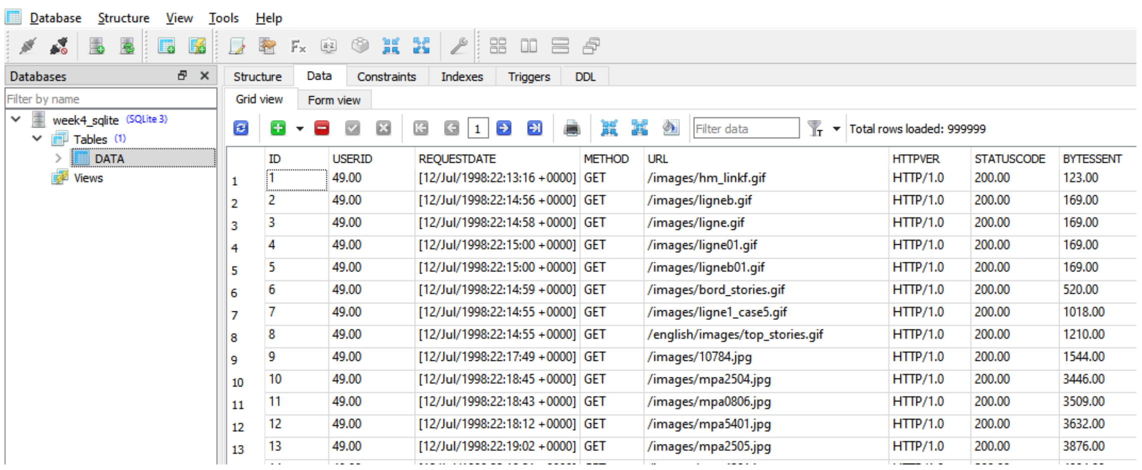
Extension Exercises for Tutorial Pack 2

(Optionally completed during LW2 tutorial or as homework)

In this week's tutorial pack, we have seen how to work with web log data stored in a CSV file. In practice, with very large log files (such as those from a busy site) it may be more appropriate to periodically store the website logs in a database management system (DBMS). Storing web log data in a database has many benefits over using flat files, including the flexibility to query the exact data we need using SQL (structured query language) and providing everyone in the organisation access to a central data repository.

In this case I have imported a large web log file into an SQLite database. This database can be found on in the module site on blackboard. The filename is "extension_exercises.db". In the screenshot below I have opened the database using the SQLiteStudio application (available on appsanywhere) to browse the available tables.



Total rows loaded: 999999

	ID	USERID	REQUESTDATE	METHOD	URL	HTTPVER	STATUSCODE	BYTESSENT
1	1	49.00	[12/Jul/1998:22:13:16 +0000]	GET	/images/hm_linkf.gif	HTTP/1.0	200.00	123.00
2	2	49.00	[12/Jul/1998:22:14:56 +0000]	GET	/images/ligneb.gif	HTTP/1.0	200.00	169.00
3	3	49.00	[12/Jul/1998:22:14:58 +0000]	GET	/images/ligne.gif	HTTP/1.0	200.00	169.00
4	4	49.00	[12/Jul/1998:22:15:00 +0000]	GET	/images/ligne01.gif	HTTP/1.0	200.00	169.00
5	5	49.00	[12/Jul/1998:22:15:00 +0000]	GET	/images/ligneb01.gif	HTTP/1.0	200.00	169.00
6	6	49.00	[12/Jul/1998:22:14:59 +0000]	GET	/images/bord_stories.gif	HTTP/1.0	200.00	520.00
7	7	49.00	[12/Jul/1998:22:14:55 +0000]	GET	/images/ligne1_case5.gif	HTTP/1.0	200.00	1018.00
8	8	49.00	[12/Jul/1998:22:14:55 +0000]	GET	/english/images/top_stories.gif	HTTP/1.0	200.00	1210.00
9	9	49.00	[12/Jul/1998:22:17:49 +0000]	GET	/images/10784.jpg	HTTP/1.0	200.00	1544.00
10	10	49.00	[12/Jul/1998:22:18:45 +0000]	GET	/images/mpa2504.jpg	HTTP/1.0	200.00	3446.00
11	11	49.00	[12/Jul/1998:22:18:43 +0000]	GET	/images/mpa0806.jpg	HTTP/1.0	200.00	3509.00
12	12	49.00	[12/Jul/1998:22:18:12 +0000]	GET	/images/mpa5401.jpg	HTTP/1.0	200.00	3632.00
13	13	49.00	[12/Jul/1998:22:19:02 +0000]	GET	/images/mpa2505.jpg	HTTP/1.0	200.00	3876.00

