Some benchmarks with `comp.g_secd`

Ian D. Gow

21 January 2026

In this note, I use a simple query to benchmark performance of various approaches to accessing data in `comp.g_secd`. Many international researchers will know that `comp.g_secd` holds daily security-level data for non-US firms covered by Compustat Global. It is the global version of the North America daily security file, `comp.secd`, and, I guess, the closest analogue of `crsp.dsf` for finance and accounting researchers studying non-US firms.

I use this data set to do some benchmarking. I find that a query using `comp.g_secd` that takes 6 minutes using SAS on the WRDS servers, takes about 1 minute using the WRDS PostgreSQL server and about 0.2 seconds using a local Parquet file. The Parquet file occupies less than 4 GB on my hard drive, which compares with about 145 GB for the SAS file on the WRDS server. While creating the Parquet file takes 45 minutes, this may be a reasonable trade-off for a researcher who is analysing `comp.g_secd` frequently and does not need the very latest iteration of `comp.g_secd` for research purposes.

> **Tip**
>
> In writing this note, I used the R packages listed below.[1] I also used Python for portions of the code. This note was written using Quarto and compiled with RStudio, an integrated development environment (IDE) for working with R and Python. The source code for this note is available here and the latest version of this PDF is here.

```
library(DBI)
library(tidyverse)
library(farr)
```

1 A little about WRDS data

In the beginning was the ... SAS data. Well, not really. Surely the databases such as CRSP and Compustat that have powered research in accounting and finance since the beginning of time predate SAS

1. Execute `install.packages(c("tidyverse", "DBI", "duckdb", "dbplyr", "farr", "RPostgres")` within R to install all the packages you need to run the code in this note.

and WRDS. But since long before I entered a business PhD program, WRDS has been the preeminent provider of data to academic researchers and SAS was long the form in which WRDS provided data.

In 2011, I created a predecessor of my `wrds2pg` Python package to create a PostgreSQL database containing WRDS data.[2] The `wrds2pg` package has developed over time, but almost nothing has changed since 2023.[3]

By early 2017, WRDS offered its own PostgreSQL database. Early on the data in the WRDS PostgreSQL database was not complete and it seemed to have been created from the SAS data files, but generally was of lower quality than what I had in my database (e.g., `wrds2pg` did a better job with inference of data types and handled non-Latin characters better). But today it is unclear whether the SAS data or the PostgreSQL data are the "canonical" version of the data; they can probably both be considered canonical data sources.

Working in R with data in databases such as PostgreSQL has been greatly facilitated by the `dbplyr` Tidyverse package. I don't recall exactly when I started using `dbplyr`, but I have email exchanges with Hadley Wickham about backend functionality for PostgreSQL as far back as 2015 and I can barely remember the days when I didn't have it. I would say that `dbplyr` is what made the code approach used in Gow and Ding (2024) feasible, even if some critical pieces fell into place after work on the book began. Today, packages such as Ibis mean that a similar approach could be taken with Python.[4]

In late 2023, I added functionality to `wrds2pg` that allows SAS files to be turned into Parquet files. The Parquet file format is provided by Apache Arrow, "a software development platform for building high performance applications that process and transport large data sets."[5] The Parquet format offers much of the versatility of CSV files, while also allowing for a rich system of data types and random access to data. As seen in Chapter 23 of Gow and Ding (2024), Parquet files can be combined with DuckDB to provide a lightweight, high-performance alternative to PostgreSQL

Also in late 2023, I created the `db2pq` Python library to create Parquet files directly from a PostgreSQL database. Around the same time, I updated the online version of Gow and Ding (2024) to include a Parquet variant of the code and I used this approach when teaching courses based on the book in 2024. While the `db2pq` package has a mere 29 commits on [GitHub] at the time of writing, it has seen more recent development than the `wrds2pg` package.[6]

2. My original code was in Perl, but by 2015 I had Python code to do the same job, and by 2019 there was an installable Python package, created with the help of Jingyu Zhang. Early on, I considered SQLite and MySQL as well as PostgreSQL, but I decided on PostgreSQL because of its rich type system, powerful SQL, and support for server programming (though I ended up not using the last one much). Though I had zero expertise, it seems that I somehow made the right choice. I was also an early adopter of RStudio in 2010 or 2011.
3. The only recent change was I recently exposed the (previously internal) `sas_to_pandas()` function that I used in my recent note on SAS and pandas.
4. Trying to do Gow and Ding (2024) using pandas and SQL would be painful and slow.
5. See the Apache Arrow website at https://arrow.apache.org/overview/.
6. A lot of early work involved using poorly documented libraries to keep the memory impact to a minimum and to facilitate the `wrds_update_pq()` conditional-update functionality.

