BCWAT Data Model & ETL Pipeline Refactor

This document aims to document the current status and proposed changes as part of the Water tool and Water Portal consolidation of three major facets of the applications being consolidated:

1. ETL Pipeline Orchestration
2. ETL Pipeline Structure
3. Data Model

The applications considered include: Northern Water Portal (NWP), Southern Water Portal (SWP), Kootenay-Boundry Water Tool (KBWT), Cariboo Water Tool (CWT), Northwest Water Tool (NWWT), and Omineca Water Tool (OWT).

Current Implementation

Orchestration

The current ETL Pipelines are orchestrated via 6 cronjobs, 5 of which are active on AquaDB3 (BCER's server), and the last is ran on Kubernetes. Most of the cronjobs import multiple of the ~27 datasources simultaneously.

Another task that has to be completed is the quarterly task which needs to be ran manually and has no orchestration whatsoever.

Current issues

1. No orchestration whatsoever for the quarterly task is something devs must remember to do and balance with other workloads.
2. When one data source has issues at a certain time of day all data sources within the same job must move aswell potentially causing issues for them.
3. Running cronjobs directly on AquaDB3 can be annoying (GlobalProtect + needing to run the code on the server itself leads to some rare un-reproduceable bugs)
4. Poor visibility of when the scrapers are running/logs per run etc

Data Model

The current data model is split in to two databases: `bcwt-staging` and `bcwt-dev` (or `ogc` for production). The data gets initially scraped in to `bcwt-staging` as raw data. The only changes that are made is that the unneeded attributes are dropped from the data.

One of the cronjobs that is ran runs the `convert_hourly_to_daily` workflow, which converts the hourly data that has been scraped, in to daily data. Following that, another cronjob executes, which pushes the data to the dev and prod database.

This staging and dev/prod database separation was initially made so that no data was directly piped in to the production environment (as far as I understand)

The dev/prod database have a schema for each of the projects, which makes the ERD diagram (generated by PgAdmin4) look like this:

Actually Illegible

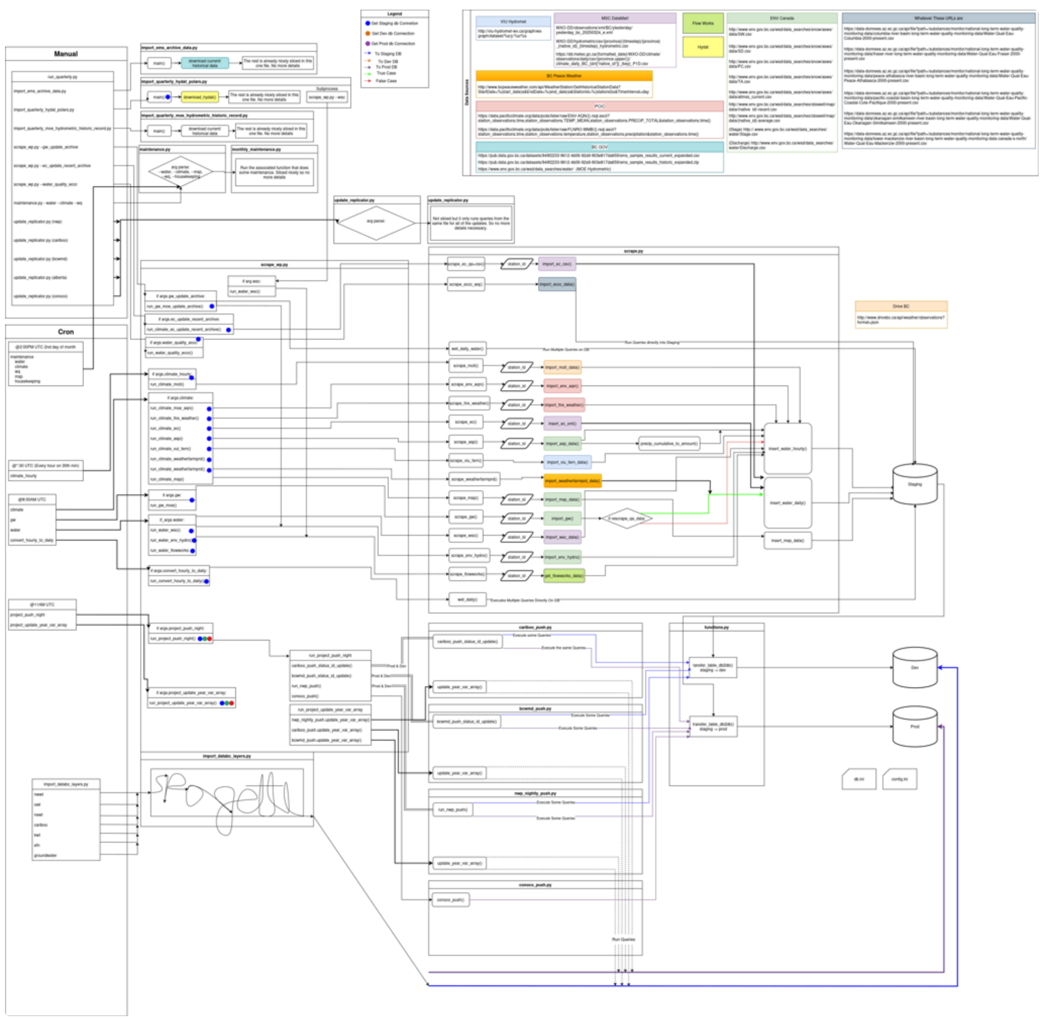
At least 50% of the tables are redundant, and can be merged into a handful of tables with a common structure, or dropped after making some changes to the data model. Additionally there are many tables within the existing data model that are missing keys or have composite types (arrays, json, etc) within columns which violate database normality. Note the above erd is only for 1 of the 2 DBs per deployment

