Development tasks for ‘dlairflow‘ package

Robert Nikutta, Benjamin Weaver

Last modified: 2025-03-29

Contents

1 Assumptions NOSHOW

Development of the `dlairflow` package makes the following global assumptions:

- The package is generic, such that it can be used on any system (with the appropriate configuration file).
- The actual staging and production database engines and machines are not hard-coded, but rather configurable.
- All sub-modules and methods are written in a generic sense. Example: `ingest_csv_file(*args)` is good, `ingest_sdss_dr12_csv_file(*args)` is not.
- For each dataset's metadata there is only a single source of truth (ideally a Felis .yaml file). All operations can rely on the Felis file being correct, complete, and can validate actual data against the Felis truth file.
- This implies that all metadata information necessary to, e.g., create schemas, tables, columns, indices, run quality assurance tasks, etc., can be looked or derived from the Felis .yaml file.
- Each task name described in this document is a template / suggestion at first, and should be marked with a DEFINITION property. The possible values are: tbw (nothing defined beyond the need for this functionality), draft (definition is work in progress), production (defined, implemented, deployed).
- The mandatory and optional arguments for every methods are not yet defined in most cases.

2 Functionality

2.1 Metadata handling and initial quality assurance show

2.1.1 TODO `meta.validate_schema_file()`

Validate Felis yaml file (is the file syntactically correct?).

- The tests (done by Felis calls) likely include: self-consistency, column datatypes and UCDs have allowed values, etc.
- The command to use is likely: `felis validate [options] schema.yaml`

See https://felis.lsst.io/user-guide/validation.html

2.1.2 TODO `meta.get()`

Extract schema/tables/column metadata from Felis yaml file.

- Should be flexible to extract and return any of these:
  - tables –> list
  - columns –> list
  - column –> dict of column name|dtype|ucd|description.

2.1.3 TODO `meta.validate_data_files()`

Validate initial data against its Felis yaml file (are the data and the yaml file compatible?.

2.1.4 TODO `meta.validate_db_schema()`

Validate DB contents against its Felis yaml file.
Ensure that all of the following are true:

- All tables and columns as defined in the yaml file for a given schema are present in the DB under that schema.
- No additional tables and column are present in the DB which aren't part of the yaml schema file.
- All column datatypes in the DB correspond to the datatypes defined in the yaml file.
- All columns in the TapSchema in the DB have a column description, and that it is identical to the column descriptions in the yaml file.
- Ensure that for every column in the DB that has UCD defined, the USD corresponds to tyhe one defined for said column in the yaml file.

2.1.5 TODO `meta.get_counts()`

Count and return number of tables, columns per table, rows per table.

- This method could do it based on files, and based on a DB schema.

2.2 Data file transformations show

2.2.1 TODO `file.add_data_column()`

Add arbitrary data column.

- For FITS bintable files: use STILTS.

2.2.2 TODO `file_add_coords()`

/Add a pair of coordinate columns to the FITS bintable files, using STILTS.\

- Usually, we add glon/glat, and elon/elat.

2.2.3 TODO `file.add_spatial_pixels()`

Compute and add columns for spatial indexing.

- This is atypical addition, and usually adds `nest4096`, `ring256`, and `htm9` columns.

2.3 DB operations show

2.3.1 TODO `db.create_schema()`

Create schemas, tables, columns (from Felis yaml file).

- If Felis, command is: `felis create schema.yaml` (or use API).
- See https://felis.lsst.io/user-guide/databases.html

2.3.2 TODO `db.load_tap_schema()`

Create/load TapSchema.

- Use `felis load-tap-schema` (or API).
- See https://felis.lsst.io/user-guide/tap.html

2.3.3 TODO `db.load_table_from_files()`

Load data from file(s) to DB table.

