MySQL

David J. Scott

d.scott@auckland.ac.nz

Department of Statistics, University of Auckland

Outline

- Introduction
- Tutorial
- Practicalities
- Data types
- Creating, selecting and dropping databases
- Creating, altering, and dropping tables
- Retrieving data
- MySQL and **R**: the RODBC package

Resources

- The MySQL website is the major resource:
  `http://www.mysql.com/`
- There is a tutorial available on the MySQL website:
  `http://dev.mysql.com/doc/refman/5.0/en/tutorial.html/`
- The tutorial is part of the MySQL manual, which for version 5.0 is:
  `http://dev.mysql.com/doc/refman/5.0/en/index.html`
- To install and run MySQL on a windows machine, there are some instructions put together by James Curran:
  `www.stat.auckland.ac.nz/~dscott/782/workingwithdatabases.pdf`

Introduction

What is MySQL?

- It is a relational database management system
  - A database is a structured collection of data
  - A database management system is used to add, access and process data in a database
  - A relational database stores data in tables
- It uses SQL, Structured Query Language, to access data in a database
- It is Open Source: you can download and modify the software
  - There are restrictions on use of the Open Source software
  - A commercial version is available
  - MySQL is owned by for-profit company
- Contributed software is available
- Commonly used in web applications in conjunction with PHP

MySQL Basics

- Written in C and C++
- Works on many platforms, including Windows and Linux
- Scalable, used on large databases: 60,000 tables, 5,000 million rows

Relational Databases

- A database consists of one or more tables
- Each tables is organized into rows and columns
- Each row of a table is a record
- Records may contain several pieces of information: each column in a table corresponds to one of those pieces
- Relational databases strengths are:
  - Information need only be stored one time
  - Data can retrieved by joining information from different tables

Relational Databases

- Suppose you have a problem with customer orders: some goods have not arrived
- You wish to contact all customers who have ordered the item
- Your customer database contains two tables:
  - A table of customer details giving contact information, indexed by a customer ID
  - A table of orders, which includes the customer ID as one column
- If the item ordered is given in the orders table, you need both tables to prepare the information to email all customers who have ordered that item
- It would be inefficient and cause errors if the customer addresses were stored **with** the orders

Client/Server Architecture

- You use two programs when using MySQL:
  - The database server which is on the machine where the data is stored
  - Client programs which connect to the database server to modify and retrieve data
- The server controls concurrency: two users cannot modify the same record at the same time
- You don’t have to be logged into the machine where the database is located
- Secure access to the database is controlled by passwords and permissions
- Besides `mysql` which allows access to databases, there are programs `mysqldump` for backup, `mysqlimport`, and `msyqladmin`

Tutorial

Getting Started

- Need access to MySQL Server on some machine
- Need the MySQL Client running on a machine you can log on to
- If you are going to create a database and its tables you need permission to do so—it will be created on the server
- If you are the MySQL administrator, you are able to do that using a `create database` command
- Otherwise permission is granted by the MySQL administrator using a `grant` command

Getting Started

- Assume that the database has already been created
- To access it with the user name `dscott`, assuming it is located on the machine `stat71.stat.auckland.ac.nz`, the command is

```
mysql -u dscott -h stat71.stat.auckland.ac.nz -p
```

- The `-p` means `mysql` prompts for your *mysql* password
- The other options specify the user and the host

Queries

- A query is typed in and ended with a semicolon (`;`)

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2007-10-07 22:38:28 |
+---------------------+
1 row in set (0.00 sec)
```

- `now()` is a useful function, for example to find the time between the current time and another time
- Case doesn't matter in command and function names. The following are equivalent:

```
select now();
SELECT NOW();
SeLecT nOw();
```

Queries

- `mysql` is an interpreted language and waits for the semi-colon before sending the query to the server

