

# Chapter 1

## Introduction to MySQL

Aoife McLysaght

### 1.0 Conventions in this document

Commands to be entered on the UNIX/Linux command line are preceded by `unixprompt>` and those to be entered on the MySQL prompt are preceded by `mysql>`. UNIX commands are case sensitive whereas `mysql` commands (except passwords) are not.

### 1.1 The MySQL interface

One simple (in terms of its appearance and capabilities) way of accessing MySQL is through the standard interface. To enter the MySQL interface

```
unixprompt> mysql -u wbyeats -p
Enter password:
```

```
mysql>
```

The “-u” tag precedes the username, and the “-p” tag invokes the password prompt. To log in as any other user replace the username accordingly.

MySQL commands are then typed on the `mysql` command line. The end of each command is defined by a semicolon `;`. Once you have entered the `mysql` interface you can select a database to look at (with the `use` command) and use any MySQL queries to read, edit, or add data.

```
mysql> use tissueinfo;
mysql> show tables;
+-----+
| Tables_in_tissueinfo |
+-----+
| gene_map              |
| gene_info             |
+-----+
mysql> quit
```

## 1.2 Basic MySQL administration tools

Usually only the root user has permission to create new databases and new users for the MySQL server. The MySQL user is independent of any other username. This means that an individual may use more than one MySQL username. To enter the MySQL interface as root:

```
unixprompt> mysql -u root -p
Enter password:
```

```
mysql>
```

### 1.2.1 Create a MySQL database

It is easy to create a new MySQL table within the MySQL prompt.

```
mysql> CREATE DATABASE tissueinfo;
```

### 1.2.2 Users, passwords, and privileges

We now have an empty database, but we don't yet have any users to access this database. To simultaneously create a user, assign a password, and grant access to this newly created database enter:

```
mysql> GRANT USAGE ON tissueinfo.* to wbyeats@localhost
IDENTIFIED BY 'ode2maud';
```

This creates the user “wbyeats” if it doesn't already exist, and sets the password to “ode2maud”. Note that if the user “wbyeats” already exists the password will be set to “ode2maud” (even if the password was previously set to something different). This statement also grants wbyeats access to all of the tables within the tissueinfo database (specified by “tissueinfo.\*”).

One of the attractive features of MySQL is the strict security it gives your data. The tradeoff is some extra work for the database administrator, because access privileges must be individually set. Granting usage only allows the user to log in to the database, but not to actually look at the data or enter any data. To grant these privileges the root user must also specify:

```
mysql> GRANT SELECT, INSERT ON tissueinfo.* to wbyeats@localhost
IDENTIFIED BY 'ode2maud';
```

Which gives “wbyeats” permission to look at data (SELECT) and to add new data (INSERT). If you trust “wbyeats” you can grant all possible permissions (including permission to delete any data in the database) with the simple statement:

```
mysql> GRANT ALL ON tissueinfo.* to wbyeats@localhost
IDENTIFIED BY 'ode2maud';
```

### 1.2.3 Account settings: .my.cnf

If you are frequently using `mysql` through the unix commands or the `mysql` interface then the requirement to specify username and password every time quickly becomes tedious. Within UNIX/Linux you can write these parameters into a file called `.my.cnf` in your home directory. This file should contain your username and password information in exactly the following format.

```
[client]
user=wbyeats
host=localhost
password=ode2maud
```

MySQL will automatically read this information when you are using the MySQL interface or system commands (at the UNIX prompt), but not when connecting to the MySQL database from within a Perl script (see later). This means you do not need to specify `-u wbyeats -p` when executing commands. For the rest of this document the commands will be written as if this file is in place. If it is not you will need to add the `-u wbyeats -p` parameters to the command line.

## 1.3 MySQL database structure

MySQL databases consist of a(ny) number of tables. Tables hold the data. Tables are made up of columns and rows. A user that has been given `CREATE` and `DROP` permissions on a database can create and remove tables of that database. The `CREATE TABLE` command simultaneously creates the table and defines its structure (although the structure of the table can later be changed using the `ALTER TABLE` command).

