

## SQL Airflow pipelines

We will start by describing a simple DAG that will perform a series of simple steps in a Postgres table (create the table, load some data, count them and, finally, drop or delete the table randomly). The implementation of all DAGs are placed on the `dags` directory of the git repository and, in this particular case, the sql comands and even the csv data will be placed in a separate `sql` subdirectory:

```
# You may need to change the cd command in order to be in the right directory  
# Execute this cell
```

```
cd ../dags/  
ls -l sql*
```

If everything went well, you will see an output similar to this:

```
# Do not execute this cell. Just for information  
-rw-r--r--  1 Angel  wheel  3372 Mar 20 15:09 sql_airflow_dag.py  
  
sql:  
total 312  
-rw-r--r--@ 1 Angel  wheel 137845 Mar 13 11:06 motogp.csv  
-rw-r--r--@ 1 Angel  wheel    181 Mar 13 11:06 motogp_create_table.sql  
-rw-r--r--  1 Angel  wheel     21 Mar 13 11:06 motogp_delete_table.sql  
-rw-r--r--  1 Angel  wheel     20 Mar 13 11:06 motogp_drop_table.sql  
-rw-r--r--  1 Angel  wheel    469 Mar 13 11:06 motogp_load_table.py  
-rw-r--r--@ 1 Angel  wheel     30 Mar 13 11:06 motogp_select_table.sql
```

More interesting than the contents of the `*.sql` files (which are simple sql statements) is the file `sql_airflow_dag.py`. Even if you are not a python programmer or have no Airflow skills, it is advisable to review the stucture of the code to understand what and how the DAG will do:

<pre> 1 # File: sql_airflow_dag.py 2 # Simple DAG for the Databand hands-on workshop 3 4 # It must be embedded in Airflow to run properly, not using the command line 5 # Copy this file in the dags directory and Airflow will recognize it as a DAG 6 7 # Create a test table, load data from csv, select data, delete data or drop the table 8 # Featuring: 9 # - Airflow tasks 10 # - Three Airflow operators (bash / Postgres / branch) 11 # - all tasks implemented with the Postgres operator and *.sql files but: 12 #   - exception 1: load by bash operator invoking a python script (not possible with an *.sql file) 13 #   - exception 2: decision delete/drop by branch operator 14 # - NO DATABAND CODE at all. Metadata collected automatically, 15 16 # These are mandatory imports 17 from __future__ import annotations 18 from airflow import DAG 19 from airflow.operators.bash import BashOperator 20 from airflow.operators.python import BranchPythonOperator 21 from airflow.operators.postgres_operator import PostgresOperator 22 from datetime import datetime, timedelta 23 from random import random </pre>	header comments and imports
<pre> 24 25 # An auxiliary function to decide either drop a table or delete its contents 26 def delete_or_drop(): 27     if random() &lt; 0.5: 28         return("SQL_delete_records") 29     else: 30         return("SQL_drop_table") 31 </pre>	auxiliary function
<pre> 32 # The header of the DAG with all its tasks. 33 # Notice the argument postgres_conn_id: 34 # It must match a connection name in Airflow connection menu 35 with DAG( 36     dag_id="SQL_Airflow_DAG", 37     start_date=datetime(2023, 1, 1), 38     schedule_interval=timedelta(minutes=7), 39     catchup=False, 40     default_args = {'owner': 'Angel'}, 41     tags=[ 42         "project: SQL Airflow pipelines " 43     ], 44 ) as dag: </pre>	DAG header
<pre> 45 # Create table. Column definition inside the auxiliar *.sql file 46 SQL_create_table = PostgresOperator ( 47     task_id="SQL_create_table", 48     postgres_conn_id="postgres_motogp", 49     sql="sql/motogp_create_table.sql" 50 ) </pre>	Task #1: create
<pre> 51 # Load data from a *.csv file. Note that the Postgres loader must be 52 # invoked by a python script. Otherwise, Airflow raises an error. 53 # Airflow will spawn a shell and run the python script as indicated 54 SQL_load_table = BashOperator ( 55     task_id="SQL_load_table", 56     bash_command="python3 /opt/airflow/dags/sql/motogp_load_table.py" 57 ) </pre>	Task #2: load
<pre> 58 # Select some data that goes to nowhere but Databand will track 59 # the details of the operation automatically 60 SQL_select_table = PostgresOperator ( 61     task_id="SQL_select_table", 62     postgres_conn_id="postgres_motogp", 63     sql="sql/motogp_select_table.sql" 64 ) </pre>	Task #3: select
<pre> 65 # Task to decide if we drop the table or delete the contents 66 # the decision is made randomly 67 Branch_Drop_Delete = BranchPythonOperator( 68     task_id="Branch_Drop_Delete", 69     python_callable=delete_or_drop 70 ) </pre>	Task #4: branch
<pre> 71 # Delete from table... 72 SQL_delete_records = PostgresOperator ( 73     task_id="SQL_delete_records", 74     postgres_conn_id="postgres_motogp", 75     sql="sql/motogp_delete_table.sql" 76 ) </pre>	Task #5: delete
<pre> 77 # ... or drop table 78 SQL_drop_table = PostgresOperator ( 79     task_id="SQL_drop_table", 80     postgres_conn_id="postgres_motogp", 81     sql="sql/motogp_drop_table.sql" 82 ) </pre>	Task #6: drop
<pre> 83 84 # These are the task dependencies written with Airflow syntax 85 SQL_create_table &gt;&gt; SQL_load_table &gt;&gt; SQL_select_table &gt;&gt; Branch_Drop_Delete 86 Branch_Drop_Delete &gt;&gt; SQL_delete_records 87 Branch_Drop_Delete &gt;&gt; SQL_drop_table </pre>	Execution sequence