Current Issues

1. Hard to track down issues that happen in the database and whether it has migrated to prod or dev already
2. The database is not normalized (harder to extend and confidently make changes in)
3. The large amount of schemas makes it extremely hard to work with
4. It is complicated enough that we could not even export the ERD out of PGAdmin without being told the image is too large
5. 3 databases per deployment is very confusing especially the relationship between `ogc` and `bcwt-dev`

ETL Pipeline

A simplified flow chart of the current ETL Pipeline has been attached below:



This diagram shows that there are groups of scrapers jobs that are independent of each other, mainly the `import_databc_layers.py` and `scrape_wp.py`.

As you can see in the diagram, most of the scrapers insert into the staging database using the same function: `insert_water_hourly()` and `insert_water_daily()`.

Current Issues

1. Although not directly shown by the diagram here there is way too many lines of code for how similar a lot of the code is there is 43502 total lines of code! Alot more code can be reused then what currently is. Additionally there is a ton of boilerplate code that can easily be extracted into fucntions
2. 10 + datasources per job can make it hard to debug and makes logs cluttered
3. One section was so jumbled in arrows we could not map it out and chose to instead exclude it.

Suggested Changes

Orchestration

The plan for improving orchestrating the scrapers is to use Apache Airflow instead of cronjobs. Apache Airflow is definitely more than we need for this project, but the pros out weigh the cons significantly.

Apache Airflow will bring excellent orchestration management, with a UI for identifying issues quickly. It comes with a built in logging and programmable retry feature per DAG and Task, allowing us to investigate thoroughly and systematically retry.

Providers such as PostgreSQL, and SendGrid ensures that we have the tools necessary to run the scrapers as we want it with built in support from Airflow!

How does this address our current issues?

