WRDS to Parquet

Ian D. Gow

This is place to be when your source data sets live in the WRDS PostgreSQL database and you want to turn them in Parquet files in your local data repository.

Main helpers

The main helper function on this path is `wrds_update_pq()`, which can be used to download a WRDS table as a Parquet file. This function only rewrites a Parquet file when the WRDS source is newer, so it can reduce the volume of unncessary downloads.

Two additional functions are:

- `wrds_sql_to_pq(sql, table_name, schema, ...)` exports a custom WRDS SQL query into the standard Parquet layout.
- `wrds_update_schema(schema, ...)` updates a full WRDS schema into Parquet.

A final key function is `wrds_pg_to_pq()`, which exports one WRDS table to Parquet, but without the update logic of `wrds_update_pq()`. For most purposes, you should use `wrds_update_pq()` and not the lower-level `wrds_pg_to_pq()` function.

Using `wrds_update_pq()`

An illustrative first run

A good place to start with `wrds_update_pq()` is with a smaller table, such as `comp.company`. Two lines of Python code are enough to get you started.

```
from db2pq import wrds_update_pq

wrds_update_pq("company", "comp")
```

```
comp.company already up to date.
```

Note that there are two key **environment variables** that are used by `wrds_update_pq()`: `DATA_DIR` and `WRDS_ID`.

The `DATA_DIR` setting is used to determine where the Parquet file that is created will be stored. Parquet files are written as:

```
<DATA_DIR>/<schema>/<table>.parquet
```

If `DATA_DIR` is unset and you do not pass `data_dir=`, `db2pq` falls back to the current working directory.

As you might expect `WRDS_ID` should contain your WRDS ID.[1] If you have not configured the `WRDS_ID` environment variable, you will be prompted by `db2pq` to enter your WRDS ID (`db2pq` will also suggest

[1] For compatibility with Tidy Finance, `db2pq` also accepts values in `WRDS_USER`.

adding this to and `.env` file in the directory you are calling Python from). If the WRDS PostgreSQL password is not yet in `.pgpass`, `db2pq` will prompt you for it and save it for later use.

The best approach is to save these variables in a file named `.env` either in the calling directory or in your home directory. For example:

```
WRDS_ID=your_wrds_id
DATA_DIR=~/Dropbox/pq_data
```

One approach would be to have a single `DATA_DIR` set on your computer and use a single data repository for WRDS data for all projects. But, because you can have different `DATA_DIR` values in `.env` files in different folders, you can easily maintain project-specific versions of WRDS data if that is how you prefer to manage data.

Identifying table names and schemas

The key requirement for using `wrds_update_pq()` is that you know the table name and schema that you are looking for.

This information can come from multiple sources. One source would be existing code. For example, if you see SAS code that refers to `comp.funda`, you can get the data from WRDS by running `wrds_update_pq(table_name="funda", schema="comp")`. Multiple examples of code using different table names across a number of schema are provided in *Empirical Research in Accounting: Tools and Methods* starting with Chapter 6.

Another source for this information is the WRDS web query interface, which is discussed here. Using the WRDS web query can help to identify the exact table behind a dataset or to draft a starting SQL query for you to edit. In this way, the web query can offer a practical way to move from the WRDS website to a more reproducible workflow.

More features of `wrds_update_pq()`

The `wrds_update_pq()` function accepts additional optional arguments that allow more refined handling of data.

Cleaning the data

Some times WRDS data aren't in exactly the right form. The `wrds_update_pq()` allows the user to address issues that exist in some tables.

Correcting data types

For example, the columns `lpermno` and `lpermco` on `crsp.ccmxpf_lnkhist` are stored as floating-point numbers in WRDS's PostgreSQL database, but `permno` and `permco` (what these variables relate to) are elsewhere stored as integers and `wrds_update_pq()` allows a user to "correct" data types. (For more on `crsp.ccmxpf_lnkhist`, see Chapter 7 of *Empirical Research in Accounting: Tools and Methods*.) The following code illustrates this feature:

```
from db2pq import wrds_update_pq

wrds_update_pq(
    "ccmxpf_lnkhist",
    "crsp",
    col_types={"lpermno": "int32", "lpermco": "int32"},
)
```

Setting time zones

Another issue is the absence of time zone information. WRDS PostgreSQL timestamps are generally stored as `TIMESTAMP WITH TIME ZONE` type, which is problematic. For example, the timestamps in `ravenpack_dj.rpa_djpr_equities_2024` should be encoded as UTC. This can be achieved using the followin.

```
wrds_update_pq(
    "rpa_djpr_equities_2024",
    "ravenpack_dj",
    use_sas=True,
    sas_schema="rpa",
    col_types={
        "timestamp_utc": "timestamp",
        "rpa_time_utc": "string",
        "event_start_date_utc": "timestamp",
        "event_end_date_utc": "timestamp",
    },
)
```

WRDS provides Call Report from the FFIEC as part of its `bank` schema. We know that the timestamps provided by the FFIEC are expressed in US Eastern time (see my note on Call Reports). Fortunately, we can tell `wrds_update_pq()` this using the `tz` argument.

```
wrds_update_pq("wrds_call_rcfa_1", "bank",
               tz="America/New_York")
```

Working with large tables

In principle, `wrds_update_pq()` can be used to convert any WRDS table to a local Parquet file. However, doing so for very large files will take longer and use more of your local storage space.

