

BCWAT Data Model & ETL Pipeline Refactor

Current Implementation

Orchestration

Current issues

Data Model

Current Issues

ETL Pipeline

Current Issues

Suggested Changes

Orchestration

How does this address our current issues?

Data Model

Removing extra databases

Removing redundant tables/schemas

Removing Arrays and Json objects where possible

Sorting the tables into schemas

Adding Foreign Keys and Constraints

How does this address our current issues?

ETL Pipelines

How does this address our current issues?

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-Boundary 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

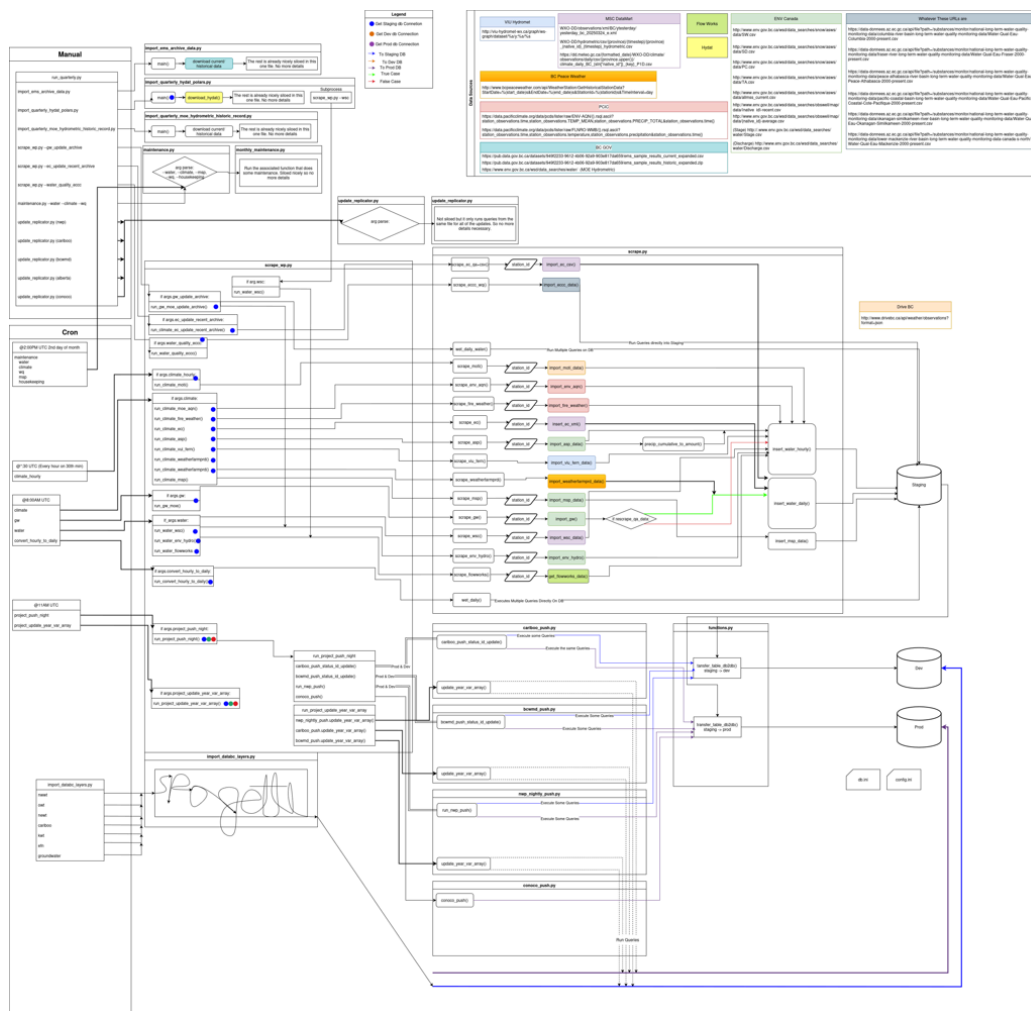
At least 50% of the tables are redundant, and can be merged into a handful of tables with a common structure, or dropped after making some changes to the data model. Additionally there are many tables within the existing data model that are missing keys or have composite types (arrays, json, etc) within 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_dataabc_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! A lot more code can be reused than what currently is. Additionally there is a ton of boilerplate code that can easily be extracted into functions
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 outweigh 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?

