SQL hints, hurdles, and help

USCOTS 2025 breakout session

Nicholas Horton (nhorton@amherst.edu) and Jo Hardin (jo.hardin@pomona.edu)

July 18, 2025

SQL hints

SQL bits

- A **SQL** *query* is a **SQL** *statement* that typically starts in `SELECT` and ends in `;`
- **SQL** *keywords* are like data verbs in the **tidyverse**
- A **SQL** *clause* is the keyword + relevant information. E.g., `GROUP BY Dest` or `WHERE ("Year" = 2024.0 AND ((Origin = 'LAX' AND Dest = 'BOS') OR (Origin = 'BOS' AND Dest = 'LAX')))`
- Exactly one **SQL** *statement* can be sent to the remote database in a **SQL** chunk.

Order of SQL keywords

Queries in **SQL** start with the `SELECT` keyword and consist of several keywords, which *must* be written in the following order:

- `SELECT`
- `FROM`
- `JOIN`
- `WHERE`

- `GROUP BY`
- `HAVING`
- `ORDER BY`
- `LIMIT`

The keywords are similar to data wrangling verbs in **R**, but the order in **SQL** is super important!

Differences in dialects

Let's say we want to combine a student's name and grade into a single text string by joining variables.

DuckDB

```
SELECT name || ' got a ' || grade AS result
FROM students;
```

SQLite

```
SELECT name || ' got a ' || CAST(grade AS TEXT) AS result
FROM students;
```

MySQL

```
SELECT CONCAT(name, ' got a ', grade) AS result
FROM students;
```

Median and quantiles

Certain operations are highly optimized for databases, but some aren't.

As an example **MySQL** doesn't have a function for calculating a median, but **duckDB** does.

noSQL and SQL

- Since their initial development, there have been many new technologies to address particular use cases.
- noSQL: different from relational databases in that they access data using `key-value` pairs.
- example: MongoDB
- attractive for very large data stores and clustered applications
- sometimes trade off "fixed consistency" for "eventual consistency" (think your cart in Amazon)

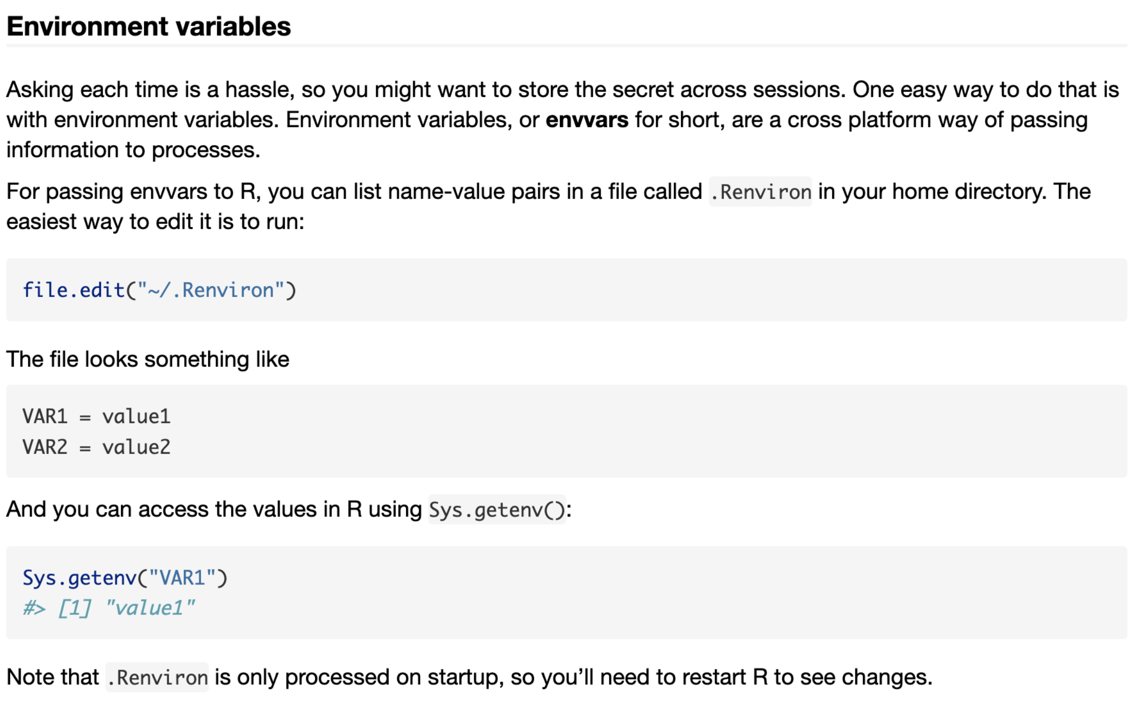
SQL connection

To set up a **SQL** connection, you need the location of the server (`host`) as well as a `user`name and `password`. For example, you may want to use the subset of of data from 2013 to 2015 which exists in a **SQL** database hosted by Ben Baumer used in *Modern Data Science in R*.

~~~
```{r}
con_mysql <- DBI::dbConnect(
  RMariaDB::MariaDB(),
  dbname = "airlines",
  host = Sys.getenv("MDSR_HOST"),
  user = Sys.getenv("MDSR_USER"),
  password = Sys.getenv("MDSR_PWD")
)
```
~~~

Keeping connections private

Hadley Wickham discusses how to use `Sys.getenv()` so that login information is kept private: https://cran.r-project.org/web/packages/httr/vignettes/secrets.html



Environment variables

Asking each time is a hassle, so you might want to store the secret across sessions. One easy way to do that is with environment variables. Environment variables, or **envvars** for short, are a cross platform way of passing information to processes.

For passing envvars to R, you can list name-value pairs in a file called `.Renviron` in your home directory. The easiest way to edit it is to run:

```
file.edit("~/.Renviron")
```

The file looks something like

```
VAR1 = value1
VAR2 = value2
```

And you can access the values in R using `Sys.getenv()`:

```
Sys.getenv("VAR1")
#> [1] "value1"
```

Note that `.Renviron` is only processed on startup, so you'll need to restart R to see changes.

Other public facing SQL servers

- MDSR data (`airlines` and `imdb` databases) : https://mdsr-book.github.io/mdsr3e/15-sqlI.html
- Bioinformatics data: https://genome.ucsc.edu/FAQ/FAQdownloads#download29
- Audiology data: https://www.science.smith.edu/wai-database/home/about

Setting up your own SQL server

- buy computing space
  - Microsoft Azure
  - Amazon Web Services (AWS)
  - Google Cloud Platform (GCP)
  - A2 Hosting
  - Digital Ocean
- does the space already have a SQL server installed?
- upload data
  - variable types
  - keys
  - connections between tables
  - security measures

Intructions for uploading the Stanford Open Policing Project data to a SQL server.

(Or use a serverless setup.)

When are databases important?

Advantages to using a database

- remote
- can hold arbitrarily large amounts of information
- multiple people can access simultaneously

Disadvantages to using a database

- painful to set up a remote server
- expensive to set up a remote server
- can’t model or plot

THANK YOU!

Nicholas Horton (nhorton@amherst.edu)

Jo Hardin (jo.hardin@pomona.edu)