1 Requirements

Before we begin there are some requirements:

2 Software requirements

You will need the latest versions of

- R
- RODBC
- MySQL
- MyODBC

As of the time of writing the latest versions are:

Package	Version
R	2.50
RODBC	1.20
MySQL	5.0.41
MyODBC	5.0.11-Beta

These files may be downloaded from

Package	URL
R	http://www.r-project.org
RODBC	http://www.r-project.org
MySQL	http://dev.mysql.com/downloads/mysql/5.0.html
MySQL Connector/ODBC	MySQL Connector/ODBC

2.1 Example input data

To follow the example you'll also need the input file I used. My input file consists of 1 million simulated DNA records. each record consists of an ID, and two alleles at each of ten loci, making a total of 21 pieces of information per individual. You can download this input file from dna.gz. Be warned though that this file is 16MB compressed and 55MB uncompressed. You can uncompress it using the unix `gzip` utility or using a zip file program like WinZip (`www.winzip.com`).

3 Flattening the data

One of the things that is difficult (for me at least) to get used to is that you're unlikely to end up storing the data in the database in the same way you would keep it for statistical purposes. That is, as statisticians, we're used to keeping our data in a rectangular array with rows representing observations, and columns representing variables. In the example dataset, you'll find that the data is stored

in exactly this way. There is a column for the ID of the individual, and then two column for each pair of alleles at each locus. This format however, doesn't not lend itself well to the type of queries we'd like to make from a database. For example, one thing I like to find out about with DNA is a list of all uniqye alleles at a particular locus. It is very difficult to frame this request in the SQL language with the data in the current format. Therefore, I am going to "flatten" the data. That is I'm going to rearrange it in the format I want to initially read it into the database with. In this example, I will have six columns. These are

Column	**Description**
RecordID	A unique indentified for every entry in the database
ID	An identifier (from 0 to 999,999) that tells the user who the alleles belong to
Subpop	An identifier (from 1 to 10) that tells the user which subpopulation this record belongs to
Locus	The name of the locus that these alleles come from
Allele1	The first allele for the individual at the given locus
Allele2	The second allele for the individual at the given locus

This makes more sense when you see an example. The first line in my input file looks like this

```
0,1,5,1,5,10,6,9,2,4,11,11,4,3,5,5,11,5,1,6,4,6
```

There are 10 loci in my database (D16,D18,D19,D2,D21,D3,D8,FGA,THO1,VWA). Therefore individual 0 from subpopulation 1 has alleles 5 and 1 at locus D16, alleles 5 and 10 at locus D18 and so on. When I flatten this it becomes:

RecordID	ID	Subpop	Locus	Allele1	Allele2
1	0	1	D16	5	1
2	0	1	D18	5	1
3	0	1	D19	6	1
4	0	1	D2	2	1
5	0	1	D21	11	1
6	0	1	D3	4	1
7	0	1	D8	5	1
8	0	1	FGA	11	1
9	0	1	THO1	1	1
10	0	1	VWA	4	1

How you do this is up to you. However, for those of you conversant in Perl, there is a script that makes this conversion available from http://www.stat.auckland.ac.nz/ curran/flatten.pl. Given that this is not a description of how to use Perl I won't explain how it works.

4 Creating the database

In this section I assume that you have run and installed MySQL in the default location `c:/mysql`. To create the database we need to run the command line interface to MySQL. Select **Start > MySQL > MySQL Server 5.0 >**

MySQL Command Line Client. Enter your password if you set one when setting up MySQL. If everything is working correctly you should see

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.41-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We will call our database `dnaexample`. Type

```
mysql> create database dnaexample;
```