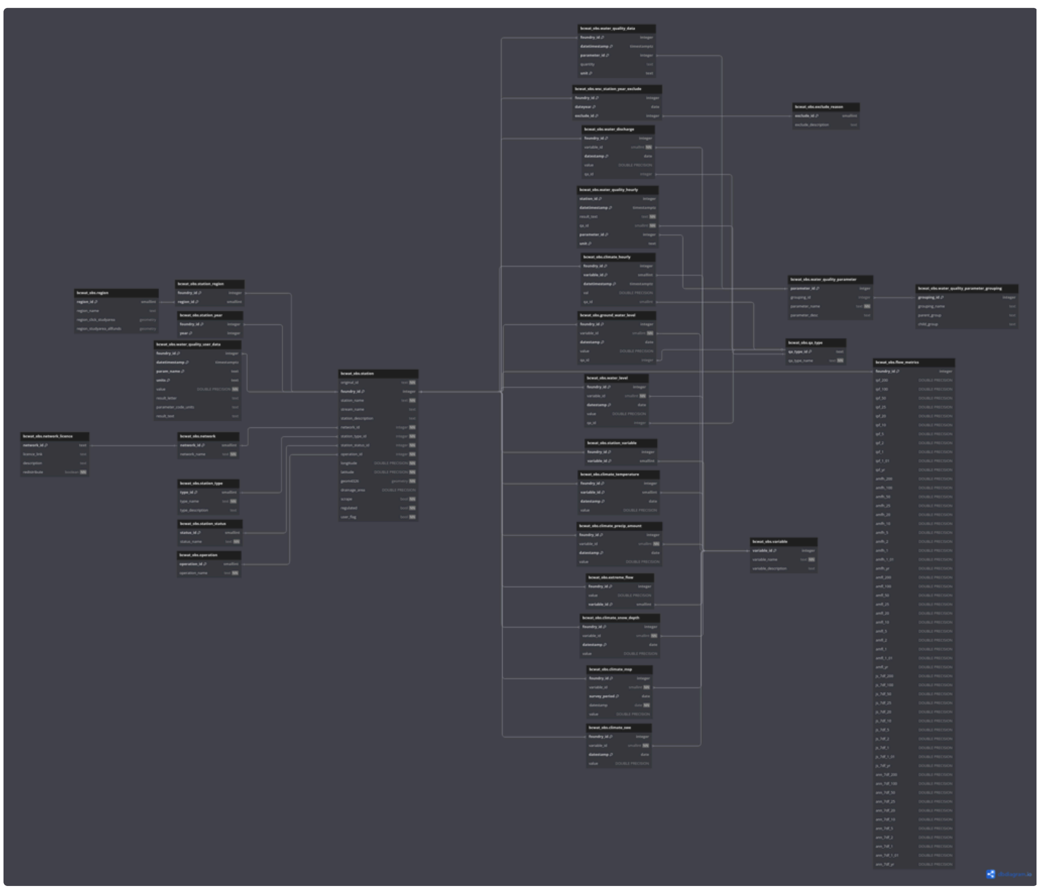
1. No orchestration whatsoever for the quarterly task is something devs must remember to do and balance with other workloads.
    a. This addresses this issue by proposing to use Airflow's sensor functionality for all 3 quarterly scripts and using ExternalTask Sensors to kick off the required normal pipelines they have to run after they are all done.
2. When one data source has issues at a certain time of day all data sources within the same job must move aswell potentially causing issues for them.
    a. By moving to Apache Airflow we can put each of the ~27 datasources into its own DAG and allow Airflow to deal with the scheduling of this many nightly tasks (this was likely not done yet due to the headache of schedulign this many tasks and not wanting to run them concurrently on AquaDB3)

3. Running cronjobs directly on AquaDB3 can be annoying (GlobalProtect + needing to run the code on the server itself leads to some rare un-reproduceable bugs)
    a. Airflow can be and is recommended to be completely dockerized leading to consistent environments for both devs and the running code in production.
4. Poor visibility of when the scrapers are running/logs per run etc
    a. Airflow's Web UI offers great visibility and will allow people unfamiliar with the pipelines to easily retry individual pipelines or temporarily disable problematic ones! Additionally it stores logs per run in a central location. (more info in lightning talk slides)

