Building and Managing Connections to SQL Server Databases in R

Matthew Sharkey

### Making a connection

R users can integrate SQL server databases using the DBI and ODBC packages. The first step is to build a connection string. A connection string contains a set of key value pairs. It tells R where the server is and what user credential to use. Different driver types need different connection string formats. Here are two common connection string configurations.

library(DBI)  
  
myDriver <- 'SQL Server'  
# Use . for a local connection, otherwise specify Server Machine Name or IP address  
myServer <- '.'  
myDatabase <- 'Cab\_Demo'  
  
  
trusted\_connection <- dbConnect(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database = myDatabase,Trusted\_Connection='yes')  
  
myUserid <- 'Cab\_App'  
myPassword <- Sys.getenv('Cab\_App\_Password')  
  
connection <- dbConnect(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database = myDatabase,Uid = myUserid,Pwd = myPassword)

The trusted\_connection uses my windows account instead of a username and password. I prefer windows logins over creating a SQL login because they are easier to manage. My AD system administrator takes care of securing windows credentials. If I use a SQL login then I’m responsible for guarding and storing the credentials. The second connection string shows how to connect with a SQL login. I use SQL logins for non-windows clients or when connecting to a DB outside the domain. It’s good practice to avoid storing clear text passwords in the client code. So, I stored an environmental variable and accessed it with the Sys.getenv function.

### Common Connection Problems

I’ve encountered a fair share of error when attempting the initial connection. They usually stem from one or more of the following:

* I didn’t build a valid connection string.
  + ConnectionStrings.com is a good place to check for that my connection string is well formed.
* I misspelled something. I botch the server name a lot. In general, I find it easy to misspell keys and values. I once read, “There’re two hard thing in programming - naming conventions and spelling.” I can’t agree more.
* I don’t have permissions to the SQL Server.
  + When I see errors like the “The server principal. is not able to access.” then I suspect permissions are the issue. If I don’t own the server then I submit a ticket to the help desk for authorization. If I own the server then I check the SQL server logs through SQL Server Management Studio. The logs usually direct me to either create a user or grant permissions.
* I don’t have the right driver selected.
  + It’s easy to attempt a connection with a driver that doesn’t exist on the client. Windows users can check available drivers by searching Administrative tools -> Double click Data Sources. If I publish to an external source e.g. ShinyApps.IO then I check their support docs for available driver names. At the time of this blog post they have SQL Server driver named “SQLServer” available.
* I need to specify the port number.
  + Sometimes the default port is changed from 1433. In that case, I must specify the port number after the server name e.g. myServer = ‘.,2050’
* A firewall is blocking the connection.
  + One of my [previous blog posts](https://www.hinttank.com/troubleshooting-client-connection-issues/) shows how to configure a firewall SQL Server access.

### Executing a Query

Now that I’ve made a connection I’ll verify it with a simple GetDate() query.

dbGetQuery(trusted\_connection,"Select GetDate()")

##   
## 1 2019-03-12 23:04:38

dbDisconnect(trusted\_connection)

I closed the connection by passing the connection variable to the dbDisconnect function. Generally, it is a good idea to close connections after use. Leaving connections open wastes memory and blocks resources for other query sessions. Too many open connections can overload the server and prevent new connections. This might crash the application or the database server.

I’ll show what open connections look like on the back-end. First, I’ll make five connections using a for loop without invoking dbDisconnect.

for (i in 1:5)  
{  
trusted\_connection <- dbConnect(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database = myDatabase,Trusted\_Connection='yes')  
date <- dbGetQuery(trusted\_connection,paste("Select GetDate() as mydate,\'leakedquery\' as c1,",i))  
}

Here’s a query that a system admin might use to watch database connections. It returns information about the five open connections. I’ve used this query to look for applications that might be leaking connections. If I a lot of sleeping connections then I contacted the app developer for a patch.