4.1 Creating a table

We will read our flattened input file into a table in the database. The table has the columns given above `RecordID`, `Id`, `SubpopID`,`Locus`, `Allele1`, and `Allele2`. The first column, `RecordID`, will be the primary key of the table. The primary key of a table uniquely identifies each record in the table. It's a good idea to put a primary key in every table you create. We'll also set it up so that the primary key is never empty (NOT NULL) and autoincrements itself (AUTO_INCREMENT) if the user doesn't enter it. To create our table we use the `CREATE TABLE` command. Firstly however, we use the `USE` command to select our new database.

```
mysql> use dnaexample;
Database changed
mysql> create table dna_raw (
     -> RecordID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     -> Id INT NOT NULL,
     -> SubopID INT NOT NULL,
     -> Locus VARCHAR(10) NOT NULL,
     -> Allele1 VARCHAR(10),
     -> Allele2 VARCHAR(10));
```

You'll notice that I have given the `Allelel1` and `Allele2` columns the type `VARCHAR`. This allows the user to specify non-numeric alleles such as X and Y for Amelogenin and use an 'R' for rare alleles.

4.2 Reading the data into the table

To read the data into the table we use the MySQL command `LOAD DATA INFILE`

```
mysql> load data infile 'c:/curran/work/book/data/db-input.csv'
    -> into table dna_raw
    -> fields terminated by ',';
```

This is a lengthy operation as there are 10 million records to read. On my system it took about 3 minutes and 20s. If you're sucessful then you should see something like:

```
Query OK, 10000000 rows affected (3 min 19.63 sec)
Records: 10000000  Deleted: 0  Skipped: 0  Warnings: 0
```

At this point you can use the `QUIT` command to exit the command line interface.

5 Setting up MySQL Connector/ODBC

The interface between R and MySQL is controlled by a driver which implements the 'Open Database Connectivity' standard (ODBC) for short. Download and run the installer. Once the installation has completed, open the Control Panel and double click **Administrative Tools**. Double click on **Data Sources (ODBC)**. This should bring up the dialog in figure 1

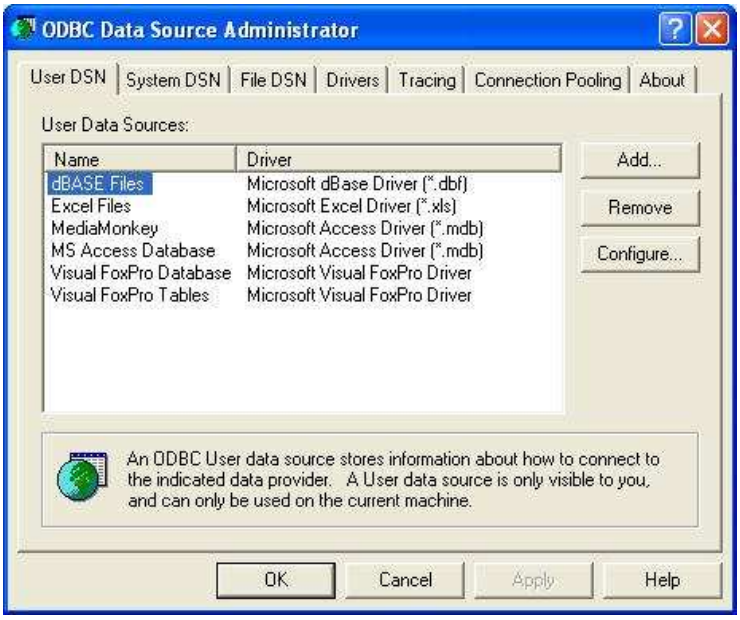

Figure 1: The ODBC Data Source Administrator dialog box

Click on the **Add...** button. This will bring up the dialog box in figure 2. Scroll down and select the **MySQL Connector/ODBC v5** item and click on **Finish**

This will bring up the configuration dialog shown in figure 3.