### 1.3.1 Columns

A table consists of several columns each of which has a specific data type (*e.g.*, integer, text). It is useful to define columns correctly because it has implications for sorting the data (numeric versus text) and for the size of an allowed element in the column. Column types include:

INT	integer
FLOAT	Small floating-point number
DOUBLE	Double-precision floating-point number
CHAR(N)	Text N characters long (N=1..255)
VARCHAR(N)	Variable length text up to N characters long
TEXT	Text up to 65535 characters long
LONGTEXT	Text up to 4294967295 characters long

### 1.3.2 Creating tables

The `CREATE TABLE` command can either be entered at the `mysql>` prompt or can be written into a file and sent into MySQL later. The latter is preferable because you retain a record of how created the table. A table may be created as follows:

```
DROP TABLE IF EXISTS gene_map;
CREATE TABLE gene_map (
  gene VARCHAR(255) NOT NULL,
  chromosome INT NOT NULL,
  cM_position FLOAT NOT NULL,
  id INT NOT NULL AUTO_INCREMENT,
);
```

This creates a table called `gene_map` with four columns: `gene`, `chromosome`, `cM_position`, `id`. The `id` column is a useful feature for keeping track of data. In this case it is an automatically assigned integer.

To make searching of long databases more efficient you can index some or all columns of a table. This will require more filespace, but makes database queries run a lot faster. The above table could be redefined with indices as follows:

```
DROP TABLE IF EXISTS gene_map;
CREATE TABLE gene_map (
  gene VARCHAR(255) NOT NULL,
  chromosome INT NOT NULL,
  cM_position FLOAT NOT NULL,
  id INT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (id),
  INDEX i_g (gene),
  INDEX i_c (chromosome),
  INDEX i_cp (cM_position)
);
```

This file (or any file of MySQL commands) can be executed for the `tissueinfo` database as follows:

```
unixprompt> mysql tissueinfo < filename
```

### 1.3.3 Adding data to MySQL

Once you have created a table, you can start filling it with data. One simple way of adding a lot of data is by using the `mysqlimport` system command. This will read in a text file where data for each table row are separated by newlines, and data for each column are separated by tabs and in the same order as the columns were defined. The file should be named according to the convention `tablename.txt.table`, replacing “`tablename`” appropriately. A sample file for the `gene_map` table might look like this:

```
agrc259    1    233.3
adp0       10   9.3
adp4       4    4.3
```

This is then imported into the tissueinfo database of MySQL with

```
unixprompt> mysqlimport --local tissueinfo gene_map.txt.table
```

Data rows may also be added one by one inside the MySQL interface:

```
mysql> INSERT INTO gene_map VALUES('agrc259','1','233.3','NULL');
mysql> INSERT INTO gene_map (gene, chromosome) VALUES('adp0','10');
```

Note that if you specify a value for every column in order it is not necessary to declare the column names in the statement (*e.g.*, first statement above). If you only want to add data to some columns, or to add them in a different order, then you must declare the column names after the table name in the statement (*e.g.*, second statement above). In this case columns with no value specified will be “NULL” or will get the default value (“0” for numeric columns, “” for text columns, and the next integer for auto.increment columns) if “NOT NULL” was specified during table definition.

### 1.3.4 Reading MySQL databases: SELECT

The **SELECT** command is used to view all or some of the elements of a table or multiple tables. The basic format is **SELECT column FROM table**; “\*” indicates all columns. “%” is the wildcard. Here are some examples:

```
mysql> SELECT * FROM gene_map;
mysql> SELECT * FROM gene_map WHERE chromosome='10';
mysql> SELECT gene FROM gene_map WHERE chromosome='10' AND cM_position
    >= '100';
mysql> SELECT gene, cM_position FROM gene_map WHERE chromosome='10'
    ORDER BY cM_position;
mysql> SELECT * FROM gene_map WHERE gene LIKE 'asg%';
```

It is also possible to select data from multiple tables where they have at least one column in common. The following example selects the chromosome column from the gene\_map table and the description column from the gene\_info table and links the data by the gene.

```
mysql> SELECT gene_map.chromosome, gene_info.description FROM gene_map,
    gene_info WHERE gene_map.gene = gene_info.gene;
```

### 1.3.5 Quick notes

Some other useful MySQL commands (read the book!).

```
ALTER TABLE  add new columns, change column types, remove columns
UPDATE        change some values of an existing data record (table row)
DELETE        delete specific rows from a table
DROP TABLE   delete a whole table
```

## Chapter 2

# MySQL and Perl

### 2.1 Set up MySQL within your Perl program

You need to tell Perl where to find certain things in order to get it to communicate with your MySQL database. I always do this via a subroutine which is neat, and easy to paste into new programs.

```
$database='tissueinfo';

$dbh=&start_MySQL($database);

sub start_MySQL{
    $database = $_[0];
    use DBI;
    $DBD = 'mysql';
    $host = 'localhost';
    $user = 'wbyeats';
    $passwd = 'ode2maud';

    $dbh = DBI->connect("DBI:$DBD:$database:$host",
                      "$user","$passwd",
                      { RaiseError => 1, AutoCommit => 1 });
    return $dbh;
}
```