Database management systems (DBMS)

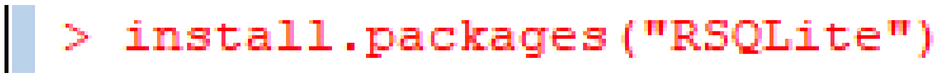
In simple terms, a DBMS is a software application that is responsible for managing and organising a set of data. Examples of DBMS include MySQL, MSSQL Server, SQLite, Oracle, and Postgres. DBMS can provide data security, tools to support data integrity and mechanisms to access, insert and query data.

The data store in this database is based on 1 million log entries from the World Cup 1998 website on the day following the final. In case you didn't know, the match was between Brazil and France and France won 3 – 0.

Database tables

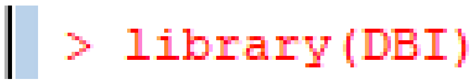
Instead of worksheets, as we are used to in Microsoft Excel, databases are often divided into tables. In this case each table might correspond to a subject matter or entity. Roughly speaking each table resembles a worksheet in Microsoft Excel. At present our database contains only a single table called "DATA". Each row (or record) in the DATA table represents a single request (or Hit) to the webserver.

To access this database from R we are going to need to install the RSQLite package. This package contains important functions used to communicate with SQLite databases. Note that, depending on the database you are using you may have to install a different package. For example, to access data stored in a Postgres database we could use the "RPostgres" package.



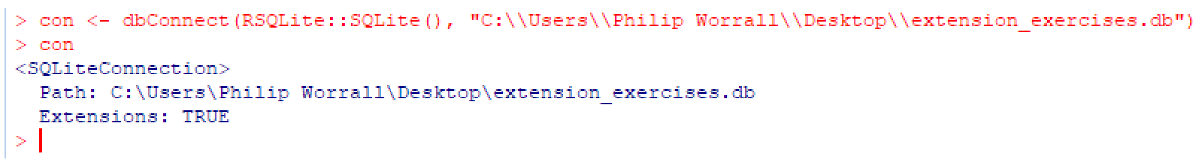
```
> install.packages("RSQLite")
```

Now that the relevant package has been installed, we need to load it by using the library function. In this case the correct library to be loaded is the "DBI" or Database Interface library.



```
> library(DBI)
```

We can connect to our database using the **dbConnect**() function. The first parameter is the name of the package and database type that we would like to access and the second is a path to where the database file is located. You will need to modify this path according to where you have saved the database file on your computer. From now on I will refer to this database as the variable "con" which is short for connection.



```
> con <- dbConnect(RSQLite::SQLite(), "C:\\Users\\Philip Worrall\\Desktop\\extension_exercises.db")
> con
<SQLiteConnection>
  Path: C:\Users\Philip Worrall\Desktop\extension_exercises.db
  Extensions: TRUE
> 
```

SQL (Structured Query Language)

SQL is a standardised syntax used to manage, update, delete and query data stored in a database. In most cases, SQL developed for one DBMS can be used across other DBMS with little to no modification.