Data Model

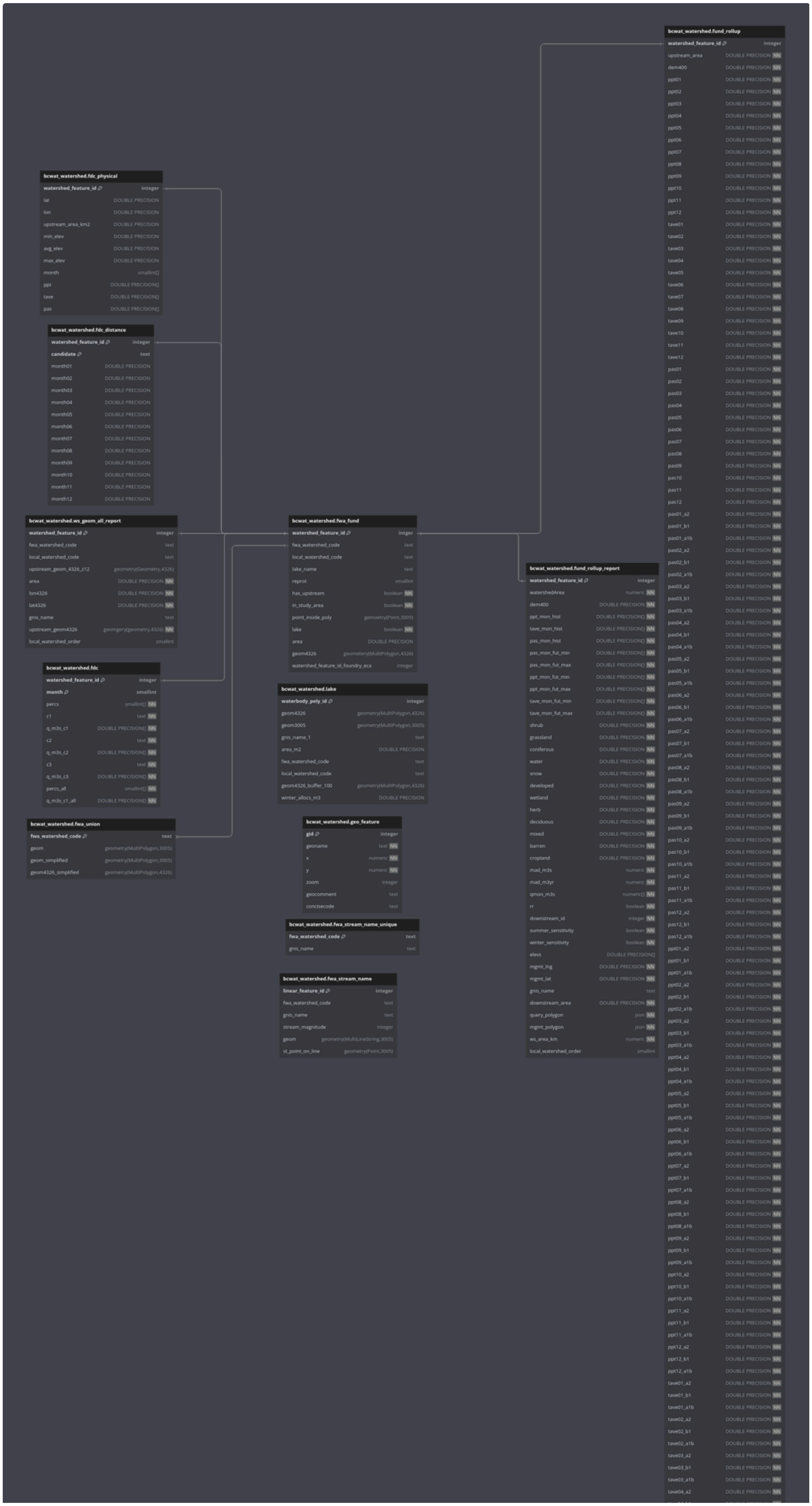
Since the current DB model will have a lot of redundant tables, the data model has been reduced to three schemas:

`bcwat_observation`:

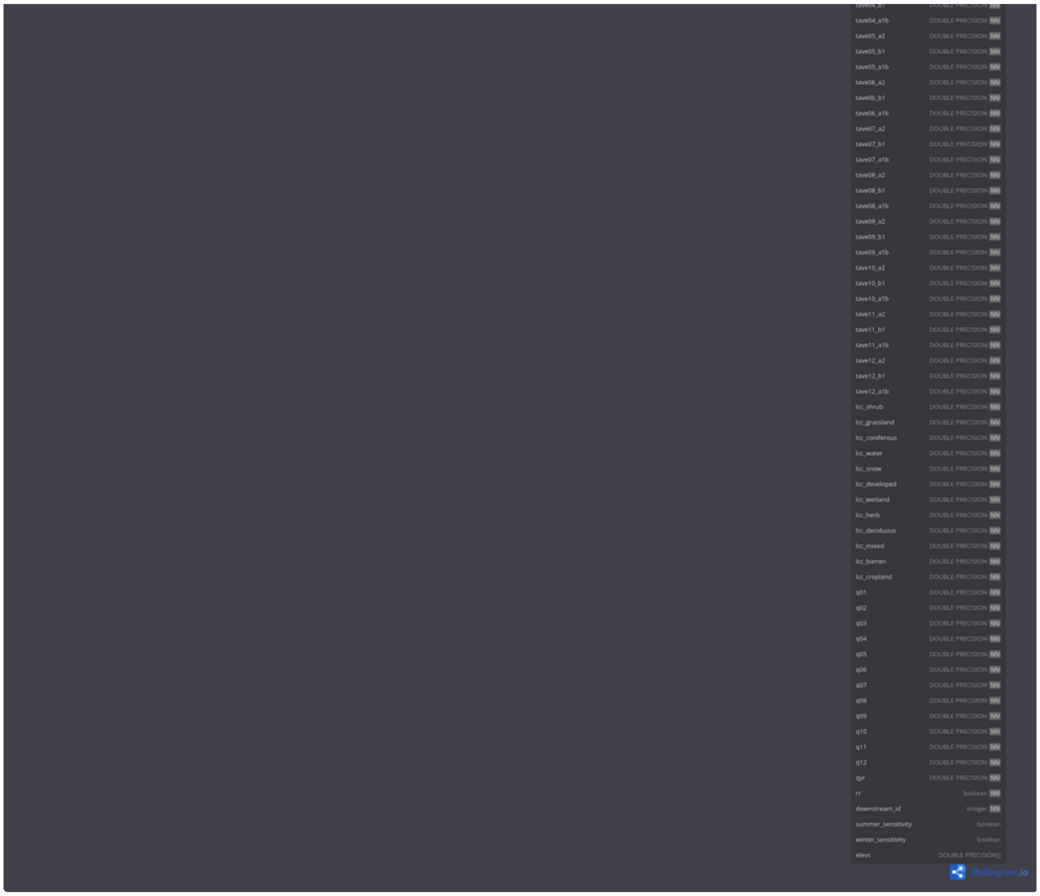


`bcwat_licence`:

`bcwat_watershed`



bcwat_watershed.fund_rollup
bcwat_watershed.fdc_physical
bcwat_watershed.fdc_distance
bcwat_watershed.ws_geom_all_report
bcwat_watershed.fwa_fund
bcwat_watershed.fund_rollup_report
bcwat_watershed.fdc
bcwat_watershed.lake
bcwat_watershed.fwa_union
bcwat_watershed.geo_feature
bcwat_watershed.fwa_stream_name_unique
bcwat_watershed.fwa_stream_name



The data model is an adapted form of the original by doing the following:

Removing extra databases

All of the above schemas will be within a single database with no need for multiple databases to 'airlock' data at different stages

Removing redundant tables/schemas

This is possible because the existing tools already share a lot of the functionality. Thankfully, the tables that consist of the same data were named the exact same in each schema, which will allow for easy aggregation. This will reduce the number of tables required by around 50%, which makes maintaining, and understanding the database easier. Additionally there were many views or tables that were straight up not being used as far as we can tell.

Removing Arrays and Json objects where possible

In the old data model, some columns were stored as arrays, which makes querying for specific data more difficult than it needs to be. This is solved by expanding the column in to a new table, making the each entry in the array occupy a row. That being said, there are still some tables that have arrays. However the regularly used tables have been made more friendly to work with.

Sorting the tables into schemas

During the investigation of the data model, it became evident that there was a divide between the tables that the Water Portals, and the Water Tools used. Furthermore, the scrapers scrape data from the DataBC CLI, which largely collects water licensing data. This lead to the conclusion that instead of putting all the tables in one schema as the current tools do, splitting it in to smaller schemas would be better for maintainability and digestibility for developers.

