USCOTS 2025 breakout session: Explore the airlines data using SQL and parquet

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

2025-07-11

Table of contents

Introduction . . . 1
Check for files . . . 2
Check reading via DuckDb . . . 2
Accessing databases using dplyr . . . 2
Which destinations are most delayed? . . . 2
How many flights are there each month? . . . 3
Extension . . . 5
Closing the SQL connection . . . 6

Introduction

This file analyzes airline flight data from the American Statistical Association's Data Expo 2024. Once the `1_download_data.qmd` Quarto file has been successfully rendered, you should be able to render this file (`3_explore_dplyr.qmd`) which shows off dplyr syntax while analyzing the downloaded data.

See https://community.amstat.org/dataexpo/home for more information on the data.

See https://beanumber.github.io/abdwr3e/12-large.html and https://mdsr-book.github.io/mdsr3e/15-sqlI.html for resources on databases in R.

See https://hardin47.netlify.app/courses/sds261-sql/ for an accessible overview of SQL and databases.

Check for files

First we check that the files are where we expect. If you run the code below with no errors, you are ready to go! (If you run into problems, try rendering the file or "Change Working Directory" to "File Location" under the "Session" Menu in RStudio.

```
folder_name <- "data_airlines"
stopifnot(file.exists(folder_name))
stopifnot(file.exists(paste0(folder_name, "/Year=2024/data_0.parquet")))
```

Check reading via DuckDb

We begin by creating an in-memory database using DuckDb. This is just a placeholder that we can reference.

```
con_duckdb <- DBI::dbConnect(duckdb::duckdb())
```

Accessing databases using dplyr

Here we use a `tbl` (like a `tibble`, but it lives remotely) to create a shadow data frame from which we can query using **dplyr** wrangling verbs. Note that the work done on the `tbl` **feels** just like working on a `tibble`, but the `tbl` object does not live in your local R environment.

```
flights_duckdb <- tbl(
  con_duckdb,
  paste0("read_parquet('", folder_name, "/Year*/*.parquet')"))
object.size(flights_duckdb |> filter(Year == 2024, Month == 3))
```

```
79056 bytes
```

```
object.size(flights_duckdb |> filter(Year == 2024, Month == 3) |> collect())
```

```
522179168 bytes
```

Which destinations are most delayed?

The following query uses the `flights_duckdb` object to group by destination and summarize the number of flights and average arrival delay.

```
flights_duckdb |>
  group_by(Dest) |>
  summarise(n = n(),
            avg_delay = mean(ArrDelay, na.rm = TRUE),
            .groups = "drop") |>
  arrange(desc(avg_delay)) |>
  head(10)
```

```
# Source:     SQL [?? x 3]
# Database:   DuckDB v1.3.2 [nhorton@Darwin 24.5.0:R 4.5.0/:memory:]
# Ordered by: desc(avg_delay)
   Dest      n avg_delay
   <chr> <dbl>     <dbl>
 1 HOB      78      92.3
 2 PVU    3003      31.6
 3 SMX     160      26.4
 4 PSE    1675      22.3
 5 BQN    4267      20.5
 6 OWB      43      19.0
 7 SFB   14023      18.1
 8 CMX    1068      17.9
 9 ASE   10899      17.6
10 PRC    1087      17.3
```

We note that the largest delays tend to be for airports that have relatively few flights (e.g., `HOB` is Lea County Regional Airport near Hobbs, Nevada, Wikipedia).

An exception to this rule is Orlando Airport (`SFB`, https://flysfb.com), which has a large number of flights and relatively large average delay.

How many flights are there each month?

```
aggregation_query <- flights_duckdb |>
  filter(Month %in% c(1,2,3,4,5)) |>
  group_by(Month, Year) |>
  summarise(
    n = n(),
    avg_delay = mean(ArrDelay, na.rm = TRUE),
    .groups = "drop"
  ) |>
```

```
  arrange(Year, Month)

aggregation_query
```

```
# Source:     SQL [?? x 4]
# Database:   DuckDB v1.3.2 [nhorton@Darwin 24.5.0:R 4.5.0/:memory:]
# Ordered by: Year, Month
   Month  Year      n avg_delay
   <dbl> <dbl>  <dbl>     <dbl>
 1     1  2023 538837     7.78
 2     2  2023 502749     4.14
 3     3  2023 580322     9.07
 4     4  2023 561441     9.11
 5     5  2023 579958     3.81
 6     1  2024 547271    10.4
 7     2  2024 519221     0.593
 8     3  2024 591767     6.50
 9     4  2024 582185     5.36
10     5  2024 609743    14.0
```