A good beginner introduction to SQL can be found at https://www.w3schools.com/sql/

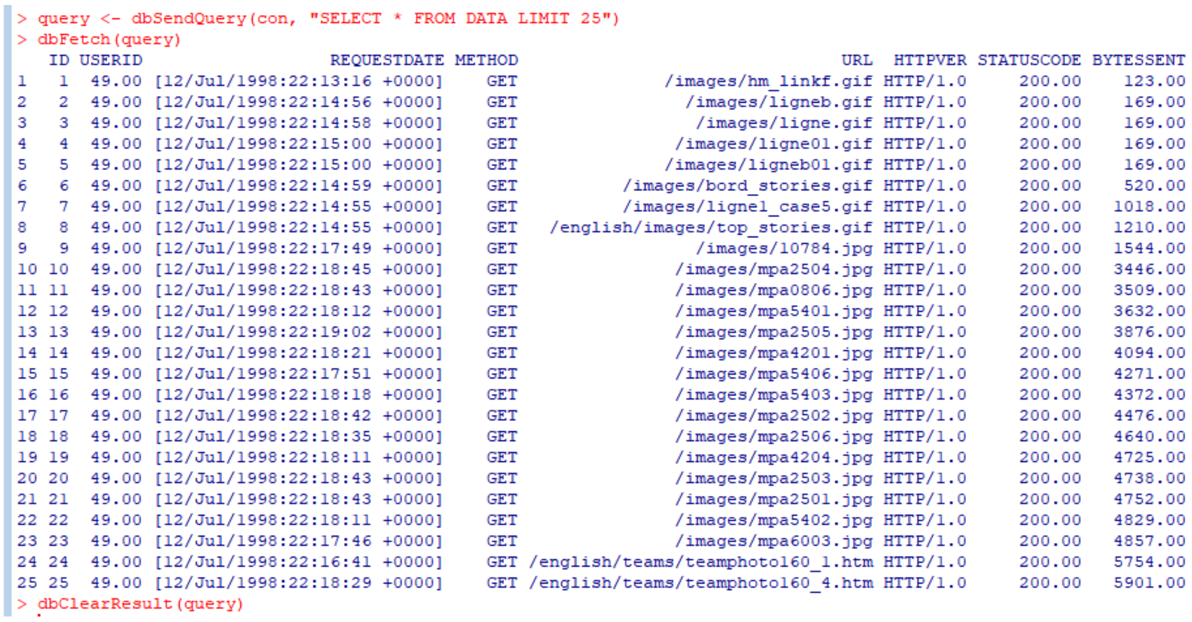
SQL statements roughly fall into one of the four main categories.

- **SELECT statements – For selecting data**
- **UPDATE statements – For updating data**
- **INSERT statements – For adding data**
- **DELETE statements – For removing data**

Queries to a database using R can be carried out in three steps.

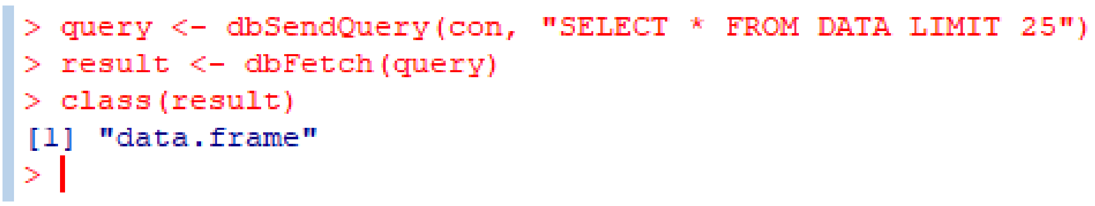
1. The first step is to use the **dbSendQuery**() function, specifying a connection to use as the first parameter and the SQL statement containing the query as the second parameter.
2. In the second step we use the **dbFetch**() function to retrieve the relevant records from the database
3. Finally, once we are done with this query we call the function **dbClearResult**(), which marks the query as complete and frees up our database ready for the next query to be sent.

