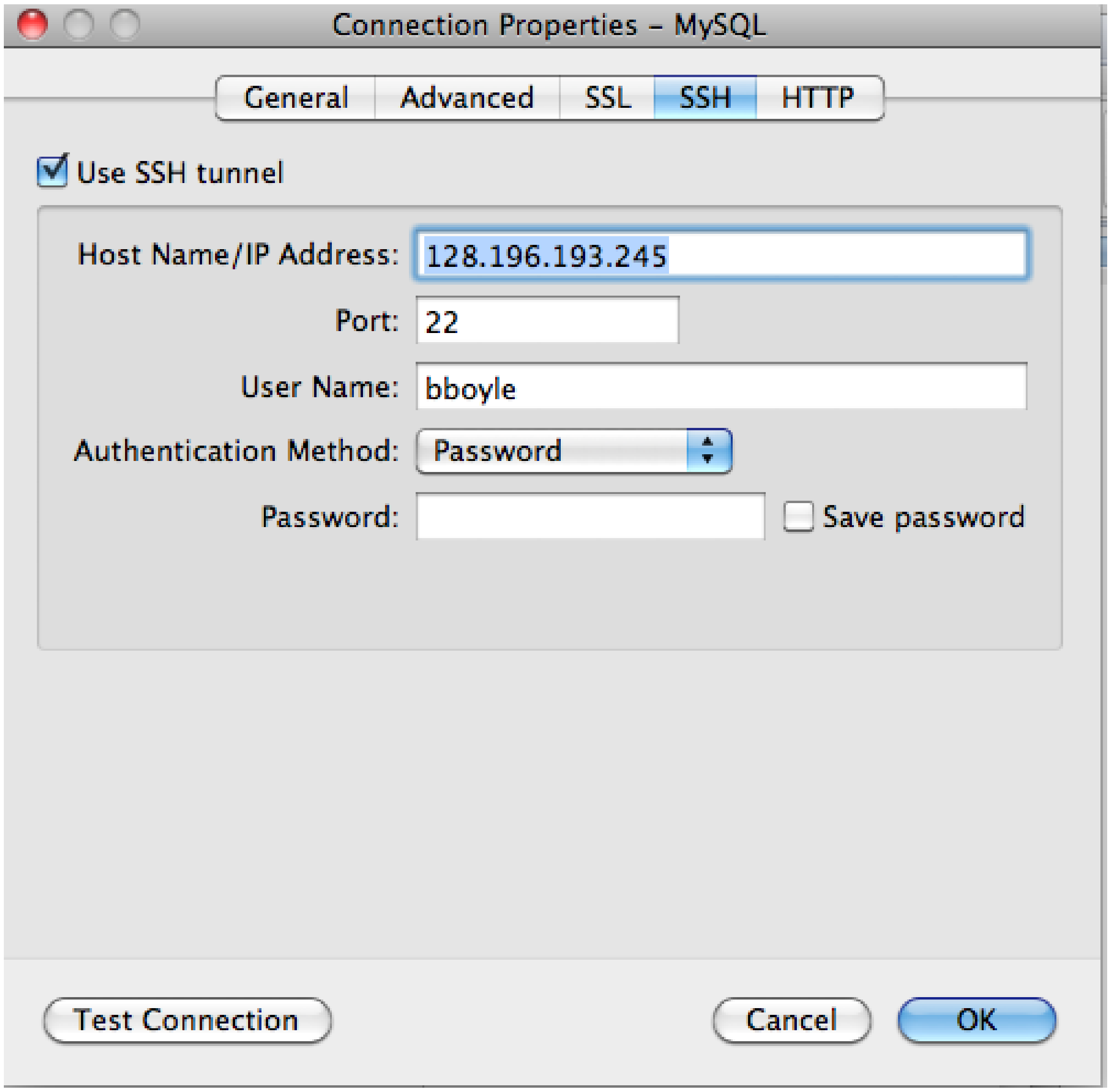
Click ‘Connection’ → ‘MySQL’

Under the general tab, enter the following, using your user name instead of JoeUser. Note that you can user either ‘localhost’ or ‘127.0.0.1’; they mean the same thing: the local server. Give the connection some memorable name.



Connection Name:	MySQL Enquist lab server (JoeUser)
Host Name/IP Address:	localhost
Port:	3306
User Name:	JoeUser
Password:	☐ Save password
Encoding:	Unicode (UTF-8)
	✓ Enable MySQL character set
	☐ Use compression

Next, select the 'SSH' tab, check the 'Use SSH tunnel' button, and fill out the rest of the form as follows, using your username (for the remote server, not for MySQL) instead of 'bboyle':



☑ Use SSH tunnel

Host Name/IP Address:	128.196.193.245
Port:	22
User Name:	bboyle
Authentication Method:	Password
Password:	☐ Save password

Leave the password box blank. It is not secure to store your password in the application! You will be asked for you passwords during the login process.

Press OK.

Back at the main Navicat admin panel, click on the connection you've just created. Navicat does things backward, so you will be asked for you MySQL password first, then your ssh password (the one you use to connect to the server itself).

There. You've just set up an ssh tunnel, and are connected to MySQL on the remote machine. Start querying!

QUERYING DATA: A SAMPLE MYSQL SESSION

Here's a sample session, with me logged in as JoeUser. JoeUser has full privileges on the database `shared`. You'll be logged in as yourself, so you will be able to see shared, as well as any other databases to which you have been given access.
Feel free to try these queries yourself.

As JoeUser, I'm going to explore the database `shared`, a simple, 3-table relational database of plot data. Table `plots` has one line per plot. `plot_observations` contains observations of individual trees in plots. `stems` contains one or more stem measurements (of dbh, or diameter at breast height, in cm) for individual trees. A plot can have one or more trees in the table plot_observations, and an individual tree can have 0, 1 or more stem measurements in the table `stems` (I'm just telling; database geeks will notice that I haven't actually enforced these rules with a foreign key constraint, but…hey, let's keep this simple).

Commands I type are in **bold**, my comments as humble author of this document are in *italics,* and everything else that the terminal spits out (the server talking to me) is in regular type face. I'll switch to ugly, equal-width courier font so raw terminal output lines up properly.

Throughout, I type SQL-specific reserved words in UPPERCASE. This helps distinguish visually between commands and variable names. But SQL does not require you to type in caps; it is perfectly fine to use all lowercase. Note however, that in most cases (depending on your installation of MySQL) table and column names must be written using the exact same case they were name with. Thus, if you have a table called 'Users', MySQL may not recognized it if you write 'users'.

Last bit of advice: when in doubt, google it. MySQL has an excellent (if dauntingly detailed) online manual. The manual pages will almost always be your top hits when you google "mysql functions" or "mysql create table". The next hits just below are usually excellent examples, walk-through tutorials and tips.

The basics

Logging in, viewing databases, selecting a database to work with, and viewing its tables and their properties.

```
[bboyle@salvias ~]$ mysql -u JoeUser -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.1.46 Source distribution

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All
rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free
software,
and you are welcome to modify and redistribute it under the GPL
v2 license
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| shared             |
+--------------------+
2 rows in set (0.00 sec)

mysql> USE SHARED;
```

You must tell mysql which database to use. `shared` is the database we're after. `Information_schema` belongs to mysql and hold metadata about all the tables in all your databases. Ignore it for now.

```
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with -A

Database changed

mysql> SHOW TABLES;
```

I ask mysql to list the tables in the database. Note the semi-colon after the command. You must include it for mysql to execute the command. By the way, most GUIs (such as Navicat) use an "execute" button to execute your SQL, so don't require the final semi-colon.

```
+-------------------+
| Tables_in_shared  |
+-------------------+
| plot_observations |
| plots             |
| stems             |
+-------------------+
3 rows in set (0.00 sec)

mysql> DESCRIBE PLOTS;
```

This gives a list of all the columns in the table, whether or not they can be NULL, their data types, indexes and default values.

```
+-----------+---------------+------+-----+---------+-------+
| Field     | Type          | Null | Key | Default | Extra |
+-----------+---------------+------+-----+---------+-------+
| PlotID    | int(11)       | NO   | PRI | NULL    |       |
| plotCode  | varchar(255)  | NO   |     | NULL    |       |
| country   | varchar(255)  | NO   |     | NULL    |       |
| state     | varchar(255)  | YES  |     | NULL    |       |
| latitude  | decimal(10,2) | YES  |     | NULL    |       |
| longitude | decimal(10,2) | YES  |     | NULL    |       |
| elevation | int(11)       | YES  |     | NULL    |       |
+-----------+---------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

The 'PRI' under 'Key' tells you that PlotID is the primary key of this table. Also note that columns state, latitude, longitude and elevation are allowed to be NULL (=no value), but PlotID, plotCode and country are not; the must have a value. Don't ask me why Default value says NULL for those three columns; I think that's just a bug or quirk of mysql.

