Wikimedia Commons,
Author alaa kaddour

Accessing SQL Databases in R: 3 Approaches

The R Foundation
r-project.org

@StatistikinDD

Accessing SQL Databases: Three Approaches

1. DBI & Backend Package

2. dplyr & dbplyr

3. R Markdown / Notebooks

4. Bonus: Best Practices & More Packages

(1) DBI & Backend Package

- Establish database connection: package ***DBI = Database Interface***
- Use the appropriate backend package

Five common **backends**:

Backend Package	Database(s)
RMySQL	*MySQL* and *MariaDB*
RPostgreSQL	*Postgres* and *Redshift*
RSQLite	*SQLite*
odbc	Many commercial DBs (*open database connectivity protocol*)
bigrquery	Google‘s *BigQuery* (hosted database for big data)

DBI has several **vignettes**, see *help(package = “DBI“)*

(1) DBI & Backend Package

```
> library(DBI)
> library(RMySQL)

> class(con)
[1] "MySQLConnection"
attr(,"package")
[1] "RMySQL"
> (table_names <- dbListTables(con))
[1] "comments" "tweats"   "users"
> users <- dbReadTable(con, "users")
> elisabeth <- dbGetQuery(con, "SELECT tweat_id FROM comments WHERE user_id = 1")
> (latest <- dbGetQuery(con, "SELECT post FROM tweats WHERE date > '2015-09-21'"))
                                                                post
1                  open and crush avocado. add shrimps. perfect starter.
2 nachos. add tomato sauce, minced meat and cheese. oven for 10 mins.
3                               just eat an apple. simply and healthy.
> dbDisconnect(con)
[1] TRUE
> rm(con)
>
```

- Establish database connection (con): e. g. con <- dbConnect(odbc::odbc(), "Oracle DB")
- Write SQL code using ***dbGetQuery()***

  or ***dbSendQuery()***

(2) dplyr & dbplyr

- Use **same syntax**, regardless whether data lives in R‘s memory or in a database

- dbplyr is **lazy**:
  - Never pulls data into R unless explicitly asked for
  - Delays work until last possible moment → sends collected query to database in one step
  - When you ask for data, dbplyr only pulls a few rows of large datasets

➔You can experiment with your code without losing much time on large data operations

- When your code works as expected: **collect()** data
- Use **show_query()** to inspect SQL translation

Caveats:
  - nrow() returns NA
  - tail() doesn‘t work

(2) dplyr & dbplyr

```
flights_db %>% select(year:day, dep_delay, arr_delay)
```

```
## # Source:   lazy query [?? x 5]
## # Database: sqlite 3.35.5 [:memory:]
##     year month   day dep_delay arr_delay
##    <int> <int> <int>     <dbl>     <dbl>
##  1  2013     1     1         2        11
##  2  2013     1     1         4        20
##  3  2013     1     1         2        33
##  4  2013     1     1        -1       -18
##  5  2013     1     1        -6       -25
##  6  2013     1     1        -4        12
##  7  2013     1     1        -5        19
##  8  2013     1     1        -3       -14
##  9  2013     1     1        -3        -8
## 10  2013     1     1        -2         8
## # ... with more rows
```

(2) dplyr & dbplyr

dbplyr supports the following databases:

- Oracle
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- Apache Hive
- Apache Impala

See https://db.rstudio.com/getting-started/database-queries

(3) R Markdown / Notebook

R Notebooks support SQL code chunks like so:

```
```{sql, connection=con, output.var = "mydataframe"}
SELECT "month_idx", "year", "month",
SUM(CASE WHEN ("term_deposit" = 'yes')
THEN (1.0) ELSE (0.0) END) AS "subscribe",
COUNT(*) AS "total" FROM ("bank")
GROUP BY "month_idx", "year", "month"
```
```

Chunk options:

- *max.print = 10* for number of records displayed (set to -1 or NA for no limit)
- tab.cap = "My Caption": caption indicates number of records displayed

- Markdown documents enable you to use different languages within the same file
- Just like specifying an R code block, you can also specify an sql code block using **```{sql,**
- Requires you to first establish a connection (here: **con**), usually via *DBI::dbConnect()*
- Benefit: syntax highlighting
- Find out more about language engines supported by R Markdown: https://bookdown.org/yihui/rmarkdown/language-engines.html

Summary

Method	Benefits
DBI & Backend Package *dbGetQuery()* etc.	• Fewer dependencies required
dplyr syntax	• Use same syntax for R and database objects; especially useful if data sources can vary in the course of a project • No knowledge of SQL required • Code is standard across SQL variants • Lazy evaluation
R Notebook SQL engine	• Copy and paste SQL – no formatting required (in *dbGetQuery()*, quotes may require escaping) • SQL syntax is highlighted