As we copied this file to the Airflow containers in the previous section of this workshop, the DAG will be visible on the Airflow console.

This is the DAG. Notice the tags of the project and the owner match the names written in the python code

slide the button to the right to enable the schedule

or trigger the dag to run it manually

Follow the instructions of the picture above to run the DAG and click on the name of the DAG to see a graphical representation (note that the **graph** tab is highlighted)

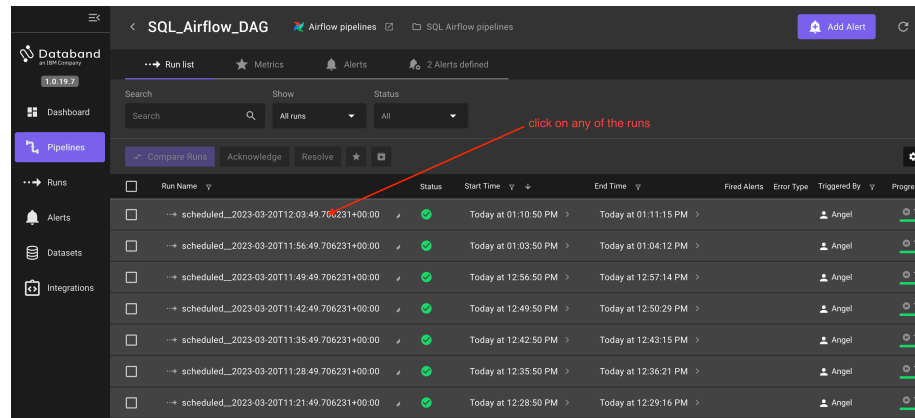
If you open the Databand main interface and navigate to the Pipelines menu on the left, you will see a list of all the pipelines, including the one we are focusing now, labeled as `SQL_Airflow_DAG`