So in this note, I will compare three alternative approaches to `comp.g_secd`: SAS data (using SAS), PostgreSQL data (using R), and Parquet data (using R).[7]

With that short detour into the history of WRDS data and so on out of the way, I now return to the topic of `comp.g_secd` and I focus initially on the "original" SAS data.

2 Exploration of `comp.g_secd` (SAS data)

While researchers looking for security returns for US-listed firms data can use CRSP's `crsp.dsf`, CRSP does not cover non-US firms. Fortunately, `comp.g_secd` provides high-quality data and (importantly) merging with Compustat fundamental data (i.e., `comp.g_funda`) is facilitated by the shared identifiers across these tables (i.e., `gvkey` and `iid`).

The fields on `comp.g_secd` include its security identifiers. The firm identifier is `gvkey` and the issue identifier—crucial for firms with multiple share classes or listings—is `iid`. As I will confirm below, the natural **primary key** for `comp.g_secd` is `(gvkey, iid, datadate)`.

There are two files on the WRDS server for `comp.g_secd`: `g_secd.sas7bdat` (144.8 GB) and `g_secd.sas7bndx` (23.1 GB).[8] The `g_secd.sas7bdat` file is the SAS data file that will be familiar to users of SAS. But even SAS users may be unfamiliar with `g_secd.sas7bndx`, which is the index file. Index files have much the same role for SAS data files as indexes do for tables in SQL databases, such as the WRDS PostgreSQL database that I will discuss in a moment.

If a SAS user copies `g_secd.sas7bdat` onto her computer *without* the index, then query performance is likely to be degraded in many cases. We can inspect the indexes that WRDS has created using the `PROC CONTENTS` procedure from SAS. If you have my package `wrds2pg` installed, then `proc_contents()` provides a convenient way to access this procedure in Python:[9]

```
from wrds2pg import proc_contents
proc_contents("g_secd", "comp")
```

The output from the Python code above is shown in Listings 1, 2, and 3 at the end of this document. The information on the indexes is provided at the bottom of Listing 3, where you can see that there are five indexes. Four indexes relate to single fields: `isin`, `sedol`, `exchg`, and `fic`. Each of these indexes improves performance of queries that focus on particular values of the associated variable. The fifth index is on `gvkey` and `datadate` and allows one to quickly zero in on particular combinations of those variables, as would be the case if merging with a subset of `comp.g_funda` for which `gvkey` and `datadate` is a primary key.[10]

7. Note that the "using R" part is relatively unimportant. For example, it would be easy to do everything with essentially identical performance using Python.
8. If you look at Listing 1, you may notice "135GB"; this is incorrect and should be "135 GiB" (a GiB is a "binary" unit in which 1 GiB = 2^{30} = 1,073,741,824 bytes).
9. Note that the following Python code use the environment variable `WRDS_ID`. Call `proc_contents("g_secd", "comp", wrds_id="your_wrds_id")` if you don't have this set.
10. For example, analogues of `funda_mod` as described in Chapter 6 of Gow and Ding (2024).

2.1 The benchmark query

Now that we understand a little about `comp.g_secd` and its SAS manifestation, I will introduce my benchmark query. The idea of the benchmark is that it is somewhat representative of the kinds of things a researcher might want to do with `comp.g_secd` without being overly complicated. Additionally, the benchmark should require actually looking at a significant part of the data (in this case *all* records) and, to keep it interesting, it should not be able to use a short cut of simply looking at an index.[11]

The SAS version of my benchmark query simply counts the number of rows associated with each value of `curcdd`, which Listing 2 tells us represents "ISO Currency Code - Daily":

```
proc sql;
    CREATE TABLE curcdd_counts AS
    SELECT curcdd, count(*) AS n
    FROM comp.g_secd
    GROUP BY curcdd
    ORDER BY n DESC;
quit;

proc print data=curcdd_counts;
run;
```