```
mysql> select now(),
    -> user(),
    -> version()
    -> ;
+---------------------+----------------+---------------------+
| now()               | user()         | version()           |
+---------------------+----------------+---------------------+
| 2007-10-07 22:40:36 | root@localhost | 5.0.41-community-nt |
+---------------------+----------------+---------------------+
1 row in set (0.00 sec)
```

- You can ask for more than one item by separating with a comma
- Abort a query by entering `\c`

Creating the Database

- Example from Paul DuBois, *MySQL*
- Database is called `sampdb`
- Steps to create a usable database
  - Create or initialize the database
  - Create tables which comprise the database
  - Insert data into tables
- Using the database involves querying the database, modifying entries, inserting new data, deleting data etc

Creating the Database

- Create a new database from within `mysql`:

```
mysql> create database sampdb;
```

- Or from the command line

```
[dscott@stat12 dscott]> mysql sampdb
```

- Select and show the current database with the commands

```
mysql> use sampdb;
Database changed
mysql> select database();
+------------+
| database() |
+------------+
| sampdb     |
+------------+
1 row in set (0.00 sec)
```

US Historical League

- Example from Paul DuBois, *MySQL*
- An historical society
  - Information on US presidents: a `president` table
  - Information on society member: a `member` table

The `president` *Table*

- Contains the data
  - Name given by `first_name` (including middle name if available), `last_name` and `suffix` (such as Jr.)
  - Birthplace given by `city` and `state`
  - Birth and death dates given by `birth` and `death`, with `death` taking the value `null` if still alive

The `member` *Table*

- Name as for the `president` table
- Member ID as `member_id`
- Date membership expires as `expiration`
- Email address as `email`
- Postal address with columns `street`, `city`, `stat`, and `zip`
- Phone number as `phone`
- Special interest keyword as `interest`

Creating Tables

- The template for this command is

  create table *tablename* (*columnspecs*)

- To create the `president` table:

```
create table president
(
  last_name varchar(15) not null,
  first_name varcar(15) not null,
  suffix varchar(5) null,
  city varchar(20) not null,
  state varchar(2) not null,
  birth date not null,
  death date null
);
```

- Column specifications are the name, the type of data, and possibly some column attributes

The `president` *Table*

- Two data types used, `varchar` and `date`
- `varchar(`*n*`)` means the column contains variable-length character values with a maximum length of *n* characters
- `date` columns must be dates in `yyyy-mm-dd` format
- The attributes `null` and `not null` mean respectively that values can be missing or may not be missing
- The column descriptions of a table can be displayed using `describe` *`tablename`* as in

```
mysql> describe president;
mysql> describe member;
```

Inserting Data

- A small number of records may be inserted using the command

```
insert into tablename values(value1,value2, ...)
```

- For example:

```
insert into president values
(
  'Bush','George', null,'New Haven','CT', '1946-07-06',null
)
```

- Single or double quotes can be used
- More than one record can be added at a time:

```
insert into tablename
values(value1,value2, ...),
      (value1,value2, ...)
```

Inserting Data

- Alternatively data can be read in from a file

```
load data local infile 'member.txt' into table member;
```

- The default is that column values are separated by tabs, and lines end with newlines
- Values must also be in the order that the columns are stored in the table
- This can be changed:

```
load data local infile "~/OriginalData/FixDM_Extract.csv"
into table extract
fields terminated by ','
ignore 1 lines
(CallID, AgentName, Queue,MajorTag, TagText,
        . . .
VarTariff, Consumption);
```

Backing Up

- Creating a backup of your database or individual tables is a **very good idea**

```
[dscott@stat12 dscott]> mysqldump sampdb > sampdb7Oct2007
[dscott@stat12 dscott]> mysqldump sampdb | gzip > sampdb7Oct2007
[dscott@stat12 dscott]> mysqldump sampdb member president > histlea
```

- Then restore tables with

```
[dscott@stat12 dscott]> mysql < histleague.sql
```

Retrieving Information

- Select statement with modifications

```
select * from president
select birth from president where last_name = 'Eisenhower'
```

- General form is