```
mysql> DESCRIBE plot_observations;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| PlotObsID | int(11)      | NO   | PRI | NULL    |       |
| PlotID    | int(11)      | NO   |     | NULL    |       |
| Line      | varchar(255) | NO   |     | NULL    |       |
| Ind       | int(11)      | NO   |     | NULL    |       |
| Family    | varchar(255) | NO   |     | NULL    |       |
| Genus     | varchar(255) | NO   |     | NULL    |       |
| Species   | varchar(255) | YES  |     | NULL    |       |
| Habit     | varchar(255) | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

'PlotObsID' is the primary key of table `plot_observations`. Also notice the column 'PlotID'; that's the foreign key linking to the table 'plots'. It's an integer data type, which is usual for primary/foregn keys So far so good. Note Species (actually, specific epithet) is allowed to be NULL, but all the other columns must have a value.

```
mysql> DESCRIBE stems;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| stem_id   | int(11) | NO   |     | NULL    |       |
| PlotObsID | int(11) | NO   |     | NULL    |       |
| dbh       | float   | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

'Float' means that the column 'dbh' is a decimal data type.

ALTER TABLE

Wait a second - 'stem_id' should be the primary key. I forgot to set it. No worries, we can fix that with an ALTER TABLE statement:

```
mysql> ALTER TABLE stems
    -> ADD PRIMARY KEY(stem_id);
Query OK, 3423 rows affected (0.06 sec)
Records: 3423  Duplicates: 0  Warnings: 0

+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| stem_id   | int(11) | NO   | PRI | NULL    |       |
| PlotObsID | int(11) | NO   |     | NULL    |       |
| dbh       | float   | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

That's better. Note how mysql let me enter my query on several lines. It waits to execute until I enter that semicolon.

SELECT…FROM

Now let's have a look at the data…

```
mysql> SELECT * FROM plots;
+--------+------------+------------+------------+----------+-----------+-----------+
| PlotID | plotCode   | country    | state      | latitude | longitude | elevation |
+--------+------------+------------+------------+----------+-----------+-----------+
|    327 | CM-1       | Costa Rica | NULL       |     NULL |      NULL |      2880 |
|    328 | CM-2       | Costa Rica | NULL       |     NULL |      NULL |      2950 |
|    329 | CM2001     | Costa Rica | NULL       |     NULL |      NULL |      3240 |
|    330 | CMPAC      | Costa Rica | NULL       |     NULL |      NULL |      2800 |
|    331 | CMPac2     | Costa Rica | NULL       |     NULL |      NULL |      2775 |
|    334 | ElSitio    | Costa Rica | NULL       |     NULL |      NULL |      2600 |
|    335 | ElSitio2   | Costa Rica | NULL       |     NULL |      NULL |      2730 |
|    336 | LS-00-9    | Costa Rica | NULL       |    10.40 |    -84.00 |        50 |
|    337 | LS-1       | Costa Rica | Heredia    |    10.40 |    -84.00 |        50 |
|    338 | LS-2       | Costa Rica | Heredia    |    10.40 |    -84.00 |        50 |
|    341 | MT         | Costa Rica | NULL       |    10.80 |    -84.80 |      1550 |
|    342 | MV-2001    | Costa Rica | NULL       |    10.80 |    -84.80 |      1500 |
|    343 | PV-1       | Costa Rica | Puntarenas |    10.50 |    -85.30 |        40 |
|    344 | PV-1-00    | Costa Rica | Puntarenas |    10.50 |    -85.30 |       130 |
|    345 | PV-2       | Costa Rica | NULL       |    10.50 |    -85.30 |        10 |
|    346 | PV-2001    | Costa Rica | NULL       |    10.50 |    -85.30 |       100 |
|    347 | PV-2B      | Costa Rica | NULL       |    10.50 |    -85.30 |        40 |
|    348 | PV-2S      | Costa Rica | NULL       |    10.50 |    -85.30 |        40 |
|    349 | PV-Aug-99  | Costa Rica | NULL       |    10.50 |    -85.30 |        40 |
|    350 | PVRiverine | Costa Rica | Puntarenas |    10.50 |    -85.30 |        10 |
+--------+------------+------------+------------+----------+-----------+-----------+
20 rows in set (0.00 sec)
```

I just asked mysql to show me the contents of every field in the table. That's what the '' means. Note the NULL values for state, latitude and longitude on some records. If you're using a GUI, they may just look blank instead of saying 'NULL'.*

The COUNT(*) function

```
mysql> SELECT COUNT(*) FROM plots;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

I used the COUNT function to find out how many plots (records) in the table. Of course that's unnecessary because MySQL already told me how many records at the end of the last query. But we'll do something more interesting with COUNT() shortly. Now, let's have a look at plot_observations. I suspect that's a big table, so maybe I don't want to show all the records, just a sample. Let's check:

```
mysql> SELECT COUNT(*) FROM plot_observations;
+----------+
| COUNT(*) |
+----------+
|     2934 |
+----------+
1 row in set (0.00 sec)
```

Just as I thought. In that case, let's set a limit to show only the first 10 records:

LIMITING records

```
mysql> SELECT * FROM plot_observations LIMIT 10;
+-----------+--------+------+-----+---------------+------------+-------------+-------+
| PlotObsID | PlotID | Line | Ind | Family        | Genus      | Species     | Habit |
+-----------+--------+------+-----+---------------+------------+-------------+-------+
|     21425 |    327 | I    |   1 | Myrsinaceae   | Myrsine    | floridana   | T     |
|     21426 |    327 | I    |   2 | Rubiaceae     | Palicourea | salicifolia | T     |
|     21427 |     327 | I     |   3 | Aquifoliaceae | Ilex       | pallida     | T      |
|     21428 |     327 | I     |   4 | Styracaceae   | Styrax     | argenteus   | T      |
|     21429 |     327 | I     |   5 | Fagaceae      | Quercus    | copeyensis  | T      |
|     21430 |     327 | I     |   6 | Rubiaceae     | Palicourea | salicifolia | T      |
|     21431 |     327 | I     |   7 | Rubiaceae     | Palicourea | salicifolia | T      |
|     21432 |     327 | I     |   8 | Rubiaceae     | Palicourea | salicifolia | T      |
|     21433 |     327 | I     |   9 | Aquifoliaceae | Ilex       | pallida     | T      |
|     21434 |     327 | I     |  10 | Cornaceae     | Cornus     | disciflora  | T      |
+-----------+---------+-------+-----+---------------+------------+-------------+--------+
10 rows in set (0.00 sec)
```

The INNER JOIN

Now, those plot IDs are a hard to read, so let's join to the plot table to see what the name of this plot is:

```
mysql> SELECT plotCode, Family, Genus, Species
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> LIMIT 10;
+----------+---------------+------------+-------------+
| plotCode | Family        | Genus      | Species     |
+----------+---------------+------------+-------------+
| CM-1     | Myrsinaceae   | Myrsine    | floridana   |
| CM-1     | Rubiaceae     | Palicourea | salicifolia |
| CM-1     | Aquifoliaceae | Ilex       | pallida     |
| CM-1     | Styracaceae   | Styrax     | argenteus   |
| CM-1     | Fagaceae      | Quercus    | copeyensis  |
| CM-1     | Rubiaceae     | Palicourea | salicifolia |
| CM-1     | Rubiaceae     | Palicourea | salicifolia |
| CM-1     | Rubiaceae     | Palicourea | salicifolia |
| CM-1     | Aquifoliaceae | Ilex       | pallida     |
| CM-1     | Cornaceae     | Cornus     | disciflora  |
+----------+---------------+------------+-------------+
10 rows in set (0.00 sec)
```

This is called an INNER JOIN: it shows records where the joining field are exactly equal in both tables. MySQL lets you write either JOIN or INNER JOIN. They are synonymous.

Aliases

In the query above, where'd all those p's and o's come from? They are aliases for the tables, so mysql knows where to get the columns. I could have written the query as:

```
SELECT plotCode, Family, Genus, Species
FROM plots JOIN plot_observations
ON plots.PlotID=plot_observations.PlotID
WHERE plotCode='MT';
```

Strictly speaking, MySQL only needs the alias for columns that are found in both tables (in this case, PlotID), but it's often good practice to put an alias in front of each field in a multi-table join. This makes your query a lot clearer once you start adding lots of columns and tables.

The WHERE clause

OK, what if we want to see the records for a specific plot? Say, plotCode='MT'? Just add a WHERE clause:

```
mysql> SELECT plotCode, Family, Genus, Species
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='MT'
    -> LIMIT 10;
+----------+------------+------------+-------------+
| plotCode | Family     | Genus      | Species     |
+----------+-----------+------------+--------------+
| MT       | Arecaceae  | Chamaedorea | tepejilote   |
| MT       | Arecaceae  | Chamaedorea | tepejilote   |
| MT       | Arecaceae  | Geonoma     | edulis       |
| MT       | Arecaceae  | Geonoma     | edulis       |
| MT       | Araliaceae | Oreopanax   | oerstedianus |
| MT       | Arecaceae  | Chamaedorea | tepejilote   |
| MT       | Arecaceae  | Chamaedorea | tepejilote   |
| MT       | Arecaceae  | Chamaedorea | tepejilote   |
| MT       | Arecaceae  | Geonoma     | edulis       |
| MT       | Arecaceae  | Geonoma     | edulis       |
+----------+-----------+------------+--------------+
10 rows in set (0.00 sec)
```

Using DISTINCT to get unique values

That's pretty useles. How about a list of all the species?