Dropping unwanted variables

The WRDS tables for Audit Analytics data often include variables from the "company financial block", which are financial statement variables such as net income or total assets for the "`closest`", "`match`", "`hiwater`", or "`prior`" periods (either quarters [`qu`] or years [`yr`]). These variables expand the size of the tables dramatically, and their provenance is unclear (likely scraped from SEC filings, which are the primary source for Audit Analytics data). Also, the meaning of the terms "`closest`", "`match`", "`hiwater`", and "`prior`" is unclear, the data have historically been very poorly documented on WRDS, and few researchers appear to use these columns.

The `wrds_update_pq()` function allows us to simply omit these variables when exporting the data using the `drop=` argument. This argument allows us to specify either a **list** of unwanted variables or a **regular expression** that matches variables to be excluded. (For more on regular expressions, see Chapter 9 of *Empirical Research in Accounting: Tools and Methods*.)

```
wrds_update_pq("feed02_auditor_changes", "audit",
               drop="^(match|prior|closest)")
```

Selecting only certain variables

In some situations, we may only want a small subset of the columns available on WRDS. For example, the Tidy Finance book only uses three columns from `crsp.dsf_v2`. We can handle this with the `keep=`

argument, which allows us to specify either a list of the variables we want or a regular expression that matches variables to be included.

```
wrds_update_pq("dsf_v2", "crsp",
               keep=["permno", "dlycaldt", "dlyret"])
```

Renaming variables

Sometimes it is helpful to rename variables as part of the export so the local Parquet file already matches your preferred naming scheme. The `rename=` argument uses a dictionary mapping source names to output names.

```
wrds_update_pq(
    "company",
    "comp",
    keep=["gvkey", "conm", "sic"],
    rename={"conm": "company_name"},
)
```

If you combine `rename=` with `col_types=`, the keys in `col_types=` should refer to the output names after renaming.

```
wrds_update_pq(
    "ccmxpf_lnkhist",
    "crsp",
    rename={"lpermno": "permno_link"},
    col_types={"permno_link": "int32"},
)
```

Selecting only certain rows

Another way to reduce the size of downloaded tables is to filter out unneeded rows using `where=`. The `where` argument should be written as an SQL `WHERE` condition. For example, it is common for researchers to focus on the so-called "standard" secondary key set when working with `comp.funda` (see Chapter 6 of *Empirical Research in Accounting: Tools and Methods* for more detailed discussion). This can be achieved using code like this:[2]

```
wrds_update_pq("funda", "comp",
               where="indfmt = 'INDL' " +
                     "AND datafmt = 'STD' " +
                     "AND consol = 'C' " +
                     "AND popsrc = 'D'")
```

Limiting the number of rows

Some times it is helpful to do a test download of a smaller table to make sure that things look right before doing a final download. This can be achieved with the `obs=` argument. The following code limits the download to 1000 rows.

```
wrds_update_pq("funda", "comp", obs=1000)
```

[2]Here the `+` glues the four pieces into a single string.

Forcing an update

Sometimes you may want to trigger a download even when the data on WRDS are no newer than what you have locally. For example, you may have used `obs=1000` or you may want download with corrected data types using `col_types`. In such cases, you can use `force=True`.

```
wrds_update_pq("funda", "comp", force=True)
```

Using SAS metadata for update logic

The update logic of `wrds_update_pq()` works by comparing information stored by WRDS in the PostgreSQL table comment with that retrieved and stored from previous runs of `wrds_update_pq()`. However, in some cases the PostgreSQL table comments are missing or may not provide the signal of "freshness" that you want to use.

WRDS data tables generally come in two forms: PostgreSQL tables and SAS files. The `wrds_update_pq()` function allows a user to specify `use_sas=True`, which will cause `wrds_update_pq()` to check the "last updated" metadata from the corresponding WRDS SAS file instead.

In some cases, the schema used for SAS differs from that used for PostgreSQL and `wrds_update_pq()` can handle such cases using the `sas_schema` argument. For example, the WRDS PostgreSQL schema and the SAS schema do not line up for RavenPack and the following code handles this.

```
wrds_update_pq(
    "rpa_entity_mappings",
    "ravenpack_common",
    use_sas=True,
    sas_schema="rpa",
)
```

Note that you will need to have installed the `[sas]` optional features to use `use_sas=True`:

```
pip install --upgrade "db2pq[sas]"
```

Additionally, you will need to set up your connection to the WRDS cloud server, which is separate from the WRDS PostgreSQL setup. More information on this step is provided on the `db2pq` package `README` page.

Archiving data when updating

The argument `archive=True` moves replaced Parquet files into an archive directory.

```
wrds_update_pq("funda", "comp", archive=True)
```

Related reference pages

- wrds_pg_to_pq
- wrds_sql_to_pq
- wrds_update_pq
- wrds_update_schema
- pq_last_modified

Related guides

- Using the WRDS web query