```
select what to select
from table or tables
where conditions the data must satisfy
```

The `select` *Statement*

- `select` can do calculations, display text, access functions

```
mysql> select 2+2, 'Hello, world', version();
+-----+--------------+------------+
| 2+2 | Hello, world | version() |
+-----+--------------+------------+
|   4 | Hello, world | 5.0.27     |
+-----+--------------+------------+
1 row in set (0.03 sec)
```

The `select` Statement

- A standard use of `select` is to select from a table subject to the record satisfying a `where` clause
- Multiple columns being selected are separated by commas

```
mysql> select last_name, first_name from president
    ->   where last_name = 'Bush';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Bush      | George H.W. |
| Bush      | George W.   |
+-----------+-------------+
2 rows in set (0.00 sec)
```

The `select` Statement

- Use Boolean logic to create complex conditions
- Use brackets to ensure conditions are combined correctly

```
mysql> select last_name, first_name, birth, state from president
    ->   where birth <'1750-1-1' and (state='VA' or state='MA');
+------------+------------+------------+-------+
| last_name  | first_name | birth      | state |
+------------+------------+------------+-------+
| Washington | George     | 1732-02-22 | VA    |
| Adams      | John       | 1735-10-30 | MA    |
| Jefferson  | Thomas     | 1743-04-13 | VA    |
+------------+------------+------------+-------+
3 rows in set (0.00 sec)
```

The `select` *Statement*

- The syntax for selecting when the value is `null` is different
- `death = null` will not work

```
mysql> select last_name, first_name from president
    ->   where death is null;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Ford      | Gerald R    |
| Carter    | James E.    |
| Bush      | George H.W. |
| Clinton   | William J.  |
| Bush      | George W.   |
+-----------+-------------+
5 rows in set (0.00 sec)
```

The `select` Statement

- Output can be sorted as ascending or descending
- Default is ascending
- Ordering can be done on multiple columns

```
mysql> select last_name, first_name, state from president
    ->   order by state desc, last_name asc;
+-----------+---------------+-------+
| last_name | first_name    | state |
+-----------+---------------+-------+
| Arthur    | Chester A.    | VT    |
| Coolidge  | Calvin        | VT    |
.
.
.
| Clinton   | William J.    | AR    |
+-----------+---------------+-------+
42 rows in set (0.00 sec)
```

The `select` *Statement*

- It is useful to be able to limit the number of rows selected

```
mysql> select last_name, first_name, birth  from president
    ->   order by birth limit 5;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
5 rows in set (0.00 sec)
```

The `select` Statement

- The `concat` function concatenates text items
- The expression is used for the column heading
- The column may be very wide

```
mysql> select concat(first_name,' ',last_name),
    ->   concat(city,' ',state)
    ->   from president limit 5;
+----------------------------------+------------------------+
| concat(first_name,' ',last_name) | concat(city,' ',state) |
+----------------------------------+------------------------+
| George Washington                  | Wakefield VA             |
| John Adams                         | Braintree MA             |
| Thomas Jefferson                   | Albemarle County VA      |
| James Madison                      | Port Conway VA           |
| James Monroe                       | Westmoreland County VA |
+----------------------------------+------------------------+
5 rows in set (0.00 sec)
```

The `select` *Statement*

- A new column can be given a shorter and more informative heading

```
mysql> select concat(first_name,' ',last_name) as name,
    ->   concat(city,' ',state) as birthplace
    ->   from president limit 5;
+--------------------+-------------------------+
| name               | birthplace              |
+--------------------+-------------------------+
| George Washington | Wakefield VA            |
| John Adams         | Braintree MA            |
| Thomas Jefferson   | Albemarle County VA     |
| James Madison      | Port Conway VA          |
| James Monroe       | Westmoreland County VA |
+--------------------+-------------------------+
```

Dealing with Dates

- It is possible to operate on dates in many ways
  - A table can be sorted in date order
  - A selection can be made of particular dates or a range of dates
  - Parts of a date can be extracted, such as the year, month or day
  - The difference between two dates can be calculated
  - A date may be computed by adding or subtracting an interval to or from a date

Dealing with Dates

- Find all presidents who died in the 1970’s