The values shown for the **Server** and **Port** items in figure 3 are the default values used with a standard MySQL. If you changed any of these defaults during your setup then the dialog box should reflect those changes. The **Data Source Name** is the external reference we will use to connect to the database. The

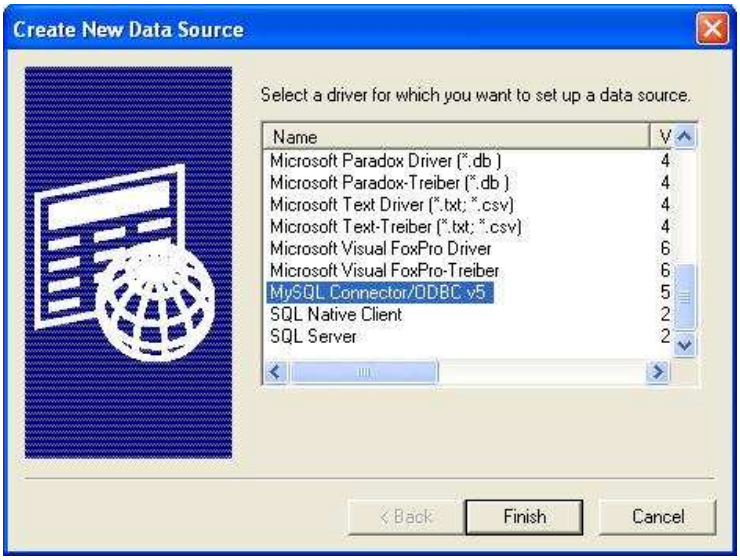

Figure 2: The Create New Data Source dialog box

Database is the database we created inside MySQL. Click **OK** to save the values and **OK** again to exit the Data Source Administrator. At this point we should be able to switch to R.

6 Using RODBC

Make sure you've installed the RODBC package. Once you have load it up using the `library` command.

```
library(RODBC)
```

6.1 Connecting to the database

The first step is to establish a connection to the database. We store the connection in a variable called `channel` - following the convention of the RODBC manual. This is sometimes referred to as a "handle" to the database.

```
channel <- odbcConnect("DNA_Example", "root", "james", believeNRows = FALSE)
```

If your password is something other than "password" then you should replace it in the command above. The `believeNRows` argument is set to `FALSE` to fix a bug in the MySQL Connector/ODBC v5 driver.

6.2 Querying the database

The workhorse of any software that implements SQL is the Query function. In RODBC this is called `sqlQuery`. This function takes two arguments, the ODBC connection handle, and a SQL query statement. MySQL, like every other database product, has it's own implementation of SQL. There is a ANSI standard but no single product implements all of its features. The reader is directed

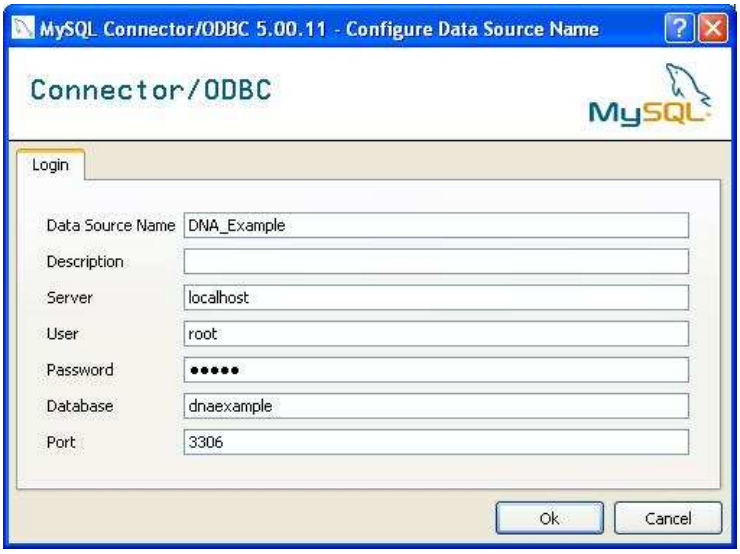

Figure 3: The Configure Data Source dialog box

to the online documentation http://dev.mysql.com/doc/refman/5.0/en/index.html, or any one of the books on the subject such as the MySQL Crash Course by Ben Forta.