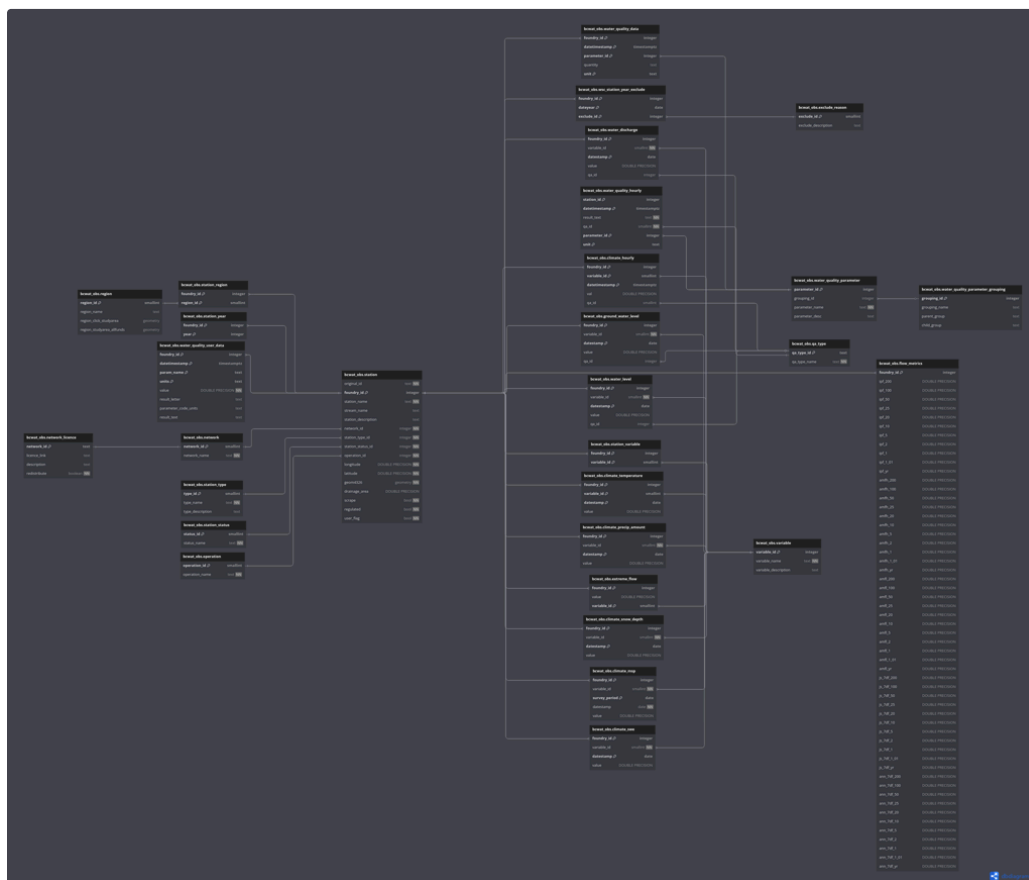
1. No orchestration whatsoever for the quarterly task is something devs must remember to do and balance with other workloads.
 - a. This addresses this issue by proposing to use Airflow's sensor functionality for all 3 quarterly scripts and using ExternalTask Sensors to kick off the required normal pipelines 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 as well 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 scheduling 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-reproducible 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_licence.lake_licence	
waterbody_poly_id ϕ	integer
lake_name	text
fs_id ϕ	text
licence_stream_name	text


bcwat_licence.elevation_bookend	
elevation_flat	DOUBLE PRECISION[]
elevation_steep	DOUBLE PRECISION[]

bcwat_licence.bc_wls_water_approvals	
fs_id ϕ	text
wsd_region	text
approval_type	text
approval_file_number	text
source	text
works_description	text
quantity	DOUBLE PRECISION
quantity_units	text
qty_diversion_max_rate	DOUBLE PRECISION
qty_units_diversion_max_rate	text
water_district	text
precinct	text
approval_status	text
application_date	date
fcbc_acceptance_date	date
approval_issuance_date	date
approval_start_date	date
approval_expiry_date	date
geom	geometry(Point,4326) 
proponent	text
qty_display	text
podno	text