```
mysql> SELECT Family, Genus, Species
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='MT';
+-----------------+-----------------+-----------------+
| Family          | Genus           | Species         |
+-----------------+-----------------+-----------------+
| Arecaceae       | Chamaedorea     | tepejilote      |
| Arecaceae       | Chamaedorea     | tepejilote      |
| Arecaceae       | Geonoma         | edulis          |
| Arecaceae       | Geonoma         | edulis          |
| Araliaceae      | Oreopanax       | oerstedianus    |
| Arecaceae       | Chamaedorea     | tepejilote      |
| Arecaceae       | Chamaedorea     | tepejilote      |
| Arecaceae       | Chamaedorea     | tepejilote      |
| Arecaceae       | Geonoma         | edulis          |
| Arecaceae       | Geonoma         | edulis          |
| Asteraceae      | Clibadium       | anceps          |
| Asteraceae      | Koanophyllon    | pittieri        |
| Asteraceae      | Koanophyllon    | pittieri        |

. . . many other records . . .

| Urticaceae      | Urera           | elata           |
| Urticaceae      | Urera           | elata           |
| Verbenaceae     | Citharexylum    | donnell-smithii |
| Verbenaceae     | Citharexylum    | donnell-smithii |
| Verbenaceae     | Citharexylum    | donnell-smithii |
| Verbenaceae     | Citharexylum    | donnell-smithii |
+-----------------+-----------------+-----------------+
214 rows in set (0.01 sec)
```

Oops! What happened here is we got a list of the species name of each individual in the plot, not a list of species. In fact, if we want a count of individuals in that plot, all is COUNT the lines in the above query:

```
mysql> SELECT COUNT(*)
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='MT';
+----------+
| COUNT(*) |
+----------+
|      214 |
+----------+
1 row in set (0.00 sec)
```

But we wanted Species. First, let's see how we get a list of species before we count them (let's just do Genus species, skip family). For this, we'll need the key word DISTINCT:

```
mysql> SELECT DISTINCT Genus, Species
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE  plotCode='MT';
+-----------------+-----------------+
| Genus           | Species         |
+-----------------+-----------------+
| Chamaedorea     | tepejilote      |
| Geonoma         | edulis          |
| Oreopanax       | oerstedianus    |
| Clibadium       | anceps          |
| Koanophyllon    | pittieri        |
| Senecio         | parasiticus     |
| Quararibea      | costaricensis   |
| Cordia          | cymosa          |
| Cecropia        | obtusifolia     |
| Cecropia        | polyphlebia     |
| Chrysochlamys   | psychotriifolia |
| Clusia          | sp.1            |
| Cyathea         | nigripes        |
| Alsophila       | polystichoides  |
| Justicia        | aurea           |
| Alchornea       | latifolia       |
| Sapium          | rigidifolium    |
| Inga            | hintonii        |
| Inga            | longispica      |
| Inga            | mortoniana      |
| Mucuna          | urens           |
| Casearia        | tacanensis      |
| Hasseltia       | floribunda      |
| Beilschmiedia   | costaricensis   |
| Nectandra       | smithii         |
| Ocotea          | meziana         |
| Ocotea          | sp.1            |
| Ocotea          | tonduzii        |
| Persea          | americana       |
| Pleurothyrium   | palmanum        |
| Bunchosia       | macrophylla     |
| Tetrapterys     | monteverdensis  |
| Hampea          | appendiculata   |
| Guatteria       | verrucosa       |
| Tabernaemontana | longipes        |
| Tabernaemontana | sp.1            |
| Malvaviscus     | palmanus        |
| Conostegia      | formosa         |
| Conostegia      | rufescens       |
| Conostegia       | sp.1             |
| Meriania         | phlomoides       |
| Miconia          | sp.1             |
| Ossaea           | sp.1             |
| Ossaea           | micrantha        |
| Guarea           | glabra           |
| Guarea           | kunthiana        |
| Guarea           | tonduzii         |
| Ruagea           | glabra           |
| Ficus            | crassiuscula     |
| Ardisia          | palmana          |
| Calyptranthes    | pittieri         |
| Eugenia          | guatemalensis    |
| Eugenia          | haberi           |
| Piper            | carpinteranum    |
| Sarcorhachis     | naranjoana       |
| Cassipourea      | elliptica        |
| Coussarea        | austin-smithii   |
| Elaeagia         | auriculata       |
| Guettarda        | poasana          |
| Hoffmannia       | psychotriifolia  |
| Palicourea       | lasiorrhachis    |
| Psychotria       | eurycarpa        |
| Psychotria       | sp.1             |
| Zanthoxylum      | juniperinum      |
| Zanthoxylum      | sp.1             |
| Meliosma         | vernicosa        |
| Paullinia        | austin-smithii   |
| Paullinia        | costaricensis    |
| Pouteria         | fossicola        |
| Cestrum          | megalophyllum    |
| Cestrum          | rugulosum        |
| Lycianthes       | sp.1             |
| Psychotria       | panamensis       |
| Solanum          | rovirosanum      |
| Witheringia      | cuneata          |
| Daphnopsis       | americana        |
| Urera            | elata            |
| Citharexylum     | donnell-smithii  |
+------------------+------------------+
78 rows in set (0.00 sec)
```

Nested queries

Now, let's count species using a nested query:

```
mysql> SELECT COUNT(*) as totSpecies
    -> FROM (
    -> SELECT DISTINCT Genus, Species
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE  p.plotCode='MT'
    -> ) as s;

+------------+
| totSpecies |
+------------+
|         78 |
+------------+
1 row in set (0.01 sec)
```

78 species. Note the alias 's', for the query inside the parentheses. Nested queries must have an alias, even if you don't use that alias anywhere else.

We could have also done the same query more succinctly by placing DISTINCT inside COUNT:

```
mysql> SELECT COUNT(DISTINCT Genus, Species)  as totSpecies
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE  p.plotCode='MT';

+------------+
| totSpecies |
+------------+
|         78 |
+------------+
1 row in set (0.00 sec)
```

Using GROUP BY with aggregate functions

Now, a count of species in each plot. For this we will need a GROUP BY clause. Using the more succinct syntax above:

```
mysql> SELECT plotCode, COUNT(DISTINCT Genus, Species) as
totSpecies
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> GROUP BY plotCode;
+------------+------------+
| plotCode   | totSpecies |
+------------+------------+
| CM-1       |         24 |
| CM-2       |         28 |
| CMPAC      |         19 |
| ElSitio    |         29 |
| ElSitio2   |         26 |
| LS-00-9    |         94 |
| LS-1       |         70 |
| LS-2       |         84 |
| MT         |         78 |
| PV-1       |         54 |
| PV-1-00    |         45 |
| PV-2       |         54 |
| PV-2B      |         14 |
| PV-2S      |         46 |
| PVRiverine |         49 |
+------------+------------+
15 rows in set (0.03 sec)
```

Because table 'plot_observations' is one record per individual, we can use simpler syntax to get a count of all individuals:

```
mysql> SELECT plotCode, COUNT(*) as totIndividuals
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> GROUP BY plotCode;
+------------+----------------+
| plotCode   | totIndividuals |
+------------+----------------+
| CM-1       |            283 |
| CM-2       |            214 |
| CMPAC      |            263 |
| ElSitio    |            169 |
| ElSitio2   |            230 |
| LS-00-9    |            202 |
| LS-1       |            178 |
| LS-2       |            194 |
| MT         |            214 |
| PV-1       |            197 |
| PV-1-00    |            175 |
| PV-2       |            221 |
| PV-2B      |             17 |
| PV-2S      |            204 |
| PVRiverine |            173 |
+------------+----------------+
15 rows in set (0.00 sec)
```

Want a fancy display of both in one query?

```
mysql> SELECT i.plotCode, s.totSpecies, i.totIndividuals
    -> FROM
    -> (
    -> SELECT plotCode, COUNT(*) as totIndividuals
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> GROUP BY plotCode
    -> ) as i
    -> JOIN
    -> (
    -> SELECT plotCode, COUNT(DISTINCT Genus, Species) as
totSpecies
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> GROUP BY plotCode
    -> ) as s
    -> ON i.plotCode=s.plotCode;
+------------+------------+----------------+
| plotCode   | totSpecies | totIndividuals |
+------------+------------+----------------+
| CM-1       |         24 |            283 |
| CM-2       |         28 |            214 |
| CMPAC      |         19 |            263 |
| ElSitio    |         29 |            169 |
| ElSitio2   |         26 |            230 |
| LS-00-9    |         94 |            202 |
| LS-1       |         70 |            178 |
| LS-2       |         84 |            194 |
| MT         |         78 |            214 |
| PV-1       |         54 |            197 |
| PV-1-00    |         45 |            175 |
| PV-2       |         54 |            221 |
| PV-2B      |         14 |             17 |
| PV-2S      |         46 |            204 |
| PVRiverine |         49 |            173 |
+------------+------------+----------------+
15 rows in set (0.03 sec)
```

I didn't have to enter the query on all those separate lines, but I think the above looks much clearer than:

```
mysql> SELECT i.plotCode, s.totSpecies, i.totIndividuals FROM  (
SELECT plotCode, COUNT(*) as totIndividuals FROM plots p JOIN
plot_observations o ON p.PlotID=o.PlotID GROUP BY plotCode  ) as
i JOIN ( SELECT plotCode, COUNT(DISTINCT Genus, Species) as
totSpecies FROM plots p JOIN plot_observations o ON
p.PlotID=o.PlotID GROUP BY plotCode ) as s ON
i.plotCode=s.plotCode;
```

Other aggregate functions: MIN, MAX, AVG

COUNT is an aggregate function. It is not the only one. Let's try a new plot and take a look at stems sizes using MAX, MIN, and AVG.

What's the maximum stem dbh of each species in the plot?