In the screenshot below I demonstrate how to create a simple query (using these three steps) that fetches all columns for the first 25 rows in the DATA table.



```
> query <- dbSendQuery(con, "SELECT * FROM DATA LIMIT 25")
> dbFetch(query)
   ID USERID                  REQUESTDATE METHOD                               URL  HTTPVER STATUSCODE BYTESSENT
1   1  49.00 [12/Jul/1998:22:13:16 +0000]    GET              /images/hm_linkf.gif HTTP/1.0     200.00    123.00
2   2  49.00 [12/Jul/1998:22:14:56 +0000]    GET                /images/ligneb.gif HTTP/1.0     200.00    169.00
3   3  49.00 [12/Jul/1998:22:14:58 +0000]    GET                 /images/ligne.gif HTTP/1.0     200.00    169.00
4   4  49.00 [12/Jul/1998:22:15:00 +0000]    GET               /images/ligne01.gif HTTP/1.0     200.00    169.00
5   5  49.00 [12/Jul/1998:22:15:00 +0000]    GET              /images/ligneb01.gif HTTP/1.0     200.00    169.00
6   6  49.00 [12/Jul/1998:22:14:59 +0000]    GET          /images/bord_stories.gif HTTP/1.0     200.00    520.00
7   7  49.00 [12/Jul/1998:22:14:55 +0000]    GET          /images/lignel_case5.gif HTTP/1.0     200.00   1018.00
8   8  49.00 [12/Jul/1998:22:14:55 +0000]    GET   /english/images/top_stories.gif HTTP/1.0     200.00   1210.00
9   9  49.00 [12/Jul/1998:22:17:49 +0000]    GET                 /images/10784.jpg HTTP/1.0     200.00   1544.00
10 10  49.00 [12/Jul/1998:22:18:45 +0000]    GET               /images/mpa2504.jpg HTTP/1.0     200.00   3446.00
11 11  49.00 [12/Jul/1998:22:18:43 +0000]    GET               /images/mpa0806.jpg HTTP/1.0     200.00   3509.00
12 12  49.00 [12/Jul/1998:22:18:12 +0000]    GET               /images/mpa5401.jpg HTTP/1.0     200.00   3632.00
13 13  49.00 [12/Jul/1998:22:19:02 +0000]    GET               /images/mpa2505.jpg HTTP/1.0     200.00   3876.00
14 14  49.00 [12/Jul/1998:22:18:21 +0000]    GET               /images/mpa4201.jpg HTTP/1.0     200.00   4094.00
15 15  49.00 [12/Jul/1998:22:17:51 +0000]    GET               /images/mpa5406.jpg HTTP/1.0     200.00   4271.00
16 16  49.00 [12/Jul/1998:22:18:18 +0000]    GET               /images/mpa5403.jpg HTTP/1.0     200.00   4372.00
17 17  49.00 [12/Jul/1998:22:18:42 +0000]    GET               /images/mpa2502.jpg HTTP/1.0     200.00   4476.00
18 18  49.00 [12/Jul/1998:22:18:35 +0000]    GET               /images/mpa2506.jpg HTTP/1.0     200.00   4640.00
19 19  49.00 [12/Jul/1998:22:18:11 +0000]    GET               /images/mpa4204.jpg HTTP/1.0     200.00   4725.00
20 20  49.00 [12/Jul/1998:22:18:43 +0000]    GET               /images/mpa2503.jpg HTTP/1.0     200.00   4738.00
21 21  49.00 [12/Jul/1998:22:18:43 +0000]    GET               /images/mpa2501.jpg HTTP/1.0     200.00   4752.00
22 22  49.00 [12/Jul/1998:22:18:11 +0000]    GET               /images/mpa5402.jpg HTTP/1.0     200.00   4829.00
23 23  49.00 [12/Jul/1998:22:17:46 +0000]    GET               /images/mpa6003.jpg HTTP/1.0     200.00   4857.00
24 24  49.00 [12/Jul/1998:22:16:41 +0000]    GET /english/teams/teamphoto160_1.htm HTTP/1.0     200.00   5754.00
25 25  49.00 [12/Jul/1998:22:18:29 +0000]    GET /english/teams/teamphoto160_4.htm HTTP/1.0     200.00   5901.00
> dbClearResult(query)
```

If we store the results of the query into a new variable called result, we can confirm with the class() function that it is stored as a data frame object type.