bcwat_licence.hypsometric_elevation_rollup	
watershed_feature_id ϕ	integer
elevs	DOUBLE PRECISION[]

bcwat_licence.bc_wls_wrl_wra	
fs_id ϕ	text
licence_no	varchar(16) 
tpod_tag	varchar(10) 
purpose	text 
pd_no	varchar(15) 
qty_original	DOUBLE PRECISION
qty_flag	varchar(1)
qty_units	varchar(25)
licensee	varchar 
lic_status_date	date
priority_date	date
expiry_date	date
longitude	DOUBLE PRECISION
latitude	DOUBLE PRECISION
stream_name	varchar
quantity_day_m3	DOUBLE PRECISION
quantity_sec_m3	DOUBLE PRECISION
quantity_ann_m3	DOUBLE PRECISION
lic_status	text
rediversion_flag	varchar(1)
flag_desc	varchar(100)
file_no	varchar(10)
water_allocation_type	varchar(2) 
geom	geometry(Point,4326)
water_source_type_desc	text
hydraulic_connectivity	varchar(215)
well_tag_number	DOUBLE PRECISION
related_licences	text[]
industry_activity	text 
purpose_groups	text 
is_consumptive	boolean 
ann_adjust	DOUBLE PRECISION
documentation	json
qty_display	text
puc_groupings_storage	text
date_updated	timestamptz

bcwat_licence.bc_data_import_date	
dataset ϕ	text
import_date	date
description	text

bcwat_licence.licence_ogc_short_term_approval	
fs_id ϕ	text
pod_number	text
short_term_water_use_num	text
water_source_type	text
water_source_type_desc	text
water_source_name	text
purpose	text
purpose_desc	text
approved_volume_per_day	integer
approved_total_volume	integer
approved_start_date	date
approved_end_date	date
status	text
application_determination_num	text
activity_approval_date	date
activity_cancel_date	date
legacy_ogc_file_number	text
proponent	text 
authority_type	text
land_type	text
data_source	text
geom	geometry(Point,4326)
latitude	DOUBLE PRECISION
longitude	DOUBLE PRECISION
is_consumptive	boolean
qty_display	text

bcwat_licence.wls_water_approvals	
fs_id	text
objctid ϕ	integer
water_approval_id	integer
wsd_region	charvar(20)
approval_type	charvar(25)
approval_file_number	charvar(15)
fcbc_tracking_number	integer
source	charvar(255)
works_description	charvar(255)
quantity	charvar(255)
water_district	charvar(30)
precinct	charvar(40)
latitude	DOUBLE PRECISION
longitude	DOUBLE PRECISION
utm_zone	smallest
utm_easting	integer
utm_northing	integer
map_sheet	charvar(20)
approval_status	charvar(20)
application_date	timestamptz
fcbc_acceptance_date	timestamptz
approval_issuance_date	timestamptz
approval_start_date	timestamptz
approval_expiry_date	timestamptz
approval_refuse_abandon_date	timestamptz
geom	geometry(Point,4326)

 dbdiagram.io

bcwat_watershed

bcast_watershed_fdc_physical	
watershed_feature_id_p	integer
lat	DOUBLE PRECISION
lon	DOUBLE PRECISION
upstream_area_km2	DOUBLE PRECISION
min_elev	DOUBLE PRECISION
avg_elev	DOUBLE PRECISION
max_elev	DOUBLE PRECISION
month	varchar(2)
ppt	DOUBLE PRECISION
tau	DOUBLE PRECISION
pas	DOUBLE PRECISION

bcast_watershed_fdc_distance	
watershed_feature_id_p	integer
catchdate_p	text
month01	DOUBLE PRECISION
month02	DOUBLE PRECISION
month03	DOUBLE PRECISION
month04	DOUBLE PRECISION
month05	DOUBLE PRECISION
month06	DOUBLE PRECISION
month07	DOUBLE PRECISION
month08	DOUBLE PRECISION
month09	DOUBLE PRECISION
month10	DOUBLE PRECISION
month11	DOUBLE PRECISION
month12	DOUBLE PRECISION