```
mysql> SELECT Genus, Species, MAX(dbh) as maxDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE p.plotCode='CMPAC';
+-------+---------+--------+
| Genus | Species | maxDBH |
+-------+---------+--------+
| Ilex  | pallida |  128.5 |
+-------+---------+--------+
1 row in set (0.02 sec)
```

What the…? Oops, that's what happens when you forget the GROUP BY. Here's the right way to do it:

```
mysql> SELECT Genus, Species, MAX(dbh) as maxDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE p.plotCode='CMPAC'
    -> GROUP BY Genus, Species;
+------------+---------------+--------+
| Genus      | Species       | maxDBH |
+------------+---------------+--------+
| Anthurium  | concinnatum   |    3.2 |
| Cleyera    | theaeoides    |   26.5 |
| Conostegia | oerstediana   |    3.8 |
| Drimys     | granadensis   |    6.3 |
| Ilex       | lamprophylla  |    5.3 |
| Ilex       | pallida       |   15.1 |
| Magnolia   | poasana       |   24.1 |
| Myrsine    | floridana     |     29 |
| Ocotea     | sp.1          |     15 |
| Oreopanax  | nicaraguensis |    9.3 |
| Palicourea | salicifolia   |    8.2 |
| Persea     | vesticula     |      8 |
| Prunus      | annularis     |   12.3 |
| Quercus     | copeyensis    |  128.5 |
| Quercus     | costaricensis |     76 |
| Styrax      | argenteus     |     20 |
| Vaccinium   | consanguineum |   10.4 |
| Viburnum    | costaricanum  |    9.5 |
| Zanthoxylum | melanostictum |    6.4 |
+-------------+---------------+--------+
19 rows in set (0.02 sec)
```

That’s better.

In many cases, you can combine multiple aggregate functions in a single query:

```
mysql> SELECT Genus, Species, MAX(dbh) as maxDBH, MIN(dbh) as minDBH, AVG(dbh) as avgDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE p.plotCode='CMPAC'
    -> GROUP BY Genus, Species;
+------------+---------------+--------+--------+------------------+
| Genus      | Species       | maxDBH | minDBH | avgDBH           |
+------------+---------------+--------+--------+------------------+
| Anthurium  | concinnatum   |    3.2 |    2.5 | 2.67999997138977 |
| Cleyera    | theaeoides    |   26.5 |      3 |  9.4892857330186 |
| Conostegia | oerstediana   |    3.8 |    3.6 | 3.69999992847443 |
| Drimys     | granadensis   |    6.3 |    6.3 | 6.30000019073486 |
| Ilex       | lamprophylla  |    5.3 |    3.9 | 4.52500009536743 |
| Ilex       | pallida       |   15.1 |    2.6 | 6.81428570747376 |
| Magnolia   | poasana       |   24.1 |     13 | 18.5500001907349 |
| Myrsine    | floridana     |     29 |      4 | 10.6272727142681 |
| Ocotea     | sp.1          |     15 |    2.5 | 6.72000007629395 |
| Oreopanax  | nicaraguensis |    9.3 |    3.2 | 5.55833339691162 |
| Palicourea | salicifolia   |    8.2 |    2.2 | 4.44615384248587 |
| Persea     | vesticula     |      8 |      8 |                8 |
| Prunus     | annularis     |   12.3 |    2.5 | 5.71250003576279 |
| Quercus    | copeyensis    |  128.5 |    2.5 | 19.2244187094444 |
| Quercus    | costaricensis |     76 |    3.4 | 25.5000000681196 |
| Styrax     | argenteus     |     20 |      0 | 5.65957443257596 |
| Vaccinium  | consanguineum |   10.4 |    2.9 | 5.46666653951009 |
| Viburnum   | costaricanum  |    9.5 |    2.6 | 6.04999995231628 |
| Zanthoxylum | melanostictum |    6.4 |    2.6 | 3.60000003708733 |
+------------+---------------+--------+--------+------------------+
19 rows in set (0.02 sec)
```

The CAST() function

To format the average to look more readable, use the CAST() function:

```
mysql> SELECT Genus, Species, MAX(dbh) as maxDBH, MIN(dbh) as minDBH, CAST(AVG(dbh) AS
DECIMAL(4,1)) AS avgDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE p.plotCode='CMPAC'
    -> GROUP BY Genus, Species;
+-------------+---------------+--------+--------+--------+
| Genus       | Species       | maxDBH | minDBH | avgDBH |
+-------------+---------------+--------+--------+--------+
| Anthurium   | concinnatum   |    3.2 |    2.5 |    2.7 |
| Cleyera     | theaeoides    |   26.5 |      3 |    9.5 |
| Conostegia  | oerstediana   |    3.8 |    3.6 |    3.7 |
| Drimys      | granadensis   |    6.3 |    6.3 |    6.3 |
| Ilex        | lamprophylla  |    5.3 |    3.9 |    4.5 |
| Ilex        | pallida       |   15.1 |    2.6 |    6.8 |
| Magnolia    | poasana       |   24.1 |     13 |   18.6 |
| Myrsine     | floridana     |     29 |      4 |   10.6 |
| Ocotea      | sp.1          |     15 |    2.5 |    6.7 |
| Oreopanax   | nicaraguensis |    9.3 |    3.2 |    5.6 |
| Palicourea  | salicifolia   |    8.2 |    2.2 |    4.4 |
| Persea      | vesticula     |      8 |      8 |    8.0 |
| Prunus      | annularis     |   12.3 |    2.5 |    5.7 |
| Quercus     | copeyensis    |  128.5 |    2.5 |   19.2 |
| Quercus     | costaricensis |     76 |    3.4 |   25.5 |
| Styrax      | argenteus     |     20 |      0 |    5.7 |
| Vaccinium   | consanguineum |   10.4 |    2.9 |    5.5 |
| Viburnum    | costaricanum  |    9.5 |    2.6 |    6.0 |
| Zanthoxylum | melanostictum |    6.4 |    2.6 |    3.6 |
+-------------+---------------+--------+--------+--------+
19 rows in set (0.02 sec)
```

Interesting the 0 cm stem size for Styrax. Who knows how that crept in there?

Aggregate values as filter criteria: the HAVING clause

What if you wanted a list only of species with stems >= 10 cm dbh? For that, you need another clause, HAVING:

```
mysql> SELECT Genus, Species, MAX(dbh) as maxDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE p.plotCode='CMPAC'
    -> GROUP BY Genus, Species
    -> HAVING maxDBH<10;
+-------------+---------------+--------+
| Genus       | Species       | maxDBH |
+-------------+---------------+--------+
| Anthurium   | concinnatum   |    3.2 |
| Conostegia  | oerstediana   |    3.8 |
| Drimys      | granadensis   |    6.3 |
| Ilex        | lamprophylla  |    5.3 |
| Oreopanax   | nicaraguensis |    9.3 |
| Palicourea  | salicifolia   |    8.2 |
| Persea      | vesticula     |      8 |
| Viburnum    | costaricanum  |    9.5 |
| Zanthoxylum | melanostictum |    6.4 |
+-------------+---------------+--------+
9 rows in set (0.02 sec)
```

Why not put maxDBH in the WHERE clause? That's because it is a part of an aggregate function. Value calculated using aggregate functions go in a HAVING clause at the very end; everything else goes in WHERE.

The CONCAT function

The CONCAT and CONCAT_WS functions are used to combine text fields. For example,

```
mysql> SELECT DISTINCT Family, CONCAT_WS(' ',Genus, Species) as
genusSpecies
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='CM-1';
+-----------------+--------------------------+
| Family          | genusSpecies             |
+-----------------+--------------------------+
| Myrsinaceae     | Myrsine floridana        |
| Rubiaceae       | Palicourea salicifolia   |
| Aquifoliaceae   | Ilex pallida             |
| Styracaceae     | Styrax argenteus         |
| Fagaceae        | Quercus copeyensis       |
| Cornaceae       | Cornus disciflora        |
| Araceae         | Anthurium concinnatum    |
| Rosaceae        | Prunus annularis         |
| Theaceae        | Cleyera theaeoides       |
| Ericaceae       | Vaccinium consanguineum  |
| Clusiaceae      | Clusia stenophylla       |
| Magnoliaceae    | Magnolia poasana         |
| Fagaceae        | Quercus costaricensis    |
| Myrsinaceae     | Myrsine coriacea         |
| Rutaceae        | Zanthoxylum melanostictum |
| Myrsinaceae     | Myrsine sp.1              |
| Cunoniaceae     | Weinmannia pinnata        |
| Lauraceae       | Persea vesticula          |
| Caprifoliaceae  | Viburnum costaricanum     |
| Araliaceae      | Oreopanax nicaraguensis   |
| Lauraceae       | Ocotea pittieri           |
| Lauraceae       | Nectandra sp.1            |
| Melastomataceae | Miconia tonduzii          |
| Theaceae        | Symplococarpon purpusii   |
+-----------------+---------------------------+
24 rows in set (0.00 sec)
```

CONCAT_WS means concat-with-separator, and take the separator (in this case a blank space) as the first argument.

Using just the CONCAT function:

```
CONCAT(Genus, Species) as genusSpecies
```

would have produced the following result:

```
+-----------------+---------------------------+
| Family          | genusSpecies              |
+-----------------+---------------------------+
| Myrsinaceae     | Myrsinefloridana          |
| Rubiaceae       | Palicoureasalicifolia     |
| Aquifoliaceae   | Ilexpallida               |
```

etc.

The IFNULL function

Be very careful with CONCAT and CONCAT_WS! They skip records with NULL values in any of the columns used in the function. If you suspect that any of your values might be null, wrap that column in an IFNULL function. IFNULL returns an empty string ('’), which can be concatenated, if the value is null. For example, a safer version of the above examples would be:

```
CONCAT_WS(' ',IFNULL(Genus,’’), IFNULL(Species,’’)) as
genusSpecies
```

ORDER BY

You can sort fields by adding an order by clause:

```
mysql> SELECT DISTINCT Family, CONCAT_WS(' ',Genus, Species) as
genusSpecies
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='CM-1'
    -> ORDER BY Family, genusSpecies;
+-----------------+---------------------------+
| Family          | genusSpecies              |
+-----------------+---------------------------+
| Aquifoliaceae   | Ilex pallida              |
| Araceae         | Anthurium concinnatum     |
| Araliaceae       | Oreopanax nicaraguensis   |
| Caprifoliaceae   | Viburnum costaricanum     |
| Clusiaceae       | Clusia stenophylla        |
| Cornaceae        | Cornus disciflora         |
| Cunoniaceae      | Weinmannia pinnata        |
| Ericaceae        | Vaccinium consanguineum   |
| Fagaceae         | Quercus copeyensis        |
| Fagaceae         | Quercus costaricensis     |
| Lauraceae        | Nectandra sp.1            |
| Lauraceae        | Ocotea pittieri           |
| Lauraceae        | Persea vesticula          |
| Magnoliaceae     | Magnolia poasana          |
| Melastomataceae  | Miconia tonduzii          |
| Myrsinaceae      | Myrsine coriacea          |
| Myrsinaceae      | Myrsine floridana         |
| Myrsinaceae      | Myrsine sp.1              |
| Rosaceae         | Prunus annularis          |
| Rubiaceae        | Palicourea salicifolia    |
| Rutaceae         | Zanthoxylum melanostictum |
| Styracaceae      | Styrax argenteus          |
| Theaceae         | Cleyera theaeoides        |
| Theaceae         | Symplococarpon purpusii   |
+-----------------+---------------------------+
24 rows in set (0.00 sec)
```

You can also sort descending:

```
mysql> SELECT DISTINCT Family, CONCAT_WS(' ',Genus, Species) as
genusSpecies
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='CM-1'
    -> ORDER BY Family DESC, genusSpecies DESC;
+-----------------+---------------------------+
| Family          | genusSpecies              |
+-----------------+---------------------------+
| Theaceae        | Symplococarpon purpusii   |
| Theaceae        | Cleyera theaeoides        |
| Styracaceae     | Styrax argenteus          |
| Rutaceae        | Zanthoxylum melanostictum |
| Rubiaceae       | Palicourea salicifolia    |
| Rosaceae        | Prunus annularis          |
| Myrsinaceae     | Myrsine sp.1              |
| Myrsinaceae     | Myrsine floridana         |
| Myrsinaceae     | Myrsine coriacea          |
| Melastomataceae | Miconia tonduzii          |
| Magnoliaceae    | Magnolia poasana          |
| Lauraceae       | Persea vesticula          |
| Lauraceae       | Ocotea pittieri           |
| Lauraceae       | Nectandra sp.1            |
| Fagaceae        | Quercus costaricensis     |
| Fagaceae        | Quercus copeyensis        |
| Ericaceae       | Vaccinium consanguineum   |
| Cunoniaceae     | Weinmannia pinnata        |
| Cornaceae       | Cornus disciflora         |
| Clusiaceae      | Clusia stenophylla        |
| Caprifoliaceae  | Viburnum costaricanum     |
| Araliaceae      | Oreopanax nicaraguensis   |
| Araceae         | Anthurium concinnatum     |
| Aquifoliaceae   | Ilex pallida              |
+-----------------+---------------------------+
24 rows in set (0.00 sec)
```

INDEXES

Adding indexes on key fields or any field that appears in a WHERE clause can greatly speed up queries. This is important if you are querying big tables, doing queries with joins across many tables, or looping SQL inside some other code.

Our `shared` database isn't really big enough for a convincing, but I'll give it a try. Here's a query to determine the species and size of the largest individual in each plot:

```
mysql> SELECT plotCode, Genus, Species, MAX(dbh) as maxDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> GROUP BY plotCode;
+------------+--------------+----------------+--------+
| plotCode   | Genus        | Species        | maxDBH |
+------------+--------------+----------------+--------+
| CM-1       | Myrsine      | floridana      |    113 |
| CM-2       | Quercus      | costaricensis  |     96 |
| CMPAC      | Ilex         | pallida        |  128.5 |
| ElSitio    | Ilex         | lamprophylla   |    180 |
| ElSitio2   | Ilex         | sp.1           |  130.5 |
| LS-00-9    | Tapirira     | guianensis     |  111.5 |
| LS-1       | Rinorea      | deflexiflora   |     80 |
| LS-2       | Tapirira     | myriantha      |    153 |
| MT         | Chamaedorea  | tepejilote     |    130 |
| PV-1       | Simarouba    | glauca         |     54 |
| PV-1-00    | Lonchocarpus | phaseolifolius |   57.7 |
| PV-2       | Tabebuia     | rosea          |     89 |
| PV-2B      | Tabebuia     | rosea          |     25 |
| PV-2S      | Brosimum     | alicastrum     |     89 |
| PVRiverine | Guarea       | glabra         |  103.5 |
+------------+--------------+----------------+--------+
15 rows in set (0.03 sec)
```

Now I'll add indexes on the foreign key in plot_observations (PlotID) and in stems (PlotObsID).

First, plot_observations:

```
mysql> ALTER TABLE plot_observations
    -> ADD INDEX (PlotID);
Query OK, 2934 rows affected (0.07 sec)
Records: 2934  Duplicates: 0  Warnings: 0

mysql> DESCRIBE plot_observations;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| PlotObsID | int(11)      | NO   | PRI | NULL    |       |
| PlotID    | int(11)      | NO   | MUL | NULL    |       |
| Line      | varchar(255) | NO   |     | NULL    |       |
| Ind        | int(11)      | NO   |     | NULL    |       |
| Family     | varchar(255) | NO   |     | NULL    |       |
| Genus      | varchar(255) | NO   |     | NULL    |       |
| Species    | varchar(255) | NO   |     | NULL    |       |
| Habit      | varchar(255) | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

There's the index on PlotID. "MUL" stands for "multiple", meaning the index does not have to be unique; you can have multiple instances of the same value.

I did not need to add an index on the primary key, PlotObsID< because primary keys are indexed by definition. A primary key-type index IS required to be unique (no value occurs twice).

Now stems:

```
mysql> ALTER TABLE stems
    -> ADD INDEX (PlotObsID);
Query OK, 3423 rows affected (0.04 sec)
Records: 3423  Duplicates: 0  Warnings: 0

mysql> DESCRIBE stems;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| stem_id   | int(11) | NO   | PRI | NULL    |       |
| PlotObsID | int(11) | NO   | MUL | NULL    |       |
| dbh       | float   | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Now let's run that query again:

```
mysql> SELECT plotCode, Genus, Species, MAX(dbh) as maxDBH FROM
plots p JOIN plot_observations o JOIN stems s ON
p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID GROUP BY plotCode;
+------------+-------------+----------------+--------+
| plotCode   | Genus       | Species        | maxDBH |
+------------+-------------+----------------+--------+
| CM-1       | Myrsine     | floridana      |    113 |
| CM-2       | Quercus     | costaricensis  |     96 |
| CMPAC      | Ilex        | pallida        |  128.5 |
| ElSitio    | Ilex        | lamprophylla   |    180 |
| ElSitio2   | Ilex        | sp.1           |  130.5 |
| LS-00-9    | Tapirira    | guianensis     |  111.5 |
| LS-1       | Rinorea     | deflexiflora   |     80 |
| LS-2       | Tapirira    | myriantha      |    153 |
| MT         | Chamaedorea | tepejilote     |    130 |
| PV-1       | Simarouba   | glauca         |     54 |
| PV-1-00    | Lonchocarpus| phaseolifolius |   57.7 |
| PV-2       | Tabebuia    | rosea          |     89 |
| PV-2B      | Tabebuia    | rosea          |     25 |
| PV-2S      | Brosimum    | alicastrum     |     89 |
| PVRiverine | Guarea      | glabra         |  103.5 |
+------------+-------------+----------------+--------+
15 rows in set (0.02 sec)
```

Oh well. Faster, but not dramatically so. But take my word for it, if these table were huge you would want the index. Also, if you were searching ofter on certain fields, it would pay to index them them. For example if you did a lot of queries with the criterion

WHERE Family='SomeFamilyAceae'

You would want to index family.

Indexes can increase speed by orders of magnitude on big tables.

LEFT and RIGHT JOINS

Let's use different types of joins to look at overlap between different subsets of data. So far you've only had the (INNER) JOIN. Now you're going to learn LEFT and RIGHT JOINS. These are collectively called OUTER joins.

Here's a list of families found in PV-1, in tropical dry forest:

```
mysql> SELECT DISTINCT Family as dryForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='PV-1';
+-----------------+
| dryForestFamily |
+-----------------+
| Simaroubaceae   |
| Fabaceae        |
| Euphorbiaceae   |
| Theophrastaceae |
| Sapindaceae     |
| Malpighiaceae   |
| Bignoniaceae    |
| Rubiaceae       |
| Sapotaceae      |
| Anacardiaceae   |
| Flacourtiaceae  |
| Verbenaceae     |
| Combretaceae    |
| Tiliaceae       |
| Myrtaceae       |
| Annonaceae      |
| Hippocrateaceae |
| Erythroxylaceae |
| Menispermaceae  |
| Ebenaceae       |
| Trigoniaceae    |
| Cecropiaceae    |
| Boraginaceae    |
| Nyctaginaceae   |
| Sterculiaceae   |
| Olacaceae       |
| Apocynaceae     |
+-----------------+
27 rows in set (0.01 sec)
```

Note the use of the descriptive alias 'dryForestFamily' to make the table a bit more readable.

Now, here's a list of families found in LS-1, a plot in Tropical Rain Forest:

```
mysql> SELECT DISTINCT Family as rainForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='LS-1';
+------------------+
| rainForestFamily |
+------------------+
| Violaceae        |
| Arecaceae        |
| Dichapetalaceae  |
| Araliaceae       |
| Sapindaceae      |
| Fabaceae         |
| Capparaceae      |
| Malpighiaceae    |
| Lecythidaceae    |
| Meliaceae        |
| Cecropiaceae     |
| Annonaceae       |
| Burseraceae      |
| Rubiaceae        |
| Moraceae         |
| Monimiaceae      |
| Lauraceae        |
| Boraginaceae     |
| Melastomataceae  |
| Myristicaceae    |
| Bombacaceae      |
| Ulmaceae         |
| Sapotaceae       |
| Rhamnaceae       |
| Rhizophoraceae   |
| Olacaceae        |
| Solanaceae       |
| Lacistemataceae  |
| Malvaceae        |
| Dilleniaceae     |
| Sabiaceae        |
| Acanthaceae      |
| Apocynaceae      |
| Myrtaceae        |
+------------------+
34 rows in set (0.00 sec)
```

Using a regular (INNER) JOIN, the one we've been using all along, we can get a list of shared families:

```
mysql> SELECT dryForestFamily, rainForestFamily
    -> FROM
    -> (
    -> SELECT DISTINCT Family as dryForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='PV-1'
    -> ) AS df
    -> JOIN
    -> (
    -> SELECT DISTINCT Family as rainForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='LS-1'
    -> ) AS rf
    -> ON dryForestFamily=rainForestFamily;
+-----------------+------------------+
| dryForestFamily | rainForestFamily |
+-----------------+------------------+
| Sapindaceae     | Sapindaceae      |
| Fabaceae        | Fabaceae         |
| Malpighiaceae   | Malpighiaceae    |
| Cecropiaceae    | Cecropiaceae     |
| Annonaceae      | Annonaceae       |
| Rubiaceae       | Rubiaceae        |
| Boraginaceae    | Boraginaceae     |
| Sapotaceae      | Sapotaceae       |
| Olacaceae       | Olacaceae        |
| Apocynaceae     | Apocynaceae      |
| Myrtaceae       | Myrtaceae        |
+-----------------+------------------+
11 rows in set (0.00 sec)
```

Now, by specifying a LEFT JOIN instead of an (INNER) JOIN, you get a list of all records in the table (or in the case, the nested query) preceding the LEFT JOIN statement, and only the shared records from the table/query following LEFT JOIN statement.

In this case, we get a list of all families in dry forest on the left, and families shared between dry and rainforest on the right.

```
mysql> SELECT dryForestFamily, rainForestFamily
    -> FROM
    -> (
    -> SELECT DISTINCT Family as dryForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='PV-1'
    -> ) AS df
    -> LEFT JOIN
    -> (
    -> SELECT DISTINCT Family as rainForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='LS-1'
    -> ) AS rf
    -> ON dryForestFamily=rainForestFamily;
+-----------------+------------------+
| dryForestFamily | rainForestFamily |
+-----------------+------------------+
| Simaroubaceae   | NULL            |
| Fabaceae        | Fabaceae        |
| Euphorbiaceae   | NULL            |
| Theophrastaceae | NULL            |
| Sapindaceae     | Sapindaceae     |
| Malpighiaceae   | Malpighiaceae   |
| Bignoniaceae    | NULL            |
| Rubiaceae       | Rubiaceae       |
| Sapotaceae      | Sapotaceae      |
| Anacardiaceae   | NULL            |
| Flacourtiaceae  | NULL            |
| Verbenaceae     | NULL            |
| Combretaceae    | NULL            |
| Tiliaceae       | NULL            |
| Myrtaceae       | Myrtaceae       |
| Annonaceae      | Annonaceae      |
| Hippocrateaceae | NULL            |
| Erythroxylaceae | NULL            |
| Menispermaceae  | NULL            |
| Ebenaceae       | NULL            |
| Trigoniaceae    | NULL            |
| Cecropiaceae    | Cecropiaceae    |
| Boraginaceae    | Boraginaceae    |
| Nyctaginaceae   | NULL            |
| Sterculiaceae   | NULL            |
| Olacaceae       | Olacaceae       |
| Apocynaceae     | Apocynaceae     |
+-----------------+-----------------+
27 rows in set (0.01 sec)
```

To produce a list of families only found in dryforest and not in rainforest, exclude the shared families with a where clause (and, if you wish, omit the second column for clarity):

```
mysql> SELECT dryForestFamily
    -> FROM
    -> (
    -> SELECT DISTINCT Family as dryForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='PV-1'
    -> ) AS df
    -> LEFT JOIN
    -> (
    -> SELECT DISTINCT Family as rainForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='LS-1'
    -> ) AS rf
    -> ON dryForestFamily=rainForestFamily
    -> WHERE rainForestFamily IS NULL;
+-----------------+
| dryForestFamily |
+-----------------+
| Simaroubaceae   |
| Euphorbiaceae   |
| Theophrastaceae |
| Bignoniaceae    |
| Anacardiaceae   |
| Flacourtiaceae  |
| Verbenaceae     |
| Combretaceae    |
| Tiliaceae       |
| Hippocrateaceae |
| Erythroxylaceae |
| Menispermaceae  |
| Ebenaceae       |
| Trigoniaceae    |
| Nyctaginaceae   |
| Sterculiaceae   |
+-----------------+
16 rows in set (0.00 sec)
```

Conversely, you can use a RIGHT JOIN to get a list of families found only in rain forest:

```
mysql> SELECT rainForestFamily
    -> FROM
    -> (
    -> SELECT DISTINCT Family as dryForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='PV-1'
    -> ) AS df
    -> RIGHT JOIN
    -> (
    -> SELECT DISTINCT Family as rainForestFamily
    -> FROM plots p JOIN plot_observations o
    -> ON p.PlotID=o.PlotID
    -> WHERE plotCode='LS-1'
    -> ) AS rf
    -> ON dryForestFamily=rainForestFamily
    -> WHERE dryForestFamily IS NULL;
+------------------+
| rainForestFamily |
+------------------+
| Violaceae        |
| Arecaceae        |
| Dichapetalaceae  |
| Araliaceae       |
| Capparaceae      |
| Lecythidaceae    |
| Meliaceae        |
| Burseraceae      |
| Moraceae         |
| Monimiaceae      |
| Lauraceae        |
| Melastomataceae  |
| Myristicaceae    |
| Bombacaceae      |
| Ulmaceae         |
| Rhamnaceae       |
| Rhizophoraceae   |
| Solanaceae       |
| Lacistemataceae  |
| Malvaceae        |
| Dilleniaceae     |
| Sabiaceae        |
| Acanthaceae      |
+------------------+
23 rows in set (0.01 sec)
```

Here's an example of a more complex query using outer joins.

The query produces a list of species of Melastomataceae which never get bigger than 10 cm across all the plots. It works by selecting all individuals whose largest stem is <10 cm dbh (the subquery 'smallInd'), makes a list of those species (subquery 'smallSpp') and compares it with a similar list of species having individuals with one or more stems>10 cm dbh (subqueries 'bigInd' and 'bigSpp'). A LEFT join produces the final list of species never reaching the 10 cm dbh size class.