I put this SAS code into a file `qsas dsf_to_pd.sas` in my home directory on the WRDS server and ran `qsas g_secd.sas`. Inspecting `g_secd.log` a few minutes later, I see `The SAS System used: real time 6:08.66`. So SAS needs more than 5 minutes to run this query. And here is a sample of the output in `g_secd.lst`:

```
                          The SAS System                                1
                                  Wednesday, January 21, 2026 12:21:00 PM

                      Obs    curcdd          n

                        1      EUR      47691233
                        2      JPY      31756491
                        3      CNY      25462741
                        4      GBP      20561720
                        5      INR      20555693
                        6      KRW      15582955
                        7      HKD      14632508
                        8      AUD      13752538
```

11. Whether queries can use such shortcuts obviously depends on the specifics of the available indexes (in this case, there is no index to use for `curcdd`), but also on whether the backend system will use them for the query.

3 Using the WRDS PostgreSQL server

Now, let's do the same query using the WRDS PostgreSQL server. I have my WRDS ID in the `WRDS_ID` environment variable and I have my password stored in `~/.pgpass`, so I can use PostgreSQL-related environment variables to point R to the WRDS server.

```
Sys.setenv(PGHOST = "wrds-pgdata.wharton.upenn.edu",
           PGPORT = 9737L,
           PGUSER = Sys.getenv("WRDS_ID"),
           PGDATABASE = "wrds")
```

Now, two lines of code suffice to set up a connection to `comp.g_secd` on the WRDS PostgreSQL server.

```
db <- dbConnect(RPostgres::Postgres())

g_secd <- tbl(db, Id(schema = "comp", table = "g_secd"))
```

With `dbplyr`, we can write the query as follows.[12]

```
g_secd |>
  count(curcdd) |>
  arrange(desc(n)) |>
  collect() |>
  system_time()
```

```
   user  system elapsed
  0.014   0.017 150.921
```

```
# A tibble: 115 x 2
   curcdd        n
   <chr>     <int64>
 1 EUR    48037544
 2 JPY    31823971
 3 CNY    25635258
 4 INR    20653714
 5 GBP    20619033
 6 KRW    15641761
 7 HKD    14697439
 8 AUD    13800095
```

12. Note that `system_time()` comes from the `farr` package.

```
 9 TWD    11646012
10 THB     8980840
# i 105 more rows
```

We see two things here. First, the output is slightly different from that above, suggesting that the SAS data are slightly newer (higher numbers) than the PostgreSQL data. I will return to this point below.

Second, the PostgreSQL server delivers the query about six times faster than SAS does.

4 Using Parquet data

Now, I do it again using Parquet data. The first step is to update (or get the data) using the following two lines of Python code. Again, this code relies on `WRDS_ID` and `DATA_DIR` environment variables. I specify `archive=True` to keep a copy of any existing `g_secd.parquet` file I have.

```
from db2pq import wrds_update_pq
wrds_update_pq("g_secd", "comp", archive=True)
```

```
comp.g_secd already up to date.
False
```

When I did this update earlier today, it took a bit over 45 minutes on my computer. But, as can be seen above, `wrds_update_pq()` will not fetch new data if the data are up to date.

However, WRDS updates Compustat data every day. As this may not be an update you'd want to be running every day, the daily updates might be "more bug than feature" for many researchers.

Now, two lines of code suffice to set up a connection to a DuckDB database pointing to the `comp.g_secd` Parquet file on my hard drive.

```
db <- dbConnect(duckdb::duckdb())

g_secd <- load_parquet(db, table = "g_secd", schema = "comp")
```

Now, we can run exactly the same query as we did for the WRDS PostgreSQL server.

```
g_secd |>
  count(curcdd) |>
  arrange(desc(n)) |>
  collect() |>
  system_time()
```

```
   user  system elapsed
  1.431   0.112   0.179

# A tibble: 115 x 2
   curcdd        n
   <chr>     <dbl>
 1 EUR    48037544
 2 JPY    31823971
 3 CNY    25635258
 4 INR    20653714
 5 GBP    20619033
 6 KRW    15641761
 7 HKD    14697439
 8 AUD    13800095
 9 TWD    11646012
10 THB     8980840
# i 105 more rows
```

Wow! So *once we have the data in Parquet form*, the query is blazingly fast. Something taking over 6 minutes on the SAS "supercomputers" takes about 0.2 seconds on my consumer-grade Mac mini.

Note that the results match the output from the PostgreSQL server exactly, because we updated the data before running the query.