```
mysql> select last_name, first_name, death
    ->   from president
    ->   where death >= '1970-01-01' and death < '1980-01-01';
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S    | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

Dealing with Dates

- You can select on the *name* of the month or day of the week
- Alternatively month *number* or day of the month *number* can be used

```
mysql> select last_name, first_name, birth
    ->   from president
    ->   where monthname(birth)='March' and dayofmonth(birth) = 29;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Tyler     | John       | 1790-03-29 |
+-----------+------------+------------+
```

Dates and Calculations

- There are functions to deal with numerical data
- Normal arithmetic operations are possible

```
mysql> select last_name, first_name, birth, death,
    ->   floor((to_days(death) - to_days(birth)/365)) as age
    ->   from president where death is not null
    ->   order by age desc limit 5;
+------------+------------+------------+------------+--------+
| last_name  | first_name | birth      | death      | age    |
+------------+------------+------------+------------+--------+
| Reagan     | Ronald W.  | 1911-02-06 | 2004-06-05 | 730189 |
| Nixon      | Richard M  | 1913-01-09 | 1994-04-22 | 726490 |
| Johnson    | Lyndon B.  | 1908-08-27 | 1973-01-22 | 718735 |
| Truman     | Harry S    | 1884-05-08 | 1972-12-26 | 718732 |
| Eisenhower | Dwight D.  | 1890-10-14 | 1969-03-28 | 717356 |
+------------+------------+------------+------------+--------+
```

Wild Cards

- % matches any sequence of characters
- _ matches any single character
- Selecting using wildcards uses `like` rather than `=`

```
mysql> select last_name, first_name from president
    ->   where last_name like 'W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
+------------+------------+
```

Summaries

- Summaries including statistical summaries can be obtained directly from MySQL
- `count(*)` gives the number of rows selected by your query

```
mysql> select count(*) from president;
+----------+
| count(*) |
+----------+
|       42 |
+----------+
```

Summaries

- `count(*)` counts every row
- `count(`*columnname*`)` only counts non-null values

```
mysql> select count(*), count(suffix), count(death)
    ->   from president;
+----------+---------------+--------------+
| count(*) | count(suffix) | count(death) |
+----------+---------------+--------------+
|       42 |             1 |           37 |
+----------+---------------+--------------+
```

Summaries

- `count` can be combined with `distinct` to count only the number of distinct values

```
mysql> select count(distinct state) as 'State Count'
    ->   from president;
+-------------+
| State Count |
+-------------+
|          20 |
+-------------+
```

Summaries

- `count` combined with `group` can be used to create tabulations

```
mysql> select state, count(*) as count from president
    ->   group by state order by count desc limit 5;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| NY    |     4 |
| MA    |     4 |
| NC    |     2 |
+-------+-------+
```

Summaries

- The obvious statistical functions are available

```
mysql> select state as State,
    ->   round(avg((to_days(death) - to_days(birth))/365.25),2)
    ->     as Age
    ->   from president where death is not null
    ->   group by state order by age limit 4;
+-------+-------+
| State | Age   |
+-------+-------+
| KY    | 56.17 |
| VT    | 58.81 |
| NC    | 60.10 |
| OH    | 62.82 |
+-------+-------+
```

Summaries

- Combining information from tables is very important
- This example combines a table with itself
- It also illustrates how to identify the correct column where the same name appears in more than one table

```
mysql> select p1.last_name, p1.first_name, p1.city, p1.state
    ->   from president as p1, president as p2
    ->   where p1.city = p2.city and p1.state = p2.state
    ->         and (p1.last_name != p2.last_name or
    ->              p1.first_name != p2.first_name)
    ->   order by state, city, last_name;
+-----------+-------------+-----------+-------+
| last_name | first_name  | city      | state |
+-----------+-------------+-----------+-------+
| Adams     | John        | Braintree | MA    |
| Adams     | John Quincy | Braintree | MA    |
+-----------+-------------+-----------+-------+
```

