STAT 209
SQL Intro

November 14, 2019

Colin Reimer Dawson

What is SQL?

- Stands for "Structured Query Language"
- An old programming language used to interact with databases
- `dplyr` is based on it

Do we still need SQL?

- ... maybe

Do we still need SQL?

- ... maybe
- It's old, but modern data science applications are based on it

Do we still need SQL?

- ... maybe
- It's old, but modern data science applications are based on it
- When datasets are bigger than we can fit into RAM but not too big to fit on disk (tens of GB), SQL lets us read just the part of the data we need into memory

Using SQL "Statically" to read data to R

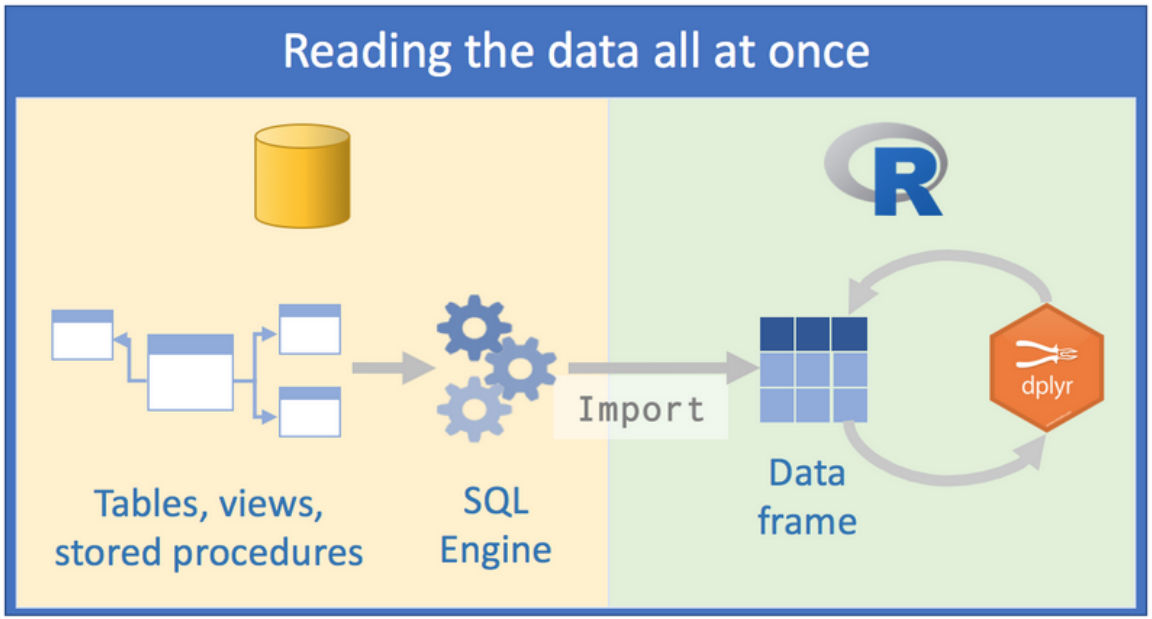

Image Source: `https://rviews.rstudio.com/2017/05/17/databases-using-r/`

Using SQL "Dynamically" to interact with data in R

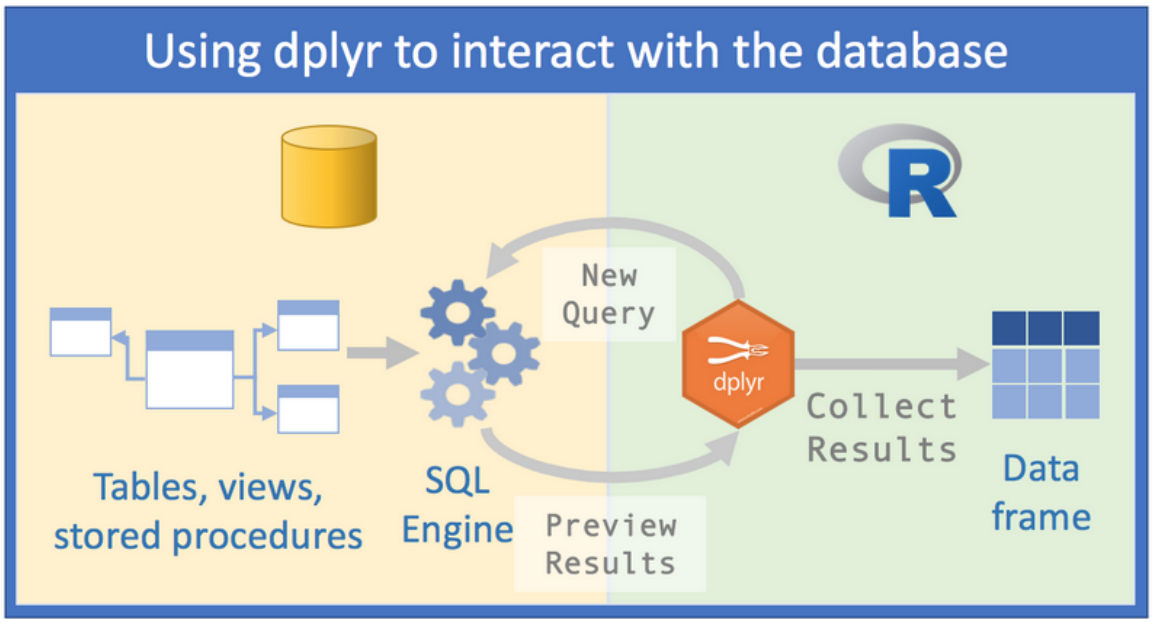

Image Source: `https://rviews.rstudio.com/2017/05/17/databases-using-r/`

SQL with a `dplyr()` "wrapper"

Demo

MySQL Workbench

- Optional, but potentially useful for browsing available data, and tweaking/debugging queries before copying/pasting into R
- Though note: RStudio/RMarkdown supports code chunks written in SQL, with syntax highlighting, which can be executed directly from RStudio (but results can't be saved as R objects)
- Install MySQL Workbench here: `https://dev.mysql.com/downloads/workbench/`

Outline

Main SQL Verbs (copied from MDSR book)

`SELECT` allows you to list the columns, or functions operating on columns, that you want to retrieve. This is an analogous operation to the `select()` verb in `dplyr`, potentially combined with `mutate()`.