Adding Foreign Keys and Constraints

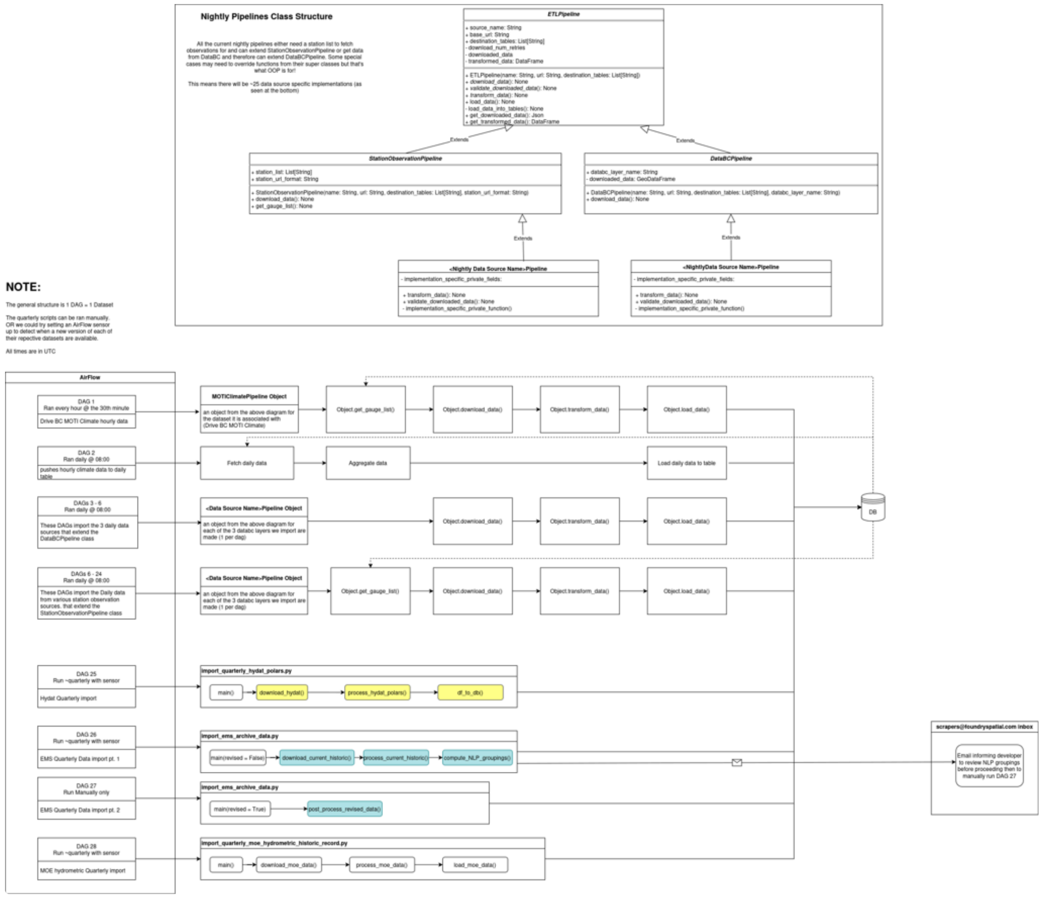
Added Foreign keys everywhere where it was possible. If a table didn't have a primary key, a surrogate key has been added to make sure that we get as close to possible as 3NF.

How does this address our current issues?

1. Hard to track down issues that happen in the database and whether it has migrated to prod or dev already
    a. By eliminating the excess databases this is no longer a concern as the data will not be in a delayed rollout to the different environments. Similarly easy access to logs will make it more clear what of the environments (dev, test, prod) have defects as they appear.
2. The database is not normalized (harder to extend and confidently make changes in).
    a. Although not fully in 3rd normal form. This proposed data structure is significantly closer and only uses non-normal practices where it benefits the developer.
3. The large amount of schemas makes it extremely hard to work with
    a. Reducing the schemas to just 3 all of which serve a unique purpose helps address this issue. This also helps remove a majority of the duplicated data across the 3 databases and multiple schemas within `bcwt-dev` and `ogc`
4. It is complicated enough that we could not even export the ERD out of PGAdmin without being told the image is too large
    a. The ERD's made above are very legible, at-least when compared to the original. Ultimately this is still a complicated data-driven project with 25+ sources. The ERD still reflects these complications with many tables, however the hope is it does so in a more manageable way.
5. 3 databases per deployment is very confusing especially the relationship between `ogc` and `bcwt-dev`
    a. Removing the databases! The relationship between `ogc` and `bcwt-dev` may go down as one of mans greatest unsolved mysteries.

ETL Pipelines

Below is the flow of the new ETL pipelines that we will be using:



The pipeline will be refactored in to an object oriented architecture. This redesign will make the scrapers more modular, which will ensure that the code base is less bloated. It additionally has the benefit of being more linear as shown by the above diagram. Less criss crossing of different pipelines. This modular approach additionally plays nicely into Airflows strengths making it clear what piece of work should be put into a DAG. 1 DAG = 1 Object

How does this address our current issues?

1. Although not directly shown by the diagram here there is way too many lines of code for how similar a lot of the code is there is 43502 total lines of code! Alot more code can be reused then what currently is. Additionally there is a ton of boilerplate code that can easily be extracted into functions
   a. By using an object oriented structure all of the shared code can be put into private inner functions within the super-classes reducing the great amount of redundant code. Additionally using the Airflow Taskflow API will reduce the amount of boilerplate code needed

2. 10+ data-sources per job can make it hard to debug and makes logs cluttered
    a. reducing to 1 DAG per data-source will make logs more digestible and easy to figure out what exactly went wrong
3. One section was so jumbled in arrows we could not map it out and chose to instead exclude it.
    a. This section has been turned into a simple straight line (DAGs 3- 6). In-fact all the lines have been made into straight linear flows in contrast with the original diagram that had many backtracking steps or other conditional branches, this leads to easier to follow code for developers!