Source: https://db.rstudio.com/getting-started/database-queries

Get More Power: RStudio Professional Drivers

- Available at no extra cost if you use RStudio Professional products:
  RStudio Server Pro / RStudio Connect / Shiny Server Pro

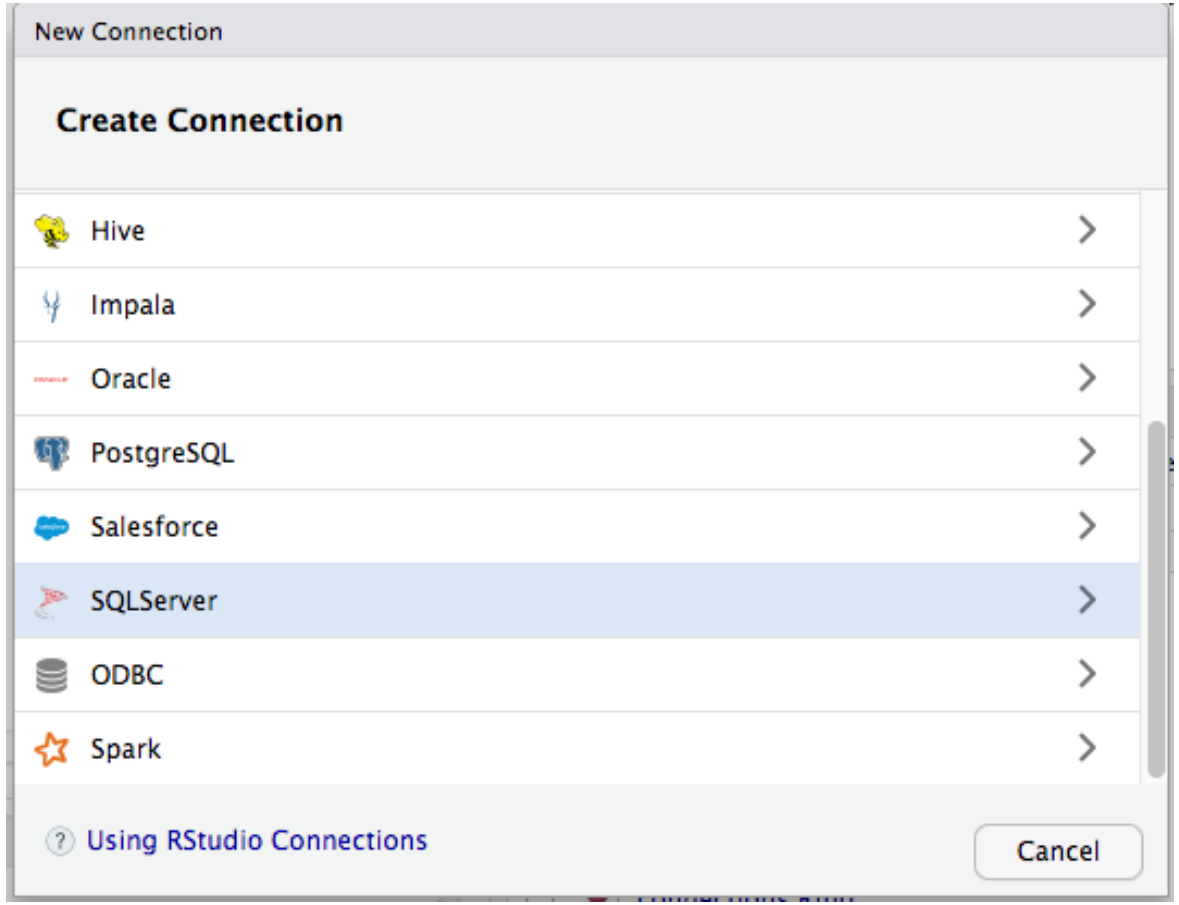
- ODBC drivers for enterprise databases
  - Microsoft SQL Server
  - Oracle
  - Teradata
  - PostgreSQL
  - Apache: Hive, Impala, Cassandra
  - Amazon: Athena, Redshift
  - MongoDB
  - Google BigQuery
  - IBM Netezza
  - Salesforce
  - MySQL

