Converting lazy data frames into Parquet files

Ian D. Gow

22 March 2026

Abstract

In this short note, I show how my `db2pq` package can be used to turn "lazy data frames" produced by `dbplyr` into Parquet files on disk without needing to load the data into RAM and with minimal RAM requirements.

> **Tip**
>
> This note was written and rendered with Quarto. The source code for this note is available here. In writing this note, I used the packages listed below.[1]
> I have not submitted `db2pq` to CRAN. To install `db2pq`, use `pak` by typing `pak::pak("iangow/db2pqr")` in the R console. The `db2pq` is located at `iangow/db2pqr` because `iangow/db2pq` was already taken by my Python `db2pq` package.

```
library(tidyverse)
library(DBI)
library(arrow)
library(farr)
library(db2pq)
library(tidyfinance)
library(frenchdata)
library(dbplyr)
```

1 Motivating example

According to Tidy Finance for R:

1. Execute `install.packages(c("tidyverse", "farr", "DBI", "arrow", "pak", "tidyfinance", "frenchdata", "dbplyr")` within R to install all the packages [other than `db2pq`] that you will need to run the R code in this note.

> The daily CRSP data file is substantially larger than monthly data and can exceed 20 GB. This has two important implications: you cannot hold all the daily return data in your memory (hence it is not possible to save the entire dataset to your local folder), and in our experience, the download usually crashes (or never stops) because it is too much data for the WRDS cloud to prepare and send to your R session.
>
> There is a solution to this challenge. As with many big data problems, you can split up the big task into several smaller tasks that are easier to handle. That is, instead of downloading data about all stocks at once, download the data in small batches of stocks consecutively.

Below I show how one can simplify the code dramatically (no batches!) and download the data faster (for me, it takes *less than a minute*) and with no significant burden on either the CPU or RAM.

The `lazy_tbl_to_pq()` just uses functions supplied by the `DBI` and `arrow` packages, so should work with any **lazy data frame** using those tools Please try out the package and feel free to log a GitHub issue if you have any problems with it.

1.1 Getting Fama-French data

While the `ff` schema on WRDS seems to have these data, I follow Tidy Finance in using `frenchdata`, which I presume gets the data from Ken French's website.

I use the same date range as I found on the Tidy Finance page.

```
start_date <- ymd("1960-01-01")
end_date <- ymd("2024-12-31")
```

Like Tidy Finance, I save the data as a Parquet file.

```
factors_ff3_daily_raw <- download_french_data("Fama/French 3 Factors [Daily]")

factors_ff3_daily <-
  factors_ff3_daily_raw$subsets$data[[1]] |>
  mutate(
    date = ymd(date),
    across(c(RF, `Mkt-RF`, SMB, HML), ~ as.numeric(.) / 100),
    .keep = "none"
  ) |>
  rename_with(str_to_lower) |>
  rename(mkt_excess = `mkt-rf`, risk_free = rf) |>
  filter(between(date, start_date, end_date)) |>
  write_parquet("factors_ff3_daily.parquet")
```

1.2 Getting CRSP daily data

Now that I have the daily Fama-French factor data, I can move onto the main query. The following is based on code in Tidy Finance.

Apart from a little tidying, I remove the batch infrastructure and just get all data at once and append a `lazy_tbl_to_pq()` at the end (along with a `system_time()` to see how long this all takes).[2]

Note that I use `dbplyr::copy_inline()` to get the local `factors_ff3_daily` to WRDS. I wrote about `copy_inline()` in a note in December 2025.

```
# set_wrds_credentials()
wrds <- get_wrds_connection()

dsf_db <- tbl(wrds, I("crsp.dsf_v2"))
stksecurityinfohist_db <- tbl(wrds, I("crsp.stksecurityinfohist"))

factors_ff3_daily <-
  read_parquet("factors_ff3_daily.parquet") |>
  copy_inline(wrds, df = _)

crsp_daily <-
  dsf_db |>
  filter(between(dlycaldt, start_date, end_date)) |>
  inner_join(
    stksecurityinfohist_db |>
      filter(
        sharetype == "NS",
        securitytype == "EQTY",
        securitysubtype == "COM",
        usincflg == "Y",
        issuertype %in% c("ACOR", "CORP"),
        primaryexch %in% c("N", "A", "Q"),
        conditionaltype %in% c("RW", "NW"),
        tradingstatusflg == "A",
      ) |>
      select(permno, secinfostartdt, secinfoenddt),
    join_by(permno)
  ) |>
  filter(between(dlycaldt, secinfostartdt, secinfoenddt)) |>
  select(permno, date = dlycaldt, ret = dlyret) |>
  filter(if_all(everything(), \(x) !is.na(x))) |>
```

2. For example, because I don't `collect()` the data I cannot use `drop_na()` and need to use `filter(if_all(everything(), \(x) !is.na(x)))` instead.

```
  left_join(
    factors_ff3_daily |> select(date, risk_free),
    join_by(date),
  ) |>
  mutate(
    ret_excess = ret - risk_free,
    # Note that excess returns are *not* bound below by -1 ...
    # ... but I follow Tidy Finance here.
    ret_excess = pmax(ret_excess, -1, na.rm = TRUE)
  ) |>
  select(permno, date, ret, ret_excess) |>
  lazy_tbl_to_pq(out_file = "crsp_daily.parquet") |>
  system_time()
```

```
   user  system elapsed
 27.798   3.085  62.889
```

```
dbDisconnect(wrds)
```

Now we have downloaded the data, let's take a look.

```
crsp_daily <- read_parquet("crsp_daily.parquet")
crsp_daily
```

```
# A tibble: 72,394,137 x 4
   permno date           ret ret_excess
    <int> <date>       <dbl>      <dbl>
 1  10000 1986-01-08 -0.0244    -0.0247
 2  10000 1986-01-09  0         -0.0003
 3  10000 1986-01-10  0         -0.0003
 4  10000 1986-01-13  0.05       0.0497
 5  10000 1986-01-14  0.0476     0.0473
 6  10000 1986-01-15  0.0455     0.0452
 7  10000 1986-01-16  0.0435     0.0432
 8  10000 1986-01-17  0         -0.0003
 9  10000 1986-01-20  0         -0.0003
10  10000 1986-01-21  0         -0.0003
# i 72,394,127 more rows
```

For larger Parquet data files, I would recommend using DuckDB or Arrow's `open_dataset()` function. The `parquet` tabs of "Empirical Research in Accounting: Tools and Methods" use an in-memory DuckDB database connection to work with Parquet files.