Database connections in R

19/02/20

Chris Mainey

Senior Statistical Intelligence Analyst
Healthcare Evaluation Data (HED)
University Hospitals Birmingham NHS FT

chris.mainey@uhb.nhs.uk

Introduction

- Relational (and other) databases common in 'real world'.
- Not always importing csv files into R!
- Look at how to connect to them and use them, with `odbc` type connection.

This session is adapted from HED's Introduction to R course:

- Two day introduction course, public or onsite. **(24th - 25th March, 9th -10 June)**

- We also offer other courses, including:
  - Introduction to R Markdown - **26th Feb, Birmingham**
  - Machine Learning methods in R **28th - 28th April**
  - Regression Modelling in R - **22nd - 23rd September**
  - R Essentials - **20th October**

More info, or book at: https://www.hed.nhs.uk/Info/hed-courses.aspx

SQL in one slide...

- **S**tructured **Q**uery **L**anguage
- Standard syntax (ANSI and ISO), but vendor specific dialects

Key elements:

- **SELECT**: The data fields you want out of a table
- **FROM**: The table (or tables, with joins) to query
- **WHERE**: Filter criteria
- **GROUP BY**: When using aggregates in SELECT, assigns group

```
SELECT Name,
       Age,
       MSOA
FROM Demographic
WHERE Age > 17
```

Joins:

```
SELECT tab1.AttendanceDate,
       tab2.Name
FROM PatientAttendances tab1 inner join
     Demographics tab2 ON tab1.PatID = tab2.PatID
```


Two common methods

There are two common methods of connection, both of which use Open Database Connectivity (ODBC) drivers:

1. The `RODBC` package.
2. The `DBI` system, `odbc` and also `dplyr` and `dbplyr`.

- Both of these create a connection, using a 'connection string'

- This can be used to create a connection object

- We can use this object to manipulate or pull data into `R`.

1. RODBC

- This is the simpler of the two interfaces, and uses slightly older code.
- It can be used to connect to anything that uses ODBC.

```
library("RODBC")

#Connection string
# e.g. with a server called "Donald" and a database called "Duck" your string
RODBC_connection <- odbcDriverConnect('driver={SQL Server};server=Donald;data

dt1 <- sqlFetch(channel=RODBC_connection, sqtable = "MyTable")
```

```
# Load data from SQL query
dt2 <- sqlQuery(  channel=RODBC_connection
                , query = "select TOP 5 * from MyTable")
dt2
##   id Org year month Category_A Category_B events
## 1  2   A 2015     4      22476      21611    963
## 2  3   B 2015     4      13415      13673    208
## 3  4   C 2015     4       5872       6067    223
## 4  5   D 2015     4       2571       3197    145
## 5  6   E 2015     4       3380       3782    140
```


What is going on here?

- `trusted_connection=true` passes your windows credentials to the server
- You can, instead, specify a username (`uid`) and a password (`pwd`)

- You can also use `RODBC` to write back to database tables, choosing to append or not:

```
sqlSave(  channel = RODBC_connection
        , dat = dt2,
        , tablename = "Mytable_version2"
        , append = FALSE
        , safer = FALSE)
```

Other functions

There are lots of other functions included with `RODBC` to allow you to see structures etc. The package vignette is a very helpful place to go for this, along with the help files.

Remember to disconnect at the end of your session:

```
odbcClose(RODBC_connection)
```

But RODBC isn't my first choice...

2. `DBI\dplyr`

- `DBI` implements a common database interface in `R`.
- Can be used with different 'back-end' drivers such as MySQL, SQL Server, SQLite, Oracle etc.
- Faster than `RODBC` to import data
- Can be used to work with data in the database, without importing it into `R`.
- `DBI` can be used on it's own, but can be combined with `dplyr`, `dbplyr` and use `%>%` to write SQL for you

DBI connection

Requires a different connection string and a few more packages to use:

- `DBI` - a common Database Interface engine for use in `S` and `R` (see here)
- `dplyr` - to make the `tbl` and use it, we'll work with `dplyr` syntax.
- `dbplyr` - this add-on package allows translation from `dplyr` to SQL.
- `odbc`- provides the odbc drivers, but you could use the functions below with other drivers instead.

```
library(DBI)
library(odbc)
library(dplyr)
library(dbplyr)

DBI_Connection <- dbConnect(odbc(),
                            driver = "SQL Server",
                            server=Sys.getenv("SERVER"),
                            database=Sys.getenv("DATABASE")
)
```


Using SQL with DBI

- Can write an SQL query directly using the `dbSendQuery` function.
- Executes the query on the *server-side* only.
- If you want the results back in `R`, you need to use `dbFetch` as well.

```
SomeRecords <- dbFetch(dbSendQuery(DBI_Connection, "Select TOP 100 * from MyT

#or

SomeRecords <- dbSendQuery(DBI_Connection, "Select TOP 100 * from MyTable") %
  dbFetch()
```

Writing to databases

You can also write back to a database using the `dbWriteTable` function.