I now make a little Python function to get information about the size of the Parquet file that has been created.

```
from pathlib import Path
import os

def file_size(path) -> str:
    n_bytes = path.stat().st_size
    for unit in ["B", "KB", "MB", "GB", "TB", "PB"]:
        if n_bytes < 1024:
            return f"{n_bytes:.1f} {unit}"
        n_bytes /= 1024
    return f"{n_bytes:.1f} EB"
```

Now, I run this function on the Parquet file created by `wrds_update_pq()`.

```
data_dir = Path(os.environ["DATA_DIR"]).expanduser()
path = data_dir / "comp" / "g_secd.parquet"
file_size(path)
```

```
'3.8 GB'
```

So the 57.71 GB PostgreSQL table is reduced to under 4 GB as a Parquet file.[13] This is much more manageable!

Returning to the mismatch between the output from the SAS query and the PostgreSQL query, we can inspect the "last modified" data for the Parquet file, which are taken from the table comments on the PostgreSQL server.

```
from pathlib import Path
from db2pq import get_modified_pq
import os

def get_modified(table, schema, *,
                 data_dir=None, archive=False):
  if data_dir is None:
      data_dir = Path(os.environ["DATA_DIR"]).expanduser()

  if archive:
      files = list((data_dir / schema / "archive")
                   .glob(f"{table}_*.parquet"))
      return [get_modified_pq(file) for file in files]
  else:
      path = data_dir / schema / f"{table}.parquet"
      return get_modified_pq(path)
```

Running this function on the main file tells us that the PostgreSQL table was updated yesterday, explaining why we see the difference from the WRDS SAS output above:

```
get_modified("g_secd", "comp")
```

```
'Merged Global Security Daily File (Updated 2026-02-11)'
```

We can run the function with `archive=True` to see what we have in the archive:

```
get_modified("g_secd", "comp", archive=True)
```

```
['Last modified: 09/07/2025 12:14:53', 'Merged Global Security Daily File (Updated 2026-01-20)']
```

13. Note that the PostgreSQL server, like SAS, also has one or more indexes, which occupy an additional 15.75 GB.

The one archived file I have has an update string taken from the SAS file existing on the WRDS server when that update occurred.[14]

References

Gow, I.D., Ding, T., 2024. Empirical research in accounting: Tools and methods. Chapman & Hall/CRC, London, UK. https://doi.org/10.1201/9781003456230

14. A recent update to `db2pq` means that the "last updated" string comes from the comments appended to the PostgreSQL table.

Listing 1 Output from `PROC CONTENTS` with `comp.g_secd` (Part 1)

```
The SAS System

The CONTENTS Procedure

Data Set Name        COMP.G_SECD                Observations          301930495
Member Type          DATA                       Variables             54
Engine               V9                         Indexes               5
Created              02/11/2026 11:02:53        Observation Length    480
Last Modified        02/11/2026 11:14:23        Deleted Observations  0
Protection                                      Compressed            NO
Data Set Type                                   Sorted                YES
Label                Merged Global Security
                     Daily File
Data Representation  SOLARIS_X86_64,
                     LINUX_X86_64, ALPHA_TRU64,
                     LINUX_IA64
Encoding             utf-8  Unicode (UTF-8)

                  Engine/Host Dependent Information

Data Set Page Size          65536
Number of Data Set Pages    2220079
First Data Page             1
Max Obs per Page            136
Obs in First Data Page      115
Index File Page Size        8192
Number of Index File Pages  2827276
Number of Data Set Repairs  0
Filename                    /wrds/comp/sasdata/d_global/g_secd.sas7bdat
Release Created             9.0401M7
Host Created                Linux
Inode Number                2465334825
Access Permission           rw-r-----
Owner Name                  wrdsadmn
File Size                   136GB
File Size (bytes)           145495162880
```

Listing 2 Output from `PROC CONTENTS` with `comp.g_secd` (Part 2)