This is the Airflow SQL pipeline

look at the names of the owner and the project

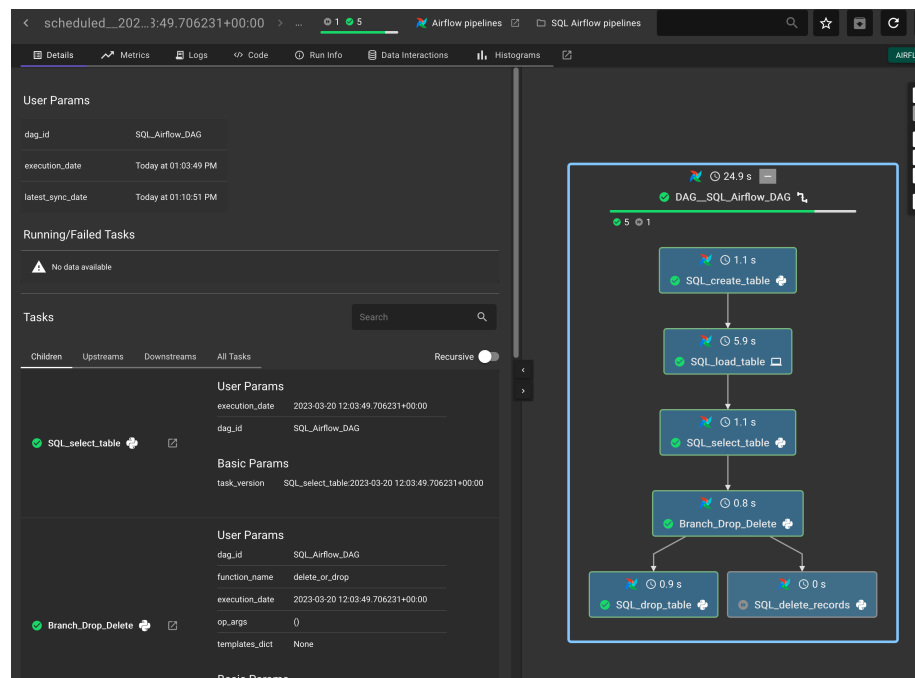
click here

If you click on the name of the pipeline, all the executions of this pipeline will be listed:



Run Name	Status	Start Time	End Time	Fired Alerts	Error Type	Triggered By	Progress
→ scheduled_2023-03-20T12:03:49.706231+00:00	✓	Today at 01:10:50 PM	Today at 01:11:15 PM			Angel	100%
→ scheduled_2023-03-20T11:56:49.706231+00:00	✓	Today at 01:03:50 PM	Today at 01:04:12 PM			Angel	100%
→ scheduled_2023-03-20T11:49:49.706231+00:00	✓	Today at 12:56:50 PM	Today at 12:57:14 PM			Angel	100%
→ scheduled_2023-03-20T11:42:49.706231+00:00	✓	Today at 12:49:50 PM	Today at 12:50:29 PM			Angel	100%
→ scheduled_2023-03-20T11:35:49.706231+00:00	✓	Today at 12:42:50 PM	Today at 12:43:15 PM			Angel	100%
→ scheduled_2023-03-20T11:28:49.706231+00:00	✓	Today at 12:35:50 PM	Today at 12:36:21 PM			Angel	100%
→ scheduled_2023-03-20T11:21:49.706231+00:00	✓	Today at 12:28:50 PM	Today at 12:29:16 PM			Angel	100%

To see the details of each run, click on anyone of them:



It is important to remark that this DAG has no sign of Databand at all, i.e. we didn't write special line in the code and nothing implies that it will be monitored by Databand. Actually, it will be scheduled and run by Airflow, which will capture the execution data as any other DAG. The execution data will be pulled by Databand to display it as a pipeline.

The information collected by Databand will include the elapsed runtimes of each

task and its return codes. This is a basic start that will be enhanced in the next chapters where we will see more valuable information.

---

[Next Section: Python pipelines](#)

[Previous Section: Preparation](#)

[Return to main](#)