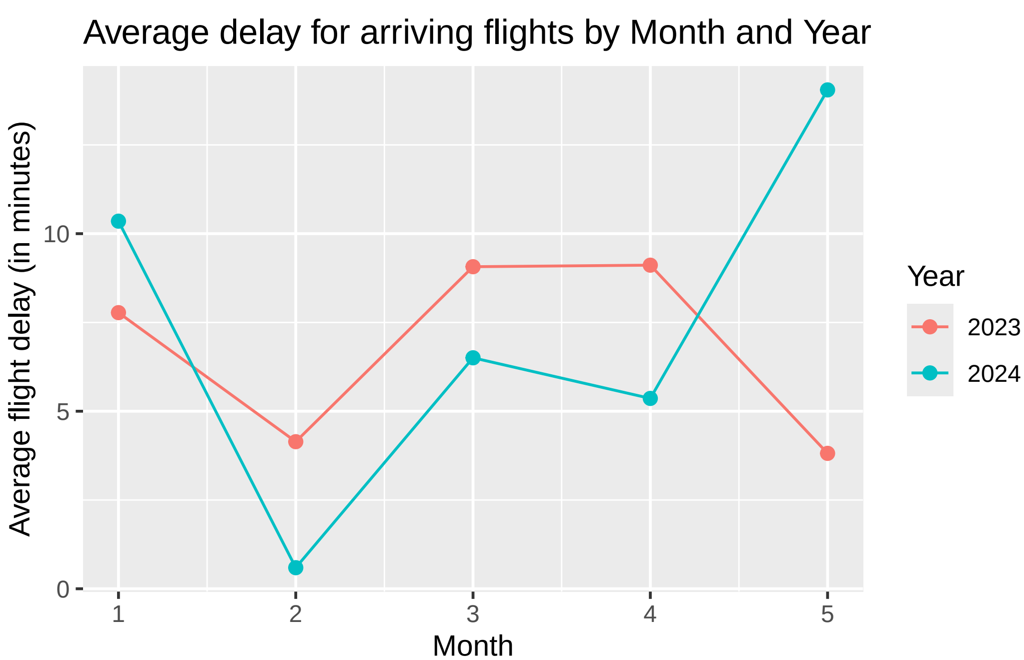
```
show_query(aggregation_query)
```

```
<SQL>
SELECT "Month", "Year", COUNT(*) AS n, AVG(ArrDelay) AS avg_delay
FROM (
  SELECT q01.*
  FROM (FROM read_parquet('data_airlines/Year*/*.parquet')) q01
  WHERE ("Month" IN (1.0, 2.0, 3.0, 4.0, 5.0))
) q01
GROUP BY "Month", "Year"
ORDER BY "Year", "Month"
```

Even though the `tbl` (and the results from the SQL query) are remote objects that do not live in your environment, you can use them as if they were local, for example, by inputting the `tbl` into a ggplot. Here we both run the query and provide the translation to the underlying SQL call (see `2_explore_sql.qmd` for more SQL examples).

```
flights_duckdb |>
  filter(Month %in% c(1,2,3,4,5)) |>
  group_by(Month, Year) |>
```

```
summarise(
  n = n(),
  avg_delay = mean(ArrDelay, na.rm = TRUE),
  .groups = "drop"
) |>
arrange(Year, Month) |>
ggplot(aes(x = Month, y = avg_delay, color = as.factor(Year))) +
labs(
  title = "Average delay for arriving flights by Month and Year",
  x = "Month",
  y = "Average flight delay (in minutes)",
  color = "Year"
) +
geom_point(size = 2) +
geom_line()
```



Extension

Use the following code chunk to answer your own question about these data using the `dplyr` interface.

Closing the SQL connection

It is always good practice to close your connection when you are through with it (this is particularly important if you are accessing a remote database).

```r
DBI::dbDisconnect(con_duckdb)
```