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
46	ISIN	Char	20	$12.		International Security Identification Number
48	SEDOL	Char	20	$7.		SEDOL
6	ajexdi	Num	8	F19.8	20.8	Adjustment Factor (Issue)-Cumulative by Ex-Date
34	anncdate	Num	8	YYMMDDN8.		Dividend Declaration Date
16	cheqv	Num	8	F17.4	18.4	Cash Equivalent Distributions
17	cheqvgross	Num	8	F17.4	18.4	Cash Equivalent Distributions, Gross
18	cheqvnet	Num	8	F17.4	18.4	Cash Equivalent Distributions, Net of Tax
35	cheqvpaydate	Num	8	YYMMDDN8.		Cash Equivalent Distributions per Share Payment Date
19	cheqvtm	Char	2	$2.	$2.	Cash Equivalent Distributions Tax Code
4	conm	Char	87	$87.	$87.	Company Name
7	cshoc	Num	8	F19.	19.	Shares Outstanding
8	cshtrd	Num	8	F29.8	30.8	Trading Volume - Daily
5	curcdd	Char	3	$3.	$3.	ISO Currency Code - Daily
15	curcddv	Char	3	$3.	$3.	ISO Currency Code - Dividend
3	datadate	Num	8	YYMMDDN8.		Data Date - Daily Prices
20	div	Num	8	F17.4	18.4	Dividends per Share - Ex Date - Daily (Issue)
21	divd	Num	8	F17.4	18.4	Cash Dividends - Daily
22	divdgross	Num	8	F17.4	18.4	Cash Dividends, Gross
23	divdnet	Num	8	F17.4	18.4	Cash Dividends, Net of Tax
36	divdpaydate	Num	8	YYMMDDN8.		Cash Dividends - Daily Payment Date
24	divdtm	Char	2	$2.	$2.	Cash Dividend Tax Code
25	divgross	Num	8	F17.4	18.4	Cash Dividend including Special, Gross
26	divnet	Num	8	F17.4	18.4	Cash Dividend including Special, Net of Tax
27	divrc	Num	8	F17.4	18.4	Return of Capital - Daily
28	divrcgross	Num	8	F17.4	18.4	Return of Capital, Gross
29	divrcnet	Num	8	F17.4	18.4	Return of Capital, Net of Tax
37	divrcpaydate	Num	8	YYMMDDN8.		Return of Capital - Daily Payment Date

Listing 3 Output from `PROC CONTENTS` with `comp.g_secd` (Part 3)

```
30 divsp       Num    8 F17.4     18.4     Special Cash Dividends - Daily
31 divspgross  Num    8 F17.4     18.4     Special Cash Dividends, Gross
32 divspnet    Num    8 F17.4     18.4     Special Cash Dividends, Net of Tax
38 divsppayda  Num    8 YYMMDDN8.          Special Cash Dividends
   te                                      - Daily Payment Date
33 divsptm     Char   2 $2.       $2.      Special Cash Dividend Tax Code
44 epf         Char   1 $1.       $1.      Earnings Participation Flag
45 exchg       Num    8 F6.       6.       Stock Exchange Code
50 fic         Char   3 $3.       $3.      Current ISO Country
                                           Code - Incorporation
51 gind        Char   6 $6.       $6.      GIC Industries
52 gsubind     Char   8 $8.       $8.      GIC Sub-Industries
 1 gvkey       Char   6 $6.       $6.      Global Company Key - Daily Prices
 2 iid         Char   3 $3.       $3.      Issue ID - Daily Price
53 loc         Char   3 $3.       $3.      Current ISO Country
                                           Code - Headquarters
54 monthend    Num    8                    End of Month Indicator
39 paydate     Num    8 YYMMDDN8.          Dividend Payment Date
 9 prccd       Num    8 F29.8     30.8     Price - Close - Daily
10 prchd       Num    8 F29.8     30.8     Price - High - Daily
11 prcld       Num    8 F29.8     30.8     Price - Low - Daily
12 prcod       Num    8 F29.8     30.8     Price - Open - Daily
13 prcstd      Num    8 F6.       6.       Price Status Code - Daily
14 qunit       Num    8 F17.4     18.4     Price Quotation Unit
40 recorddate  Num    8 YYMMDDN8.          Dividend Record Date
47 secstat     Char   1 $1.       $1.      Security Status Marker
41 split       Num    8 F19.8     20.8     Stock Split Rate
42 splitf      Char   8 $8.       $8.      Stock Split Footnote
49 tpci        Char   8 $8.       $8.      Issue Type Code
43 trfd        Num    8                    Daily Total Return Factor

     Alphabetic List of Indexes and Attributes

                              # of
                            Unique
#    Index                  Values    Variables

1    ISIN                    92319
2    SEDOL                  121897
3    exchg                     156
4    fic                       131
5    gvkey_datadate    245169413    gvkey datadate

        Sort Information

Sortedby       gvkey iid datadate
Validated      YES
Character Set  ASCII
```