### 2.2 Using Perl to query MySQL databases

#### 2.2.1 Retrieving all values from a table

There are several possible ways to do this, differences you find in programs by different people are mostly a matter of style. I like to use the variable `$st` to hold the query statement, and then the other variable names become logical

progressions from this: `$sth` is the statement handle; `$rv` is the results variable and is just used here for a quick check.

```
$table='gene_map';

$st = "SELECT id, tissue FROM $table";
$sth = $dbh-> prepare($st)
    or die "Can't prepare $sth: $dbh->errstr\n";
$rv = $sth->execute;
if ($rv > 0){
    while (($id, $gene) = $sth->fetchrow_array){
        #print these out
        print "$id\t$gene\n";

        #or save them in an array or hash (associative array) to use later
        $names{$id}=$gene;

        #or use them now
        $x += $id;
        $y .= $gene;
    }
}
else{
    die "Problem with query, no results found\n";
}
```

The MySQL statement in `$st` can be anything that would normally work in the MySQL interface. So you may order the output by appending `ORDER BY id`, or similarly filter general output by appending `WHERE chromosome > '10'` to the statement. It is good practice to replace table names with variables as shown above.

### 2.2.2 Retrieving specific results from a table

Sometimes rather than take all of the data from a table, you just want to look up one specific value or set of values.

```
$table='gene_map';

$st = "SELECT chromosome, cM_position FROM $table WHERE gene=?";
$sth = $dbh-> prepare($st)
    or die "Can't prepare $sth: $dbh->errstr\n";

@gene=('agrc259','adp0','adp4'); #or some other array you defined earlier

foreach $gene (@gene){
    $rv = $sth->execute($gene);
```

```

if ($rv > 0){
    while (($chr, $cM) = $sth->fetchrow_array){
        print "$chr\t$cM\n";    #print out the values
        $chr{$gene}=$chr;      #or save them for later
        $pos{$gene}=$cM;
    }
}
else{
    die "Problem with query, no results found\n";
}
}

```

For efficiency, the statement definition, and statement handle preparation commands should be done outside the loop as shown.

### 2.2.3 Merging results and/or queries from different tables

Sometimes the data you want is stored in several different tables for space saving reasons and to remove redundancy. MySQL allows you to query across several tables, and Perl handles this nicely.

```

$table_a='gene_map';
$table_b='gene_info';

$st = "SELECT $table_a.chromosome, $table_a.cM_position, $table_b.description
      FROM $table_a, $table_b WHERE $table_a.gene=$table_b.gene
      AND WHERE gene=?";
$sth = $dbh-> prepare($st)
    or die "Can't prepare $sth: $dbh->errstr\n";

@gene=('agrc259','adp0','adp4'); #or some other array you defined earlier

foreach $gene (@gene){
    $rv = $sth->execute($gene);
    if ($rv > 0){
        while (($chr, $pos, $desc) = $sth->fetchrow_array){
            print "$gene\t$chr\t$pos\t$desc\n";
        }
    }
    else{
        die "Problem with query, no results found\n";
    }
}

```

To be completely clear and readable you can specify which table each value is coming from by using something like `$table_a.chromosome` each time, but this is only strictly necessary when you are looking for data from a column name that exists in both tables.

## 2.3 Using Perl to create/modify MySQL tables

It is possible to create and modify tables directly from Perl, but from experience I prefer to write the create or modify commands out to a file which can then be redirected into MySQL. This is safer from the point of view of making mistakes, and also makes debugging a bit easier.

### 2.3.1 Creating new MySQL tables

There are two things you need to do to create a new MySQL table: define the structure of the table; and fill it with data.

```
open (OUT, "mysqldata.out") || die "cannot open mysqldata.out\n";

$table='gene_map';

print OUT <<end_format;
CREATE TABLE $table (
    gene VARCHAR(255) NOT NULL,
    chromosome INT NOT NULL,
    cM_position FLOAT NOT NULL,
    id NOT NULL AUTO_INCREMENT,
);
end_format

foreach $gene (@gene){
    print OUT "INSERT INTO $table VALUES('$gene','$chr{$gene}',
        '$pos{$gene}', 'NULL'); \n";
}

```

To send this data into mysql do `mysql tissueinfo < mysqldata.out` on the command line.

### 2.3.2 Modifying existing tables

```
foreach $gene (keys %chr){
    print OUT "UPDATE $table SET chromosome='$chr{$gene}' WHERE
        gene='$gene';\n";
}

```