- The file can be either a path to a single file, or to a directory of files.
- The file format can be either:
  - .fits|.fits.gz|.fz (highest implementation prioriry)
  - .csv|.ecsv
  - .parquet
- Loading with fits2db should allow to pass various fits2db flags, including
  - `--rid=random_id` (creates a column with uniformly distrobuted random double-precision floats between 0.0 and 100.)
  - `-b` (binary mode; preserves precision, MUCH faster loading; but we need to investigate behaviour when string-valued columns are present)

2.3.4 TODO `db.add_column()`

Define a new colum in a DB table.

- Issued command syntax is: `ALTER TABLE ${table} ADD ${column} ${datatype}`

2.3.5 TODO `db.insert_into_column()`

Create a new column in the DB table, and insert new records into it.

- Issued command syntax is:

  ```
  ALTER TABLE ${table} ADD ${column} ${datatype}
  INSERT INTO ${table} (${column}) VALUES ${values}
  ```

- First step can use `db.add_column()`.
- Investigate also whether `COPY` is the better/faster method to add a new column to a table.

2.3.6 TODO `db.drop_column()`

Drop colum from DB table.

- Issued command syntax is: `ALTER TABLE ${table} DROP COLUMN ${column}`

2.3.7 TODO `db.create_index()`

Create column index.

- Assumes this is a B-tree index (default in Postgres) for a single column.
- Can take `unique=True` arg, and then executes `CREATE UNIQUE INDEX [...]`
- Can take q3c=True and ra=racol, dec=deccol, and then executes

  ```
  CREATE INDEX ${table}_q3c_idx ON ${schema}.${table}(q3c_ang2ipix(${racol},${deccol})) TABLESPACE
  ${tablespace}
  ```

2.3.8 TODO `db.cluster_table()`

Cluster a table.

2.3.9 TODO `db.vaccum_analyze()`

Vacuum analyze a table.

2.3.10 TODO `db.grant_permission()`

Grant permission.

2.3.11 TODO `db.alter_table()`

Alter a table.

2.3.12 TODO `db.run_sql()`

Run arbitrary SQL.

- Other SQL-executing functions (e.g. `db.alter_table()`, `db.create_index()` could be calling this generic SQL execution function, but with the correct/validated/sanitized arguments.

2.4 Post-ingest quality assurance show

2.4.1 TODO task 1

2.4.2 TODO task 2

2.5 DB connectivity show

2.5.1 TODO task 1

2.5.2 TODO task 2

2.6 Unit testing show

2.6.1 DONE Set up CI for unit testing

2.6.2 TODO Tests that can run locally

3 Task overview table

ITEM	STATUS	PRIORITY	DEFINITION	TAGS
Metadata handling and initial quality assurance		B		:show:
`meta.validate_schema_file()`	TODO	B	tbw	
`meta.get()`	TODO	B	tbw	
`meta.validate_data_files()`	TODO	B	tbw	
`meta.validate_db_schema()`	TODO	B	tbw	
`meta.get_counts()`	TODO	B	tbw	
Data file transformations		B		:show:
`file.add_data_column()`	TODO	B	tbw	
`file_add_coords()`	TODO	B	tbw	
`file.add_spatial_pixels()`	TODO	B	tbw	
DB operations		B		:show:
`db.create_schema()`	TODO	B	tbw	
`db.load_tap_schema()`	TODO	B	tbw	
`db.load_table_from_files()`	TODO	B	tbw	
`db.add_column()`	TODO	B	tbw	
`db.insert_into_column()`	TODO	B	tbw	
`db.drop_column()`	TODO	B	tbw	
`db.create_index()`	TODO	B	tbw	
`db.cluster_table()`	TODO	B	tbw	
`db.vaccum_analyze()`	TODO	B	tbw	
`db.grant_permission()`	TODO	B	tbw	
`db.alter_table()`	TODO	B	tbw	
`db.run_sql()`	TODO	B	tbw	
Post-ingest quality assurance		B		:show:
task 1	TODO	B	tbw	
task 2	TODO	B	tbw	
DB connectivity		B		:show:
task 1	TODO	B	tbw	
task 2	TODO	B	tbw	
Unit testing		B		:show:
Set up CI for unit testing	DONE	B	tbw	
Tests that can run locally	TODO	B	tbw	