bcast_watershed_vs_geom_atl_report	
watershed_feature_id_p	integer
feat_watershed_code	text
local_watershed_code	text
upstream_geom_4326_212	geometry(Geometry,4326)
area	DOUBLE PRECISION
len4326	DOUBLE PRECISION
lat4326	DOUBLE PRECISION
gfs_name	text
upstream_geom4326	geometry(Geometry,4326)
local_watershed_order	varchar

bcast_watershed_fdc	
watershed_feature_id_p	integer
month_p	varchar
perc0	varchar
c1	text
q_m3s_c1	DOUBLE PRECISION
c2	text
q_m3s_c2	DOUBLE PRECISION
c3	text
q_m3s_c3	DOUBLE PRECISION
perc_atl	varchar
q_m3s_c1_atl	DOUBLE PRECISION

bcast_watershed_fdc_union	
feat_watershed_code_p	text
geom	geometry(MultiPolygon,3005)
geom_simplified	geometry(MultiPolygon,3005)
geom4326_simplified	geometry(MultiPolygon,4326)

bcast_watershed_fdc_fund	
watershed_feature_id_p	integer
feat_watershed_code	text
local_watershed_code	text
lake_name	text
report	varchar
feat_upstream	boolean
fit_study_area	boolean
point_mode_poly	geometry(Point,3005)
lake	boolean
area	DOUBLE PRECISION
geom4326	geometry(MultiPolygon,4326)
watershed_feature_id_boundary_area	integer

bcast_watershed_lake	
watershed_poly_id_p	integer
geom4326	geometry(MultiPolygon,4326)
geom3005	geometry(MultiPolygon,3005)
gfs_name_1	text
area_m2	DOUBLE PRECISION
feat_watershed_code	text
local_watershed_code	text
geom4326_buffer_100	geometry(MultiPolygon,4326)
vector_offset_m3	DOUBLE PRECISION

bcast_watershed_geo_feature	
gid_p	integer
geomname	text
x	numeric
y	numeric
zoom	integer
geomcomment	text
concatcode	text

bcast_watershed_fdc_stream_name_unique	
feat_watershed_code_p	text
gfs_name	text

bcast_watershed_fdc_stream_name	
feat_watershed_id_p	integer
feat_watershed_code	text
gfs_name	text
stream_imagecode	integer
geom	geometry(MultiLineString,3005)
st_point_on_line	geometry(Point,3005)