https://db.rstudio.com/rstudio/pro-drivers/

RStudio Connection Pane

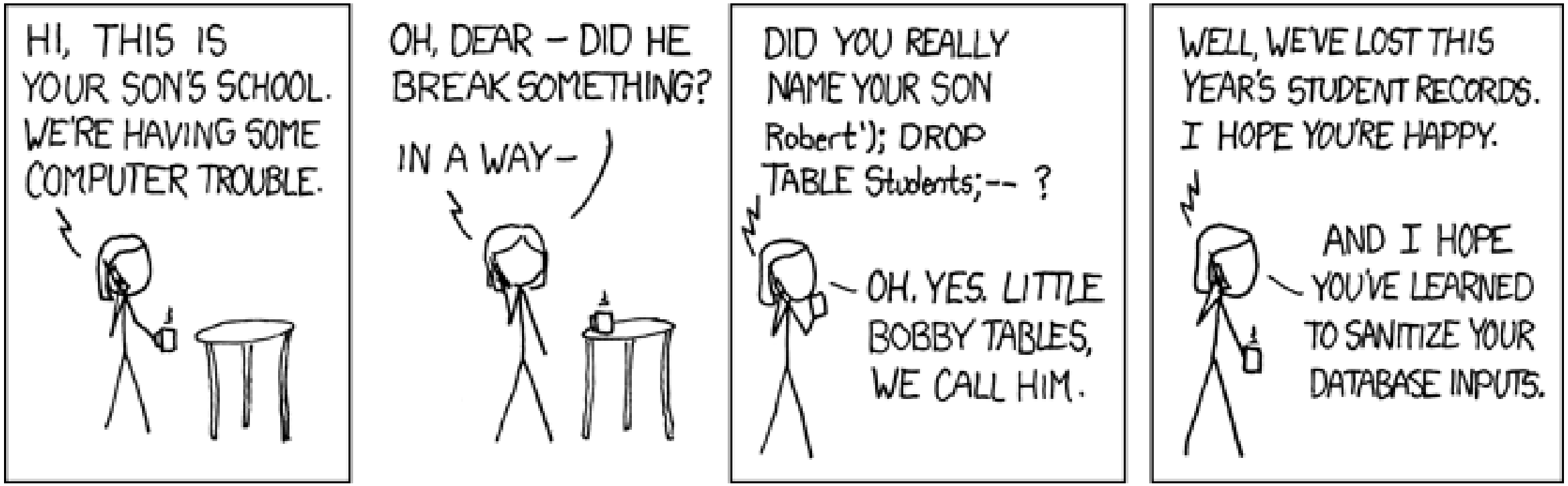
What the New Connection interface can show:

- ODBC DSNs installed on system
- Connections supplied by an administrator:
  can make connections available via **connection snippets**
  in a folder
  https://rstudio.github.io/rstudio-extensions/rstudio-connections.html
- **Packages** supporting connections;
  components: Connections File and Snippet Files
- **Shiny Application** for connection interfaces; sparklyr example:
  https://github.com/sparklyr/sparklyr/blob/master/R/connection_shinyapp.R



Best Practices

Avoid SQL Injection Attacks!



Source: https://xkcd.com/327/

Best Practices

Run Queries Safely

- Avoid DBI::dbGetQuery() to *paste strings containing user input:* Danger of **SQL Injection Attack**
- Better: **Parameterized queries** using placeholders:
  - *dbBind()* and *dbFetch()* or *glue::glue_sql()*
  - *sqlInterpolate()* if *dbBind()* doesn‘t work with a specific database connector
- See more at https://db.rstudio.com/best-practices/run-queries-safely/

Securing Deployed Content: https://db.rstudio.com/best-practices/deployment/

Securing Credentials: https://db.rstudio.com/best-practices/managing-credentials/

- **Never put credentials in plain code.** Better:
- Use **DSN** (Data Source Name) or **encrypt credentials** with the **keyring** package
- Keep credentials in a **YAML file** config.yml, managed by the **config** package
- Use **Environment variables**, see Sys.genenv(), or set **global options** → don‘t publish code that sets options!
- **Prompt for credentials** using *rstudioapi::askForPassword()*

More Packages

R Package	Description
odbc	Faster than the older RODBC package, deals better with dates and times, and still worked on
keyring	Securely store database credentials in system keychain; only decrypt when needed
config	Parameterise database connection credentials: switch between testing database and production database; use YAML file
pool	Manage a shared pool of database connections for **Shiny apps**
rquery	Query generator for R; uses an alternative pipe: “dot pipe“ from wrapr, %.>% Learn more: *vignette("PipeableSQL", package = "rquery");* there are more vignettes: *help(package = “rquery“)* https://github.com/WinVector/rquery/blob/master/README.md
rqdatatable	implementation of the rquery piped Codd-style relational algebra hosted on **data.table** https://github.com/WinVector/rqdatatable