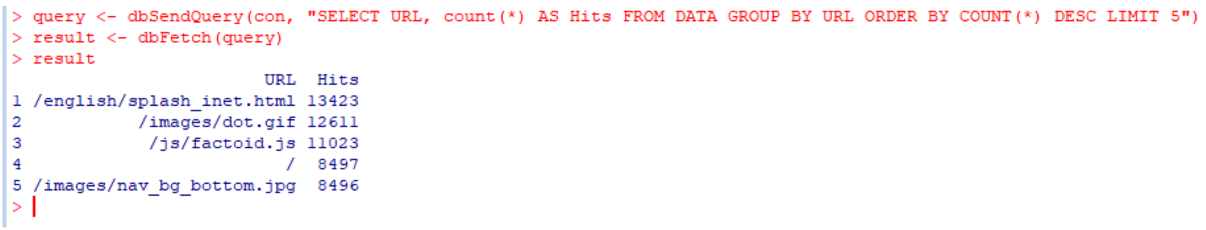
```
> query <- dbSendQuery(con, "SELECT * FROM DATA LIMIT 25")
> result <- dbFetch(query)
> class(result)
[1] "data.frame"
> 
```

Questions

Using your knowledge of R and SQL, attempt to answer the following questions.

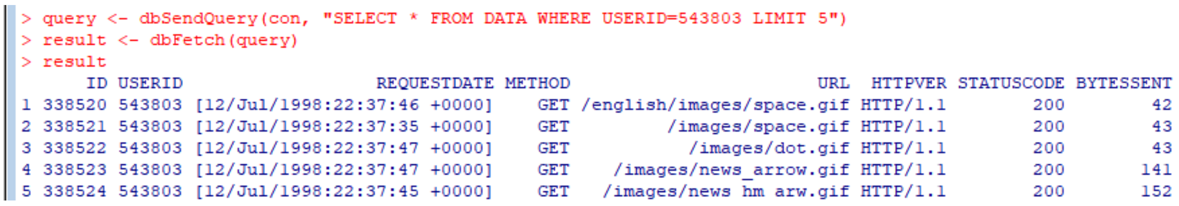
Qnum	Question	ANSWER
32	Download the SQLite database used previously and connect to it using R. Write a R program that selects the first 10 rows from the USERID column.	
33	Write a program that displays all the column names present in the DATA table.	
34	Write a R program that determines the number of hits to the entire website and prints the result.	
35	Write an R program that determines how many unique USERIDs there are in the first 1000 records.	

Previously we used the aggregate() and order() functions within R to group data and order the results. In the same way we can use SQL **GROUP BY** keyword to perform the equivalent analysis. The following SQL query groups all records from the DATA table according to their URL. For each group it counts the number of available records and stores it in a variable called "Hits". The groups are ordered in descending order of the number of Hits to each URL and only the top 5 most accessed URLs are select



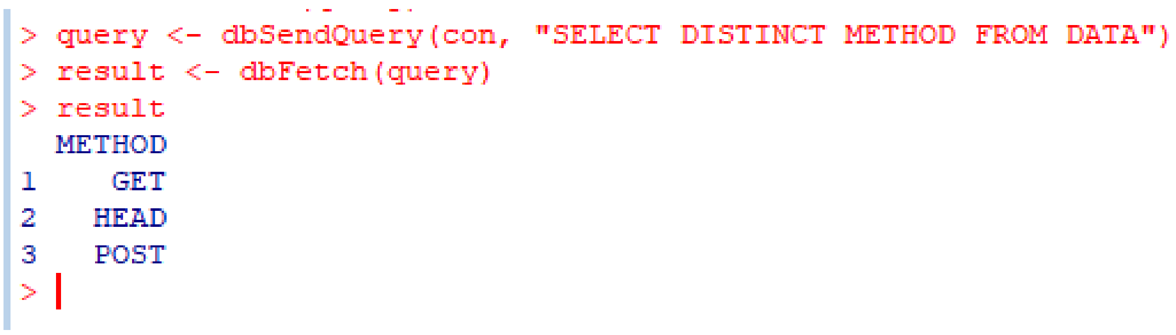
```
> query <- dbSendQuery(con, "SELECT URL, count(*) AS Hits FROM DATA GROUP BY URL ORDER BY COUNT(*) DESC LIMIT 5")
> result <- dbFetch(query)
> result
                      URL  Hits
1 /english/splash_inet.html 13423
2           /images/dot.gif 12611
3            /js/factoid.js 11023
4                         /  8497
5 /images/nav_bg_bottom.jpg  8496
> 
```