- For example:
- Writing a new table current connection, called 'NewDatabaseTable'
- Using the `R` `data.frame` called "MyTable_local" (that we created in the last section)
- `append` and `overwrite` options

```
dbWriteTable(DBI_Connection,"NewDatabaseTable", MyTable_local, overwrite=TRUE
```


Using tables in the database

Now we can define a table as if it was part of our `R` work-space, using the connection object and the names of the table in the database.

- Do this with `tbl`
- `glimpse` is a useful function that shows you a summary

```
MyTable<-tbl(DBI_Connection, "MyTable")

glimpse(MyTable)
## Observations: ??
## Variables: 7
## Database: Microsoft SQL Server 14.00.3281[UHB\CSMY@LEE\AKUMA/ISIS]
## $ id         <int> 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
## $ Org        <chr> "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K",
## $ year       <int> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2
## $ month      <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
## $ Category_A <dbl> 22476, 13415, 5872, 2571, 3380, 10711, 6297, 7877, 6613
## $ Category_B <dbl> 21611, 13673, 6067, 3197, 3782, 10161, 7137, 9114, 7402
## $ events     <int> 963, 208, 223, 145, 140, 568, 216, 366, 321, 423, 276,
```

Constructing `dplyr` query

- We can then perform select or aggregate queries without translation.
- Even though it returns results, the data are still in the database

```
MyTable %>%
  filter(year ==2015) %>%
  group_by(month) %>%
  summarise(AvgEvents = mean(events),
            MaxEvents = max(events),
            N = n()) %>%
  arrange(month)
## # Source:     lazy query [?? x 4]
## # Database:   Microsoft SQL Server 14.00.3281[UHB\CSMY@LEE\AKUMA/ISIS]
## # Ordered by: month
##   month AvgEvents MaxEvents     N
##   <int>     <int>     <int> <int>
## 1     4       302       963    25
## 2     5       318       944    25
## 3     6       325      1002    25
## 4     7       324       911    25
## 5     8       305       960    25
## 6     9       314       975    25
```

- `dplyr` can then be used to do fairly complex things in just a few lines.

Using SQL & returning data to `R` (2)

- May need to pull the data from the server into memory in `R` sometimes.
- Can do this with `collect`

```
MyTable_local<- MyTable %>%
  filter(year ==2015) %>%
  group_by(month) %>%
  summarise(AvgEvents = mean(events),
            MaxEvents = max(events),
            N = n()) %>%
  arrange(month) %>%
  collect()

print(MyTable_local)
```

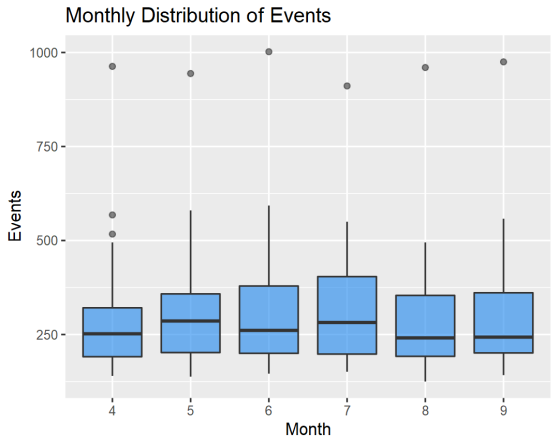
```
## # A tibble: 6 x 4
##   month AvgEvents MaxEvents     N
##   <int>     <int>     <int> <int>
## 1     4       302       963    25
## 2     5       318       944    25
## 3     6       325      1002    25
## 4     7       324       911    25
## 5     8       305       960    25
## 6     9       314       975    25
```

Example:

- I'm filtering the data for 2015 and passing it directly into `ggplot2`

```
library(ggplot2)

MyTable %>%
  filter(year ==2015) %>%
  ggplot(aes(y=events, x=factor(month), group=factor(month))) +
  geom_boxplot(fill = "dodgerblue2", alpha=0.6, )+
  labs(title = "Monthly Distribution of Events", x="Month", y="Events")
```



Useful DBI commands

Command	Summary
dbConnect()	Create a DBI connection object
dbListTables()	List the tables on the connection
dbListFields()	List the fields for a given table on a given connection
dbSendQuery()	Send a query to execute on the server/connection
dbFetch()	Fetch the results from the server/connection
dbWriteTable()	Write a table to the connection
tbl()	Set a table on the connection as a 'tibble' for `dplyr`
glimpse()	See a summary of the rows, data types and top rows

Example script:

Summary

- You don't always want to import data to `R`, keeping it in database is a good idea for many reasons
- `RODBC` is older, but useful interface
- `DBI` is a newer, agnostic, system that works with many difference drivers/systems
- Both require a connection string - You can use RStudio wizard for this too!
- `DBI` has it's own syntax including `dbSendQuery` and `dbFetch` to retrieve results from SQL queries
- `DBI` can also work with `dplyr` by adding `dbplyr` and declaring tables with `tbl`

Thanks for your time!

chris.mainey@uhb.nhs.uk

http://www.hed.nhs.uk

@chrismainey

chrismainey

http://www.mainard.co.uk

`FunnelPlotR` now available on CRAN!