bcast_watershed_fund_rctup	
watershed_feature_id_p	integer
upstream_area	DOUBLE PRECISION
den400	DOUBLE PRECISION
ppt01	DOUBLE PRECISION
ppt02	DOUBLE PRECISION
ppt03	DOUBLE PRECISION
ppt04	DOUBLE PRECISION
ppt05	DOUBLE PRECISION
ppt06	DOUBLE PRECISION
ppt07	DOUBLE PRECISION
ppt08	DOUBLE PRECISION
ppt09	DOUBLE PRECISION
ppt10	DOUBLE PRECISION
ppt11	DOUBLE PRECISION
ppt12	DOUBLE PRECISION
tau01	DOUBLE PRECISION
tau02	DOUBLE PRECISION
tau03	DOUBLE PRECISION
tau04	DOUBLE PRECISION
tau05	DOUBLE PRECISION
tau06	DOUBLE PRECISION
tau07	DOUBLE PRECISION
tau08	DOUBLE PRECISION
tau09	DOUBLE PRECISION
tau10	DOUBLE PRECISION
tau11	DOUBLE PRECISION
tau12	DOUBLE PRECISION
pnt01	DOUBLE PRECISION
pnt02	DOUBLE PRECISION
pnt03	DOUBLE PRECISION
pnt04	DOUBLE PRECISION
pnt05	DOUBLE PRECISION
pnt06	DOUBLE PRECISION
pnt07	DOUBLE PRECISION
pnt08	DOUBLE PRECISION
pnt09	DOUBLE PRECISION
pnt10	DOUBLE PRECISION
pnt11	DOUBLE PRECISION
pnt12	DOUBLE PRECISION
pnt1_a2	DOUBLE PRECISION
pnt1_b1	DOUBLE PRECISION
pnt1_a1b	DOUBLE PRECISION
pnt2_a2	DOUBLE PRECISION
pnt2_b1	DOUBLE PRECISION
pnt2_a1b	DOUBLE PRECISION
pnt3_a2	DOUBLE PRECISION
pnt3_b1	DOUBLE PRECISION
pnt3_a1b	DOUBLE PRECISION
pnt4_a2	DOUBLE PRECISION
pnt4_b1	DOUBLE PRECISION
pnt4_a1b	DOUBLE PRECISION
pnt5_a2	DOUBLE PRECISION
pnt5_b1	DOUBLE PRECISION
pnt5_a1b	DOUBLE PRECISION
pnt6_a2	DOUBLE PRECISION
pnt6_b1	DOUBLE PRECISION
pnt6_a1b	DOUBLE PRECISION
pnt7_a2	DOUBLE PRECISION
pnt7_b1	DOUBLE PRECISION
pnt7_a1b	DOUBLE PRECISION
pnt8_a2	DOUBLE PRECISION
pnt8_b1	DOUBLE PRECISION
pnt8_a1b	DOUBLE PRECISION
pnt9_a2	DOUBLE PRECISION
pnt9_b1	DOUBLE PRECISION
pnt9_a1b	DOUBLE PRECISION
pnt10_a2	DOUBLE PRECISION
pnt10_b1	DOUBLE PRECISION
pnt10_a1b	DOUBLE PRECISION
pnt11_a2	DOUBLE PRECISION
pnt11_b1	DOUBLE PRECISION
pnt11_a1b	DOUBLE PRECISION
pnt12_a2	DOUBLE PRECISION
pnt12_b1	DOUBLE PRECISION
pnt12_a1b	DOUBLE PRECISION
pnt1_a2	DOUBLE PRECISION
pnt1_b1	DOUBLE PRECISION
pnt1_a1b	DOUBLE PRECISION
pnt2_a2	DOUBLE PRECISION
pnt2_b1	DOUBLE PRECISION
pnt2_a1b	DOUBLE PRECISION
pnt3_a2	DOUBLE PRECISION
pnt3_b1	DOUBLE PRECISION
pnt3_a1b	DOUBLE PRECISION
pnt4_a2	DOUBLE PRECISION
pnt4_b1	DOUBLE PRECISION
pnt4_a1b	DOUBLE PRECISION
pnt5_a2	DOUBLE PRECISION
pnt5_b1	DOUBLE PRECISION
pnt5_a1b	DOUBLE PRECISION
pnt6_a2	DOUBLE PRECISION
pnt6_b1	DOUBLE PRECISION
pnt6_a1b	DOUBLE PRECISION
pnt7_a2	DOUBLE PRECISION
pnt7_b1	DOUBLE PRECISION
pnt7_a1b	DOUBLE PRECISION
pnt8_a2	DOUBLE PRECISION
pnt8_b1	DOUBLE PRECISION
pnt8_a1b	DOUBLE PRECISION
pnt9_a2	DOUBLE PRECISION
pnt9_b1	DOUBLE PRECISION
pnt9_a1b	DOUBLE PRECISION
pnt10_a2	DOUBLE PRECISION
pnt10_b1	DOUBLE PRECISION
pnt10_a1b	DOUBLE PRECISION
pnt11_a2	DOUBLE PRECISION
pnt11_b1	DOUBLE PRECISION
pnt11_a1b	DOUBLE PRECISION
pnt12_a2	DOUBLE PRECISION
pnt12_b1	DOUBLE PRECISION
pnt12_a1b	DOUBLE PRECISION
downstream_id	integer
summer_sensitivity	boolean
winter_sensitivity	boolean
etec	DOUBLE PRECISION
mginc_pg	DOUBLE PRECISION
mginc_m	DOUBLE PRECISION
gfs_name	text
downstream_area	DOUBLE PRECISION
quary_polygon	geom
mginc_polygon	geom
vel_area_kms	numeric
local_watershed_order	varchar