6.2.1 Selecting some rows

In this example I want to retreive all the information I have about the person with Id number 1

```
query <- "select * from dna_raw where id=1"
sqlQuery(channel, query)
```

```
   RecordID Id SubpopID Locus Allele1 Allele2
1        11  1        1   D16       2       3
2        12  1        1   D18       8       3
3        13  1        1   D19       6       3
4        14  1        1    D2       8       3
5        15  1        1   D21      11       3
6        16  1        1    D3       5       3
7        17  1        1    D8       6       3
8        18  1        1   FGA      13       3
9        19  1        1  TH01       2       3
10       20  1        1   VWA       4       3
```

6.2.2 Counting records

In this example I want to find out how many different loci are used in my database. I know the answer is ten, but this is just an example.

```
query <- "select count(distinct locus) from dna_raw"
sqlQuery(channel, query)
```

```
  count(distinct locus)
1                    10
```

6.2.3 Creating a new table

All the database operations can be carried out with a SQL query. In this example I want to create a new table so that I can use it later on to store the results of another query. The table I will create is going to hold the locus names. There are two columns: `LocusID`, which is the primary key, and `Locus`, which will hold the locus name. It's a good idea to use the `paste` function to paste together longer SQL statements.

```
 query <- paste("create table if not exists loci",
                "(LocusID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,",
                "Locus VARCHAR(10) NOT NULL)")
 sqlQuery(channel, query)
```

```
character(0)
```

6.2.4 Inserting elements into a table

In this example we'll use the table we created in the previous section to hold the list of distinct (different) loci present in the dataset.

```
 query <- "insert loci (locus) select distinct locus from dna_raw"
 sqlQuery(channel, query)
```

```
character(0)
```

6.2.5 A more complex example

In this final example we'll use a combination of R and MySQL to determine the allele counts for locus D3 for each subpopulation and insert them back into the database.

```
 query <- paste("create table if not exists a",
                "(Allele VARCHAR(10) NOT NULL)")
 sqlQuery(channel, query)
```

```
character(0)
```

```
 query <- paste("insert a (Allele) select distinct Allele1",
                "from dna_raw where locus = 'd3'",
                "order by Allele1")
 sqlQuery(channel, query)
```

```
character(0)
```

```
 query <- paste("insert a (Allele) select distinct Allele1",
                "from dna_raw where locus = 'd3'",
                "order by Allele2")
 sqlQuery(channel, query)
```

```
character(0)
```

```
 query <- paste("select distinct Allele from a", "order by Allele")
 d3.alleles <- sqlQuery(channel, query)
 d3.alleles <- d3.alleles$Allele
 sqlQuery(channel, "drop table a")
```

```
character(0)
```

```
 query <- paste("create table if not exists d3",
                "(AlleleCountID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,",
                "Allele VARCHAR(10) NOT NULL,", "SubpopID INT NOT NULL,",
                "AlleleCount INT NOT NULL)")
 sqlQuery(channel, query)
```

```
character(0)
```

```
 for (subpopid in 1:10) {
     query <- paste("select Allele1,Allele2 from dna_raw",
                    "where locus = 'd3' and subpopid =",
                    subpopid)
     result <- sqlQuery(channel, query)
     result$Allele1 <- factor(result$Allele1, levels = d3.alleles)
     result$Allele2 <- factor(result$Allele2, levels = d3.alleles)
     d3 <- table(c(result$Allele1, result$Allele2))
     for (allele in 1:length(d3.alleles)) {
         query <- paste("insert into d3 ", "(allele,subpopid,allelecount)",
             " values ('", d3.alleles[allele], "',", subpopid,
             ",", d3[allele], ")", sep = "")
         sqlQuery(channel, query)
     }
     cat(paste("Subpop", subpopid, "done\n"))
 }
```

```
Subpop 1 done
Subpop 2 done
Subpop 3 done
Subpop 4 done
Subpop 5 done
Subpop 6 done
Subpop 7 done
Subpop 8 done
Subpop 9 done
Subpop 10 done
```

6.3 Closing the database connection

When we're finished accessing the database, we should remember to close the database connection using the `close` command

```
 close(channel)
```