`FROM` specifies the table where the data are.

`JOIN` allows you to stitch together two or more tables using a key. This is analogous to the `join()` commands in `dplyr`.

`WHERE` allows you to filter the records according to some criteria. This is an analogous operation to the `filter()` verb in `dplyr`.

`GROUP BY` allows you to aggregate the records according to some shared value. This is an analogous operation to the `group_by()` verb in `dplyr`.

`HAVING` is like a `WHERE` clause that operates on the result set—not the records themselves. This is analogous to applying a second `filter()` command in `dplyr`, after the rows have already been aggregated.

`ORDER BY` is exactly what it sounds like—it specifies a condition for ordering the rows of the result set. This is analogous to the `arrange()` verb in `dplyr`.

`LIMIT` restricts the number of rows in the output. This is similar to the R command `head()`, but somewhat more versatile.

Note: unlike dplyr, verbs cannot appear out of the above order

dplyr to SQL correspondences

Concept	SQL	R

Filter by rows & columns	`SELECT col1, col2`
	`FROM a`
	`WHERE col3 = 'x'`
		`a %>%`
		`filter(col3 == "x")`
		`%>%`
		`select(col1, col2)`

Aggregate by rows	`SELECT id, sum(col1)`
	`FROM a`
	`GROUP BY id`
		`a %>%`
		`group_by(id) %>%`
		`summarize(sum(col1))`

Combine two tables	`SELECT *`
	`FROM a`
	`JOIN b ON a.id = b.id`
		`a %>%`
		`inner_join(b, by =`
		`c("id" = "id"))`

Table 12.1: Equivalent commands in SQL and R, where a and b are SQL tables and R `data.frames`.

Also copied from MDSR

Note that queries always have a `SELECT ... FROM` pattern to start with

Lab 12