Practicalities

Connecting to `mysql`

- An option file can be used to store parameters
- On Unix, the file is `~/.my.cnf`
- The syntax of the file is:

```
[client]
host=serverhost
user=yourusername
password=yourmysqlpassword
```

- This will set parameters for all client programs, such as `mysql`, and `mysqldump`
- The paragraph above could be followed by a similar paragraph headed by `[mysql]` if you wanted different connection parameters for that program
- This setup allows your password to be hidden

Connecting to `mysql`

- You could create an alias also such as

```
alias sampdb 'mysql -h stat71.stat.auckland.ac.nz
  -u root -p *****'
```

where `*****` is the password for MySQL

Using Scripts

- You can run scripts in batch mode:

```
stat71/dscott1> mysql sampdb < create_presidents.sql
stat71/dscott1> mysql -t sampdb < query.sql >
  outputfile
```

- The second command outputs results in the tabular format used when running `mysql` interactively
- The option `-t` also allows the redirection of output

Using XEmacs

- XEmacs will recognise a file of `mysql` commands if the extension `.sql` is used, provided `sql.el` is available
- Then it is possible to start `mysql` in a split window using `M-x sql-mysql`
- I can't get submission of commands to work on Unix
- I get a split window but nothing else when using Windows

Data Types

Numeric Types

- Integers can be `SIGNED` or `UNSIGNED`
- Various sizes, `TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT`
- Range from `TINYINT` with range -127 to 127 signed, or 0 to 255 unsigned, to `BIGINT` with range -2^{63} to $2^{63}-1$ signed, or 0 to $2^{64}-1$ unsigned
- Floating point can be `FLOAT` or `DOUBLE`

Character Types

- `CHAR`, a fixed length character string
- `VARCHAR`, a variable length character string
- `BLOB`, a binary large object, different sizes possible
- `TEXT`, a text string, different sizes possible
- `ENUM`, an enumeration; columns may be assigned one enumeration member
- `SET`, a set; columns may be assigned multiple set members

Date and Time Types

- `DATE`, a date in `'YYY-MM-DD'` format
- `TIME`, a time value in `'hh:mm:ss'` format
- `DATETIME`, a date and time value in `'YYYY-MM-DD hh:mm:ss'` format
- `YEAR`, a year value in `'YYYY'` format

Creating, Altering, Dropping

Databases

- `CREATE DATABASE`
- `DROP DATABASE`
- `USE`

Tables

- `CREATE TABLE`
- `DROP TABLE`
- `CREATE INDEX`
- `DROP INDEX`
- `ALTER TABLE`
- `DELETE`
- `INSERT`
- `LOAD DATA`
- `UPDATE`

Wine Cellar Example

Create the Table

```
drop table if exists cellar;
create table cellar (
  WineID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Location VARCHAR(10) NOT NULL,
  Row TINYINT UNSIGNED NOT NULL,
  Tile TINYINT UNSIGNED NOT NULL,
  Year VARCHAR(4) NULL,
  Winery VARCHAR(50) NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Grape VARCHAR(50) NOT NULL,
  Country VARCHAR(20) NOT NULL,
  Type VARCHAR(20) NOT NULL,
  Price VARCHAR(10) NOT NULL,
  Closure VARCHAR(10) NULL,
  Drink VARCHAR(50) NULL,
  Composition VARCHAR(50) NULL
);
```

Read the Data In

```
load data infile "c:/dscott/Teaching/782/Lectures/mySQLExamples/cellar.c
into table cellar
fields terminated by ','
        optionally enclosed by '"'
lines terminated by '\r\n'
ignore 1 lines
(Location, Row, Tile, @Year, Winery, Name, Grape,
         Country, Type, @Price, Closure, Drink, Composition)
set Year = if(@Year="NV",NULL,@Year),
            Price = substring(@Price,2)
;
```