```
mysql> SELECT smallSpecies
    -> FROM
    -> (
    -> SELECT DISTINCT genusSpecies as smallSpecies
    -> FROM
    -> (
    -> SELECT o.PlotObsID, CONCAT_WS(' ',Genus, Species) as
genusSpecies, MAX(dbh) AS maxDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE Family='Melastomataceae'
    -> GROUP BY o.PlotObsID
    -> HAVING maxDBH<10
    -> ) AS smallInd
    -> ) AS smallSpp
    -> LEFT JOIN
    -> (
    -> SELECT DISTINCT genusSpecies as bigSpecies
    -> FROM
    -> (
    -> SELECT o.PlotObsID, CONCAT_WS(' ',Genus, Species) as
genusSpecies, MAX(dbh) AS maxDBH
    -> FROM plots p JOIN plot_observations o JOIN stems s
    -> ON p.PlotID=o.PlotID AND o.PlotObsID=s.PlotObsID
    -> WHERE Family='Melastomataceae'
    -> GROUP BY o.PlotObsID
    -> HAVING maxDBH>10
    -> ) AS bigInd
    -> ) AS bigSpp
    -> ON smallSpecies=bigSpecies
    -> WHERE bigSpecies IS NULL;
+-----------------------+
| smallSpecies          |
+-----------------------+
| Miconia gracilis      |
| Conostegia bracteata  |
| Conostegia montana    |
| Henriettea tuberculosa |
| Miconia dorsiloba     |
| Miconia elata         |
| Miconia smaragdina    |
| Miconia tonduzii       |
| Leandra sp.1           |
| Conostegia oerstediana |
| Miconia sp.2           |
| Miconia sp.3           |
| Conostegia rufescens   |
| Conostegia sp.1        |
| Meriania phlomoides    |
| Ossaea sp.1            |
| Ossaea micrantha       |
| Miconia                |
+------------------------+
18 rows in set (0.04 sec)
```

LEFT snd RIGHT JOINs are useful, but they are also VERY slow with big tables. Use with discretion.

APPENDIX: A (VERY) BASIC UNIX TUTORIAL

The goal of this manual isn't to introduce you to the Unix operating system. But those of you opting to shell into the server instead of using a GUI will need to know a few basics. I'll cover those here. (Technically, for those of you who care or know, we are using the bash shell in the example below, one particular interface for interacting with the operating system; there are others).

When I say basic, I mean basic. If you plan on using unix at all (and you should; it's powerful), you at least read the concise little booklet, "Learning the Unix Operating System: A Concise Guide for the New User".

http://shop.oreilly.com/product/9780596002619.do

There's a copy in the lab. It's the skinny paperback with the owl on the cover.

Want more? There are also several thicker, more complete, and more intimidating unix books in the lab. Or buy your own at the bookstore or on Amazon. O'Reilly software books are good. Of course, you should be able to self-teach on a need-to-know basis using google.

Have a question about a particular command? Unix comes with a built-in manual for all its commands. Just enter "man [command-name]", for example "$ man cd".

JoeUser's first unix session

Follow along with JoeUser and you should be able to learn everything he knows, which is not much. Note that what JoeUser enters is in **bold**, my comments are in *italics*, and the server's responses are in regular font.

By default, when you log in, you will be in your home directory, which lives under the directory, guess, what, "home". The terminal will display a prompt ($) and wait for you to enter something. Here's what JoeUser sees when he logs in.

```
Bradley-Boyles-MacBook-Pro:~ bboyle$ ssh JoeUser@128.196.193.245
JoeUser@128.196.193.245's password:
Last login: Mon Nov 21 10:50:59 2011 from 128.196.198.69
 [JoeUser@salvias ~]$
```
- *Logged in. The server is ready and waiting for Joe to enter something*

```
[JoeUser@salvias ~]$ pwd
/home/JoeUser
```
- *"pwd"="print working directory", also, "print current directory". Now Joe knows where he is.*

```
[JoeUser@salvias ~]$ ls
```
- *"ls"="list" or, get a list of what's in the current directory directory. At the moment, there is nothing, so the terminal returns nothing*

```
[JoeUser@salvias ~]$ mkdir test
```
- *"mkdir"="make directory [directory name]". In this case, make a new directory called "test"*

```
[JoeUser@salvias ~]$ ls
```

```
test
```
- there's the new directory

```
[JoeUser@salvias ~]$ cd test
```
- "cd"="change directory"

```
[JoeUser@salvias test]$ ls
```
- again, nothing in the new directory "test"

```
[JoeUser@salvias test]$ touch atestfile.txt
```
- "touch [filename]" – create a new file. Not normally a useful thing to do, as you would more likely drag files from your local machine to the server, or from the server to your local machine. But bear with me.

```
[JoeUser@salvias test]$ ls
atestfile.txt
```
- there's the new file

```
[JoeUser@salvias test]$ echo This is the first line >> atestfile.txt
```
- "echo"="echo to the terminal", BUT, the ">>" means, "take the output on the left and put it into the object on the right. In this case, the new file. "atestfile.txt". Again, not super useful on it's own, but we'll put a few lines into the file so I can show you something more useful.

```
[JoeUser@salvias test]$ echo This is the second line >> atestfile.txt
[JoeUser@salvias test]$ echo And this is the third line >> atestfile.txt
```
- adding a couple more lines

```
[JoeUser@salvias test]$ more atestfile.txt
This is the first line
This is the second line
And this is the third line
```
- "more [filename]"=display the contents of file [filename]. If the file is very big, it will display a screenful at a time. Use the space bar to toggle from screenful to screenful. If you don't feel like toggling all the way to the end, press <ctrl>C to abort.

```
[JoeUser@salvias test]$ head -2 atestfile.txt
This is the first line
This is the second line
```
- "head -[linesToDisplay] [filename]" – displays the top few lines of the file, as specified by whatever number you enter after the "-". Useful for getting a glimpse of what's in a really big file.

```
[JoeUser@salvias test]$ cp atestfile.txt acopyofatestfile.txt
```
- "cp"="copy [oldfile] [newfile]" – makes a copy of oldfile and calls it newfile
```
[JoeUser@salvias test]$ ls
acopyofatestfile.txt  atestfile.txt
[JoeUser@salvias test]$ ls -l
-rw-rw-r-- 1 JoeUser JoeUser 74 2011-11-21 10:59 acopyofatestfile.txt
-rw-rw-r-- 1 JoeUser JoeUser 74 2011-11-21 10:58 atestfile.txt
```
- "ls –l"="list long format". There's a lot going on here that I won't go into, but basically, for each item in the directory, it lists permissions (all that stuff on the left , e.g., 'rw-rw-r—'), the

owner (JoeUser), the group to which the file belongs (group JoeUser), the size of the file (in this case, 74 bytes; these are tiny files), the date & time the file was created, and the file name.

```
[JoeUser@salvias test]$ more acopyof<tab>
```

- hate typing stupid long file names? Enter part of the name, press the tab key (<tab>) and unix will fill in the rest for you.

```
[JoeUser@salvias test]$ more acopyofatestfile.txt
This is the first line
This is the second line
And this is the third line
```

```
[JoeUser@salvias test]$ mv acopyofatestfile.txt testfile2.txt
[JoeUser@salvias test]$ ls
atestfile.txt  testfile2.txt
```

- "mv"="move [oldfile] [newfile]". Basically, this mean "rename oldfile to newfile". But you can also use it to move a file from one directory to another. Watch this:

```
[JoeUser@salvias test]$ mv testfile2.txt ../
```

- this means, "move testfile2.txt" up one directory. That's what the "../" means. Now let's see where the file went:

```
[JoeUser@salvias test]$ pwd
/home/JoeUser/test
[JoeUser@salvias test]$ ls
atestfile.txt
```

- gone from the current directory!

```
[JoeUser@salvias test]$ cd ..
```

- this means "go up one directory"

```
[JoeUser@salvias ~]$ pwd
/home/JoeUser
[JoeUser@salvias ~]$ ls
test  testfile2.txt
```

- there it is!

```
[JoeUser@salvias ~]$ rm testfile2.txt
```

- "rm [filename]"="remove [filename]"

```
[JoeUser@salvias ~]$ ls
test
```

- gone!

```
[JoeUser@salvias ~]$ cp -r test test2
```

- this means, "copy recursively directory1 to directory2". Why recursive ("-r")? Because you need to tell unix to copy not just the directory, but everything in it, including any files and subdirectories.

```
[JoeUser@salvias ~]$ ls
test  test2
```

- there's the new directory, a copy of "test". Let's see what's in it:

```
[JoeUser@salvias ~]$ cd test2
[JoeUser@salvias test2]$ ls
atestfile.txt
```
- same as directory "test", as it should be. Now, let's get out and remove this directory:

```
[JoeUser@salvias test2]$ cd ..
[JoeUser@salvias ~]$ pwd
/home/JoeUser
[JoeUser@salvias ~]$ ls
test  test2
[JoeUser@salvias ~]$ rm -r test2
```
- note the use of the recursive switch "-r" after "rm"

```
[JoeUser@salvias ~]$ ls
test
```
- gone! As you can see, "rm" is dangerous. OK, couple more things and we're done.

```
[JoeUser@salvias ~]$ cd /
```
- This means, change directory all the way to the root directory.