SELECT   
 DB\_NAME(p.dbid) as DBName,   
 program\_name as Program,CPU,memusage,status,SPID,nt\_username  
FROM sys.sysprocesses p  
Cross apply sys.dm\_exec\_sql\_text(p.sql\_handle) d  
WHERE program\_name = 'Rstudio'  
and text like '%leakedquery%' and text not like '%DB\_NAME%'

5 records

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DBName | Program | CPU | memusage | status | SPID | nt\_username |
| Cab\_Demo | RStudio | 0 | 3 | sleeping | 55 | [msharkey3434@gmail.com](mailto:msharkey3434@gmail.com) |
| Cab\_Demo | RStudio | 15 | 3 | sleeping | 56 | [msharkey3434@gmail.com](mailto:msharkey3434@gmail.com) |
| Cab\_Demo | RStudio | 0 | 3 | sleeping | 57 | [msharkey3434@gmail.com](mailto:msharkey3434@gmail.com) |
| Cab\_Demo | RStudio | 0 | 3 | sleeping | 58 | [msharkey3434@gmail.com](mailto:msharkey3434@gmail.com) |
| Cab\_Demo | RStudio | 0 | 3 | sleeping | 59 | [msharkey3434@gmail.com](mailto:msharkey3434@gmail.com) |

I didn’t issue a dbDisconnect so the connections will remain open until the client closes or R runs garbage collection.

### One more connection package to consider

The pool package opens and closes connections automatically. R users establish a connection to a pool. From then on pool gives the query an iddle connection or opens a new connection. Besides simplifying client code, pooled connections can also provide a performance boost. To illustrate, I wrote a sample workload consisting of three queries. Next, I wrote a funtion to execute the queries using dbConnect/dbDisconnect, and a function to execute the queries using a pool.

library(pool)  
  
queries <- c("SELECT Carrier,count(1)  
 FROM [Cab\_Demo].[dbo].[flights]  
 group by Carrier"," Select Avg(distance)  
 From [Cab\_Demo].[dbo].[flights]"," Select top 100 [2012],[2013],[2014],[2015]  
 From [Cab\_Demo].[dbo].[flights]  
 PIVOT ( count(flight)  
 FOR year in ([2012],[2013],[2014],[2015])  
 ) as pvt")  
  
  
dbconnectworkload <- function() {  
 con <- dbConnect(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database = myDatabase,Trusted\_Connection='yes')  
 dbGetQuery(con,queries[1])  
 dbDisconnect(con)  
 con <- dbConnect(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database =myDatabase,Trusted\_Connection='yes')  
 dbGetQuery(con,queries[2])  
 dbDisconnect(con)  
   
 con <- dbConnect(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database =myDatabase,Trusted\_Connection='yes')  
 dbGetQuery(con,queries[3])  
 dbDisconnect(con)  
}  
  
poolcon <- dbPool(odbc::odbc(),Driver= myDriver,Server = myServer  
 ,Database = myDatabase,Trusted\_Connection='yes')  
  
dbpoolworkload <- function() {  
 dbGetQuery(poolcon,queries[1])  
 dbGetQuery(poolcon,queries[2])  
 dbGetQuery(poolcon,queries[3])  
}  
  
results <- microbenchmark::microbenchmark(dbconnectworkload()  
 ,dbpoolworkload(),times = 5)  
  
poolClose(poolcon)  
  
results

## Unit: milliseconds  
## expr min lq mean median uq  
## dbconnectworkload() 6987.7926 6998.5904 7072.3524 7049.6344 7058.3855  
## dbpoolworkload() 771.0828 816.5347 994.6542 841.6587 883.7743  
## max neval  
## 7267.359 5  
## 1660.220 5

The pooled function connection appears to be 10 times faster. Profvis shows that most of time spent in dbconnectworkload is on opening connections.

library(profvis)  
  
profvis({dbconnectworkload ()})

### dbDisconnect(BlogPost)

Database interaction starts with a connection. I mentioned a few common connection problems and how to troubleshoot them. Once I have a connection then I begin testing queries. The DBI function dbGetQuery takes two arguments - 1) the connection 2) a query string. Finally, I mentioned that it’s good to get in the habit of closing connections or use the pool package. Database integration with R is large topic that I will continue to blog about in future posts.