bcast_watershed_fund_rctup_report	
watershed_feature_id_p	integer
watershedarea	numeric
den400	DOUBLE PRECISION
ppt_min_tot	DOUBLE PRECISION
tau_min_tot	DOUBLE PRECISION
pnt_min_tot	DOUBLE PRECISION
vel_min_fut_max	DOUBLE PRECISION
vel_min_fut_max	DOUBLE PRECISION
ppt_min_fut_max	DOUBLE PRECISION
ppt_min_fut_max	DOUBLE PRECISION
tau_min_fut_max	DOUBLE PRECISION
tau_min_fut_max	DOUBLE PRECISION
vel_min_fut_max	DOUBLE PRECISION
strub	DOUBLE PRECISION
grassland	DOUBLE PRECISION
coniferous	DOUBLE PRECISION
water	DOUBLE PRECISION
snow	DOUBLE PRECISION
developed	DOUBLE PRECISION
wetland	DOUBLE PRECISION
herb	DOUBLE PRECISION
deciduous	DOUBLE PRECISION
mixed	DOUBLE PRECISION
barren	DOUBLE PRECISION
cropland	DOUBLE PRECISION
road_m3s	numeric
road_m3yr	numeric
etmon_m3s	numeric
rr	boolean
downstream_id	integer
summer_sensitivity	boolean
winter_sensitivity	boolean
etec	DOUBLE PRECISION
mginc_pg	DOUBLE PRECISION
mginc_m	DOUBLE PRECISION
gfs_name	text
downstream_area	DOUBLE PRECISION
quary_polygon	geom
mginc_polygon	geom
vel_area_kms	numeric
local_watershed_order	varchar

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

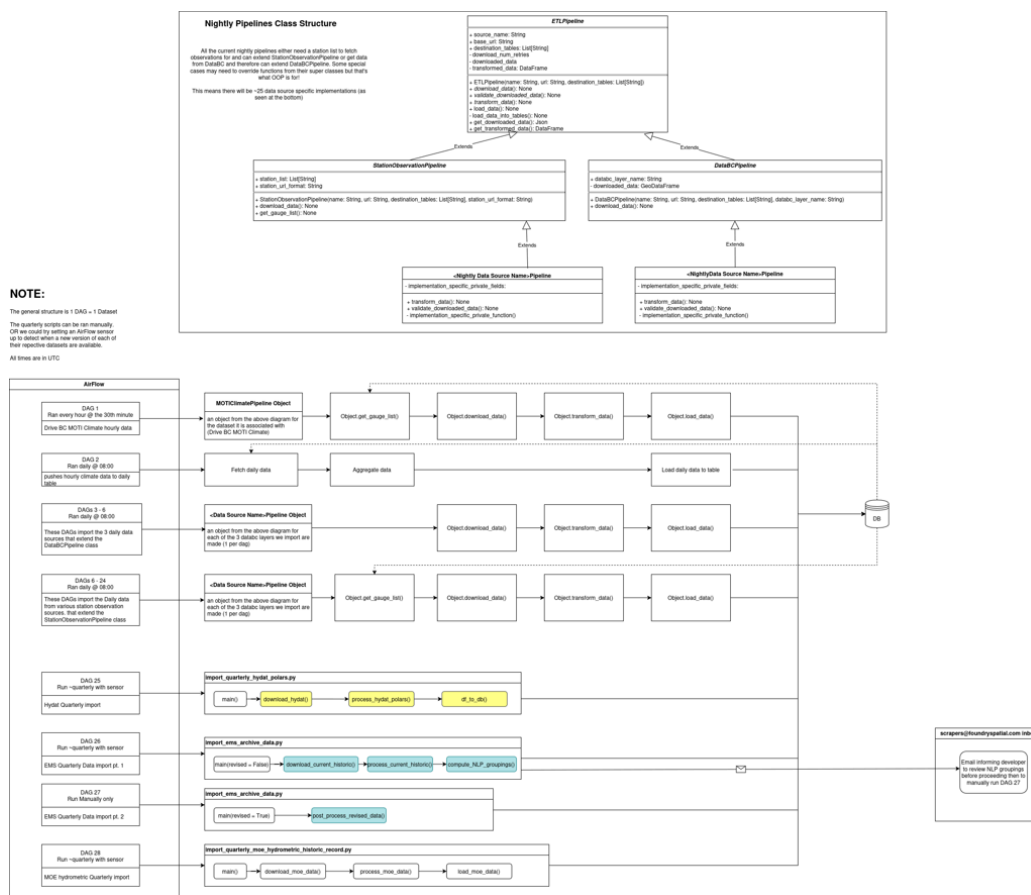
Added Foreign keys everywhere where it was possible. If a table didn't have a primary key, a surrogate key has been added to make sure that we get as close to possible as 3NF.

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!