If you recall, in R when we wanted to filter data, we used slicing in combination with logical operators. In SQL the same task is achieved by specifying a **WHERE** condition. In this case we want the first five rows for all columns where the USERID is set to 543803.



```
> query <- dbSendQuery(con, "SELECT * FROM DATA WHERE USERID=543803 LIMIT 5")
> result <- dbFetch(query)
> result
      ID USERID                  REQUESTDATE METHOD                       URL  HTTPVER STATUSCODE BYTESSENT
1 338520 543803 [12/Jul/1998:22:37:46 +0000]    GET /english/images/space.gif HTTP/1.1        200        42
2 338521 543803 [12/Jul/1998:22:37:35 +0000]    GET         /images/space.gif HTTP/1.1        200        43
3 338522 543803 [12/Jul/1998:22:37:47 +0000]    GET           /images/dot.gif HTTP/1.1        200        43
4 338523 543803 [12/Jul/1998:22:37:47 +0000]    GET    /images/news_arrow.gif HTTP/1.1        200       141
5 338524 543803 [12/Jul/1998:22:37:45 +0000]    GET   /images/news hm arw.gif HTTP/1.1        200       152
```

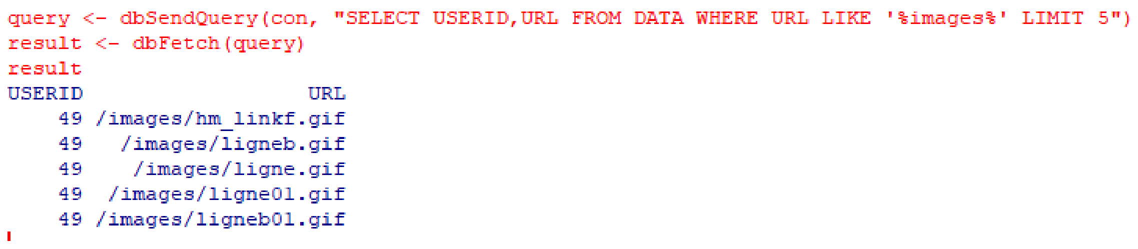
In R when we wanted to identify unique values within a vector or data frame we used the unique() function. In SQL the equivalent statement uses the **DISTINCT** keyword. In the example below I select all the unique values of the METHOD column across all rows in the DATA table



```
> query <- dbSendQuery(con, "SELECT DISTINCT METHOD FROM DATA")
> result <- dbFetch(query)
> result
  METHOD
1    GET
2   HEAD
3   POST
> 
```

In R when we wanted to search a piece of text for a pattern we used regular expressions through the gregexpr() function. In SQL a somewhat similar functionality can be achieved by using the **LIKE** keyword. In this case we can match records that match a wildcard. For example %a will match all records who data value ends with “a” whilst a% will match all records that start with “a”. %a% will match all records that contain “a”.

The following example demonstrates the use of the LIKE syntax to find all URLS containing the phrase “images” and returns the corresponding USERID and URL values for the first 5 matches.



```
query <- dbSendQuery(con, "SELECT USERID,URL FROM DATA WHERE URL LIKE '%images%' LIMIT 5")
result <- dbFetch(query)
result
USERID                   URL
    49 /images/hm_linkf.gif
    49   /images/ligneb.gif
    49    /images/ligne.gif
    49  /images/ligne01.gif
    49 /images/ligneb01.gif
```

Questions

Using your knowledge of R and SQL, attempt to answer the following questions.

Qnum	Question	ANSWER
36	Determine the top 10 most active users by number of hits and print their USERIDs and total hit counts.	
37	How many hits were made to the homepage?	
38	Determine the most used HTTP method throughout the day	
39	Determine the total number of different images available through the website	
40	Produce a table containing the number of HTTP errors by error code. For each error code identified, determine what proportion of all errors it accounts for.	