UDACITY

< Return to "Data Engineering Nanodegree" in the classroom

Data Pipelines with Airflow

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

General

DAG can be browsed without issues in the Airflow UI

The DAG has been successfully loaded

The dag follows the data flow provided in the instructions, all the tasks have a dependency and DAG begins with a start_execution task and ends with a end_execution task.

Dag configuration

DAG contains default_args dict, with the following keys:

- **Owner**
- **Depends_on_past**
- **Start_date**
- **Retries**
- **Retry_delay**
- **Catchup**

All the required keys are present in the DAG's default dictionary

The DAG object has default args set

The arguments in bound to the DAG object. Good Job!

The DAG should be scheduled to run once an hour

`@hourly` preset has been used. Alternatively, `0 * * * *` cron reset can also be used

Staging the data

There is a task that to stages data from S3 to Redshift. (Runs a Redshift copy statement)

Data has been successfully loaded to the STG tables in Redshift

Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically

The task contains a set of parameters that are used to define the S3_key, S3_bucket, target table. Good Job!

The operator contains logging in different steps of the execution

Proper log messages has been used

The SQL statements are executed by using a Airflow hook

Good job using the hook provided by airflow to execute the SQL statements

Loading dimensions and facts

Dimensions are loaded with on the LoadDimension operator

`LoadDimension` functional Operator has been used to load the dimensions and is well implemented

Facts are loaded with on the LoadFact operator

`LoadFact` functional Operator has been used to load the facts table and is well implemented

Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically

The operators are flexible and not restricted to a particular SQL statement

The DAG allows to switch between append-only and delete-load functionality

The dimension operator must allow the user to switch between append and truncate-insert. So, basically you can pass a argument to the operator to append/truncate-insert and then accordingly perform the task. It can be something like this:

```
if self.append_data == True:
            sql_statement = <sql_statement to insert into table>
            redshift.run(sql_statement)
        else:
            sql_statement = <sql_statement to truncate the
            sql_statement = <sql_statement to insert into table>
            redshift.run(sql_statement)
```

Data Quality Checks

Data quality check is done with correct operator

The data quality check has been successfully implemented

The DAG either fails or retries n times

The operator does fails in case the test does not pass which is an expected behaviour

Operator uses params to get the tests and the results, tests are not hard coded to the operator

The Data quality checks should not be hardcoded inside the operator. It should be passed as a param. It can be something like this:

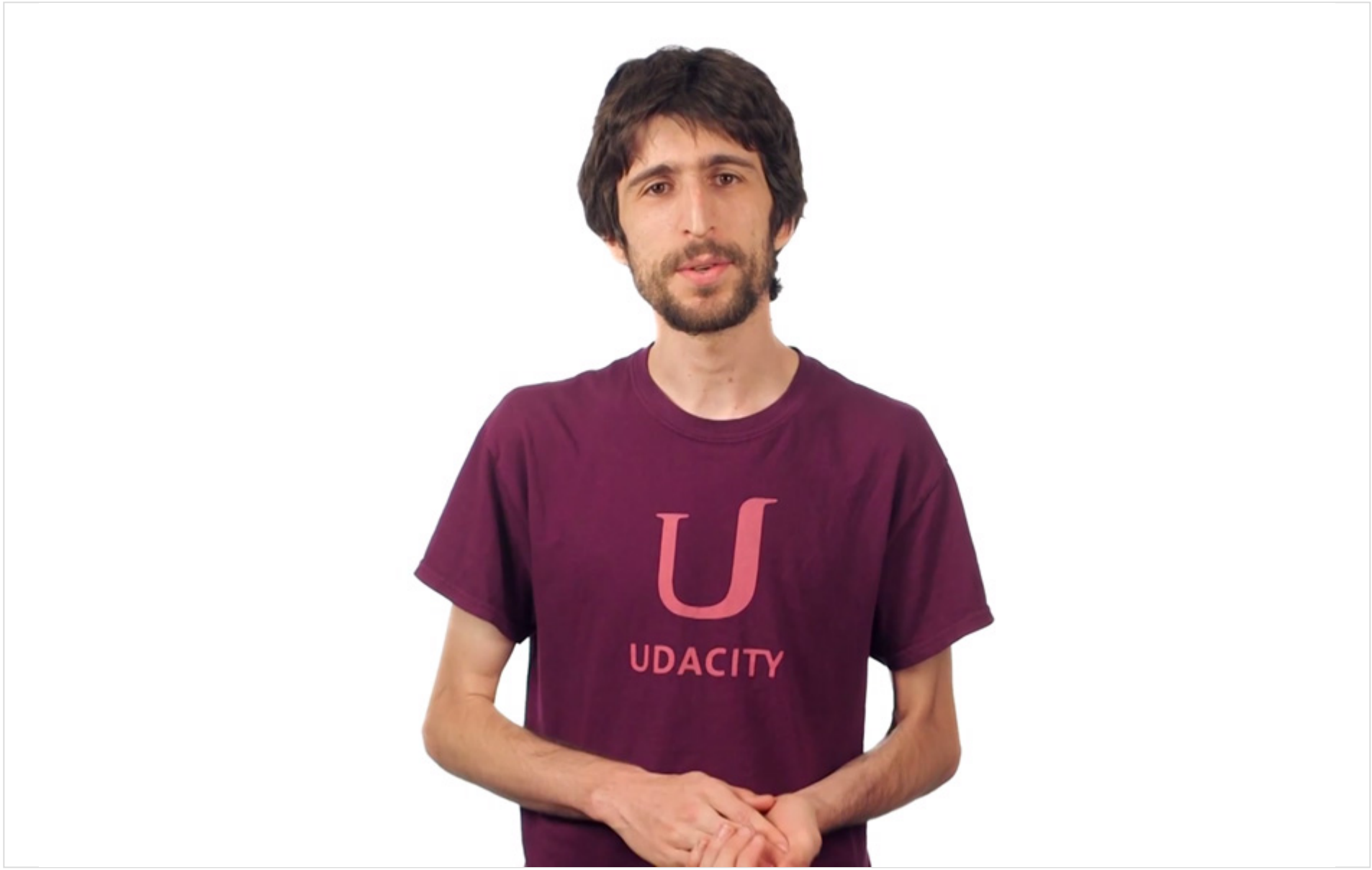
```
for check in <checks_passed_as_param>:
            sql = check.get('check_sql')
            exp_result = check.get('expected_result')

            records = redshift.get_records(sql)[0]

            if exp_result != records[0]:
                error_count += 1
                failing_tests.append(sql)
```

RESUBMIT

DOWNLOAD PROJECT



Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review