Check the Table

```
### Code to test this
select * from cellar order by WineID limit 10;

alter table cellar
  modify Year Year;

describe cellar;
```

MySQL and R

Setting Up

- On Windows, the software required for setting up the connection between **R** and MySQL is well-described in James Curran's document:

  `www.stat.auckland.ac.nz/~dscott/782/workingwithdatabases.pdf`

- Note that there are a number of parts to the jigsaw
  - Access to MySQL Server on some machine, not necessarily the same machine as you are using
  - MySQL Client, `mysql` on the machine you are using
  - MySQL Connector/ODBC
  - **R** and the package RODBC
- It is also necessary to set up MySQL Connector/ODBC after it has been installed

Setting Up on Unix

- On Unix MySQL Server, MySQL Client, **R** and the package RODBC are required again
- ODBC is primarily Windows software, but there are Unix versions
- An ODBC driver manager needs to be installed
- This is a specialized task, an experienced administrator is probably required
- A description of how to do this is given by Brian Ripley in the `README` file which comes as part of the package RODBC
- Note that there are other connection possibilities—there is a package RMySQL for example

Setting Up in R

- Connection should then be easy

```
library(RODBC)
channel <- odbcConnect("stats782", uid = "root", pwd = "secret")
### Retrieve the first record of the table and check it
query <- "SELECT *
          FROM cellar
          LIMIT 1"
queryResult <- sqlQuery(channel, query)
str(queryResult)
```

- The query returns a data frame
- Note that character strings are treated as factors, just as in using `read.table`

Queries Using R

```
> str(queryResult)
'data.frame': 1 obs. of  14 variables:
 $ WineID     : int 1
 $ Location   : Factor w/ 1 level "Box": 1
 $ Row        : int 1
 $ Tile       : int 1
 $ Year       : int 2004
 $ Winery     : Factor w/ 1 level "Te Mata": 1
 $ Name       : Factor w/ 1 level "Awatea": 1
 $ Grape      : Factor w/ 1 level "CS/Merlot/CF/PV": 1
 $ Country    : Factor w/ 1 level "NZ": 1
 $ Type       : Factor w/ 1 level "Red": 1
 $ Price      : num 29.9
 $ Closure    : Factor w/ 1 level "Cork": 1
 $ Drink      : logi NA
 $ Composition: Factor w/ 1 level "34%,33%,20%,13%": 1
```

More Sophisticated Queries

- Create a table using MySQL

```
> query <- "SELECT Name, Type, AVG(Price) AS AveragePrice
+           FROM cellar
+           WHERE Winery ='Te Mata'
+           GROUP BY Name ORDER BY AveragePrice DESC"
> queryResult <- sqlQuery(channel, query)
> queryResult
        Name  Type AveragePrice
1  Coleraine   Red     55.01944
2   Bullnose   Red     35.92500
3     Elston White     31.16714
4     Awatea   Red     29.55809
5 Cape Crest White     24.29400
6 Woodthorpe   Red     17.70000
```

More Sophisticated Queries

- Read the data into **R**, create the table in **R**

```
> query <- "SELECT Name, Type, Price
+           FROM cellar
+           WHERE Winery ='Te Mata'"
> TeMata <- queryResult[order(queryResult$Name,queryResult$Type),]
> AveragePrice <- tapply(TeMata$Price,TeMata$Name,mean)
> WineType <- unique(TeMata[,1:2])
> data.frame(WineType,AveragePrice=as.numeric(AveragePrice))
          Name  Type AveragePrice
1       Awatea   Red     29.55809
51    Bullnose   Red     35.92500
148 Cape Crest White     24.29400
13   Coleraine   Red     55.01944
141     Elston White     31.16714
49  Woodthorpe   Red     17.70000
```