```
[JoeUser@salvias /]$ ls
bin   dev  floppy  lib         media  opt   root  selinux  sys   tmp  var
boot  etc  home    lost+found  mnt    proc  sbin  srv      temp  usr
```
- there's a ton of stuff you shouldn't mess with. In most cases, unix permissions will prevent you from screwing anything up. However, there is one directory you do have access to and might find useful:

```
[JoeUser@salvias /]$ cd temp
[JoeUser@salvias temp]$ ls
plot_locations.csv
```
- Remember temp? 'temp' is a directory readable and writeable by anyone. Use it to dump mysql output if you are using the OUTFILE command (see the section about **OUTFILE** in "**GETTING DATA OUT**"). From here, you can either use "mv" or "cp" to put the file in your home directory, or you can use a GUI such as SCP or Cyberduck to drag the file to your local desktop. Now, let's try snooping in one more place before you go home:

```
[JoeUser@salvias lib]$ cd ..
```
- back in root

```
[JoeUser@salvias /]$ ls
bin   dev  floppy  lib         media  opt   root  selinux  sys   tmp  var
boot  etc  home    lost+found  mnt    proc  sbin  srv      temp  usr
 [JoeUser@salvias /]$ cd home
[JoeUser@salvias home]$ ls
bboyle  brian  eebadmin  JoeUser  llsloat  tytaylor  vbuzzard
 [JoeUser@salvias home]$ cd bboyle
-bash: cd: bboyle: Permission denied
```
- bboyle's directory is readable only by bboyle. Your own directory is similarly protected from snooping, reading or deleting by others. But JoeUser owns his own directory, so he can navigate to there:

```
[JoeUser@salvias home]$ **cd JoeUser**
[JoeUser@salvias ~]$ **pwd**
/home/JoeUser
```

Finally, here's how you log out:

```
[JoeUser@salvias ~]$ **exit**
logout
Connection to 128.196.193.245 closed.
```

‘Ecoinformatics’ Basics
BUILDING UNIX skills Part II

In accessing your account I assume that you are proficient in using ssh. You will need ssh as a way to view the unix prompt for your account. See above for more introduction to ssh. I also you assume how to transfer files via sftp.

<u>Listing files</u>

```
ls command

ls -a

ls -l

ls -F

ls -a -l
```

<u>Viewing files</u>

more *filename*

<u>Learning more about any unix command</u>

Consult the manual (or man).

man *unix_command_in question*

<u>Your Shell</u>

```
Echo $SHELL
```

Most terminals are setup to use the t shell. So you should receive output that looks like

```
/bin/tcsh
```

within the mac osX many have a bash shell where the output would look like

```
/bin/bash
```

<u>Modify your shell</u>

The power of the unix shell is that you can create an alias so that you don’t have to type so much!

You first need to view your dot files and then view your .tcshrc file. Type

```
more .tcshcr
```

you may see something like this . . .

```
#Applications acroread=adobeAcrobat arcinfo=ArcInfo arc=ARC
xmgr=xmgr
pico=pico pine=pine
#netscape=Netscape

source /home/cshrc

set path = (. $sys_path /usr/sbin /usr/bin /usr/dt/bin
/usr/openwin/bin/bin/usr/ucb)
set history = 100 notify noclobber


#lists directories in columns
alias ls 'ls -F -C'

#correction for inability to ssh newu.u.arizona.edu
alias newu 'ssh benquist@128.196.133.224'

#rm prompt for removal of files
alias rm rm -i

#Turn on immediate notification of job completion
set notify


#Shell corrections for spelling


#set prompt

if ( (! $?ENVONLY) && $?prompt ) then
        if ( -o /bin/su ) then
                        set prompt="`hostname -s` \!# "
        else
                        set prompt="`hostname -s` \!% "
        endif
endif
```

How do I change the permissions of my files?

chmod 200	*filename*	Owner has write permission
chmod 400	*filename*	Owner has read permission
chmod 100	*filename*	Owner has execute permission
chmod 040	*filename*	Group has read permission
chmod 004	*filename*	World has read permission

chmod 744 *filename* Owner can read, write and execute a file

Change ownership

chown *newowner filename*

chown Gentry .tcshrc

How do I edit my files?

Use the text editors available on the system – usually these are pico and vi

pico *filename*

How do I move around my files to other directories?

mv command
cp command

Speeding up your efficiency – use the wildcard! '*'

Use also the wildcard * to list out difficult to find files or view files!

Data compression

compress *filename*

uncompress *filename.*Z

How do I Learn more? Just like in R, in unix you can use the 'man' command to read the unix manual for each command.

You can learn more on each of these commands by typing 'man' which is short for manual. Just like R you can learn more about different commands

```
68% man grep
```

Data manipulation in UNIX

Redirection of command input and output

Command original_file > *output_file*

Or

Command specific_options < *inputfile* > *output_file*

wc command

wc *filename*

prints the number of lines, words, characters, and lines in a text file { -c only the number of characters, -l only the number of lines, -w only the number of words]

The cat, paste, merge, split commands

cat *filenames* > new_outputfile
paste *filenames* > new_outputfile

See also the following unix commands

split
sort
join
cut

Search and replace . . . How do I remove tabs, commas and replace them with spaces? (or vica versa?)

tr command

tr *options* < *filenameinput* > *filenameoutput*

Often .txt files from excel or other outputs will have odd characters in them (tabs, spaces etc.)

```
25% ls –s

total 6488  752 CtoG.txt        704 NtoT.txt        816 a-to-f.txt
            912 l-to-o.txt      24 A-to-B.txt.Z    816 HtoM.txt        640 UtoZ.txt
           1200 g-to-k.txt      624 p-to-s.txt
```

Let's remove any tabs within the file

```
26% tr '        ' ' ' < CtoG.txt > CtoG.txt2
```

Here we first type 'CTRL v' followed by hitting the [tab] button. The second region in ' ' is actually a space. So we are telling the tr command to find all tabs and replace them with a space.

Now if we wanted to insert commas between data enteries (or replace all spaces with commas) we type

```
26% tr ' ' ',' < CtoG.txt > CtoG.txt2
```

```
27% ls -s
   total 7208         720 CtoG.txt2      640 UtoZ.txt       912 l-to-o.txt
        24 A-to-B.txt.Z   816 HtoM.txt       816 a-to-f.txt     624 p-to-s.txt
        752 CtoG.txt      704 NtoT.txt      1200 g-to-k.txt
```

<u>Start writing unix scripts</u>

<u>foreach command</u>

A fun way to automate UNIX commands and

```
55% foreach filename ( *.txt )
foreach? tr '  ''' < $filename > $filename.out
foreach? end
```

```
56% ls –s

total 12784          816 HtoM.txt         624 UtoZ.txt.out     912 l-to-o.txt
  56 A-to-B.txt        784 HtoM.txt.out      816 a-to-f.txt       880 l-to-o.txt.out
  56 A-to-B.txt.out    704 NtoT.txt         784 a-to-f.txt.out   624 p-to-s.txt
 752 CtoG.txt          672 NtoT.txt.out    1200 g-to-k.txt       592 p-to-s.txt.out
 720 CtoG.txt.out      640 UtoZ.txt        1152 g-to-k.txt.out
```

```
57% cat *.out > Master.dat

58% ls

59% ls -s
total 18896          816 HtoM.txt         640 UtoZ.txt        1152 g-to-k.txt.out
  56 A-to-B.txt       784 HtoM.txt.out     624 UtoZ.txt.out     912 l-to-o.txt
  56 A-to-B.txt.out  6112 Master.dat       816 a-to-f.txt       880 l-to-o.txt.out
 752 CtoG.txt         704 NtoT.txt         784 a-to-f.txt.out   624 p-to-s.txt
 720 CtoG.txt.out     672 NtoT.txt.out    1200 g-to-k.txt       592 p-to-s.txt.out
```

Let's use the foreach command to construct a new dataset with the 'abundance' counts for each of our .txt files.

```
63% foreach filename ( *.txt )
foreach? wc -l $filename > $filename.count
foreach? end

64% ls -s
total 18914             2 HtoM.txt.count     624 UtoZ.txt.out         2 l-to-o.txt.count
  56 A-to-B.txt         784 HtoM.txt.out       816 a-to-f.txt         880 l-to-o.txt.out
   2 A-to-B.txt.count  6112 Master.dat           2 a-to-f.txt.count   624 p-to-s.txt
  56 A-to-B.txt.out     704 NtoT.txt           784 a-to-f.txt.out       2 p-to-s.txt.count
 752 CtoG.txt             2 NtoT.txt.count    1200 g-to-k.txt         592 p-to-s.txt.out
   2 CtoG.txt.count     672 NtoT.txt.out         2 g-to-k.txt.count
 720 CtoG.txt.out       640 UtoZ.txt          1152 g-to-k.txt.out
 816 HtoM.txt             2 UtoZ.txt.count     912 l-to-o.txt

65% ls *.cout
ls: No match.

66% ls *.count
A-to-B.txt.count   HtoM.txt.count    UtoZ.txt.count    g-to-k.txt.count   p-to-s.txt.count
CtoG.txt.count     NtoT.txt.count    a-to-f.txt.count  l-to-o.txt.count

67% cat *.count > Master_count.dat

68% more Master_count.dat
    1063 A-to-B.txt
   13993 CtoG.txt
   15206 HtoM.txt
   13041 NtoT.txt
   11917 UtoZ.txt
   15110 a-to-f.txt
   22384 g-to-k.txt
   16917 l-to-o.txt
   11404 p-to-s.txt
```

Interesting in learning more about powerful unix commands?
Check out the following commands

```
cat
paste
merge
grep
sed
awk
```

grep, sed, and awk are powerful commands for regular expressions and searching and pulling out certain data

Learning more?

You can learn more on each of these commands by typing 'man' which is short for manual. Just like R you can learn more about different commands

```
68% man grep

REP(1)              BSD General Commands Manual              GREP(1)

NAME
    grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
    grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]] [-e pattern]
        [-f file] [--binary-files=value] [--color[=when]] [--colour[=when]]
        [--context[=num]] [--label] [--line-buffered] [--null] [pattern] [file ...]

DESCRIPTION
    The grep utility searches any given input files, selecting lines that match one or
    more patterns.  By default, a pattern matches an input line if the regular expression
    (RE) in the pattern matches the input line without its trailing newline.  An empty
    expression matches every line.  Each input line that matches at least one of the pat-
    terns is written to the standard output.

    grep is used for simple patterns and basic regular expressions (BREs); egrep can han-
    dle extended regular expressions (EREs).  See re_format(7) for more information on
    regular expressions.  fgrep is quicker than both grep and egrep, but can only handle
    fixed patterns (i.e. it does not interpret regular expressions).  Patterns may consist
    of one or
```