Файл   Редактировать   Поиск   Запрос   Инструменты   Переход   Помощь

Donate

Фильтр баз данны   Фильтр таблиц | Хост: 127.0.0.1 | База данных: spark | Таблица: Weather_Moscow | Данные | Запрос*

- Unnamed-1
  - Airflow
  - information_schema
  - mysql
  - performance_schema
  - spark   576,0 KiB
    - HW_2_task_3   384,0 KiB
    - HW_4_all   64,0 KiB
    - Payments   64,0 KiB
    - tasketl1a   16,0 KiB
    - tasketl1b   16,0 KiB
    - Temperature_Vladivos…   16,0 KiB
    - Weather_Moscow   16,0 KiB
  - sys

spark.Weather_Moscow: 2 строк (exact) | Далее | Показать все | Сортировка | Столбцы (3/3) | Фильтр

#	source	temperature	date_time
1	OpenWeatherMap	11,59	2024-04-27 12:01:14
2	Yandex.Weather	12	2024-04-27 12:14:51

Фильтр   Регулярное выражение

```
85  SELECT * FROM information_schema.REFERENTIAL_CONSTRAINTS WHERE   CONSTRAINT_SCHEMA='spark'   AND TABLE_NAME='Weather_Moscow'   AND REFERENCED_TABLE_NAME IS NOT NULL;
86  SELECT * FROM information_schema.KEY_COLUMN_USAGE WHERE   TABLE_SCHEMA='spark'   AND TABLE_NAME='Weather_Moscow'   AND REFERENCED_TABLE_NAME IS NOT NULL;
87  SHOW CREATE TABLE `spark`.`Weather_Moscow`;
88  SELECT tc.CONSTRAINT_NAME, cc.CHECK_CLAUSE FROM `information_schema`.`CHECK_CONSTRAINTS` AS cc, `information_schema`.`TABLE_CONSTRAINTS` AS tc WHERE tc.CONSTRAINT_SCHEMA='spark' AND tc.TABLE_NAME='Weather_Moscow' AND tc.CONSTRAINT_TYPE='CHECK' AND tc.CONSTRAINT_S
89  SELECT * FROM `spark`.`Weather_Moscow` LIMIT 1000;
```

r1 : c2 | Подключено: 00:16 h | MariaDB or MySQL 8.0.36   Время работы: 03:13 h | Серверное время: ⏱ Ожидание.

```python
import pandas as pd
from sqlalchemy import inspect, create_engine, Table, Column, Float, TIMESTAMP, MetaData, String
from datetime import datetime, timedelta
from pandas.io import sql
import pendulum
import requests


    # Функция для получения температуры и даты со временем из API OpenWeatherMap
def get_openweather_data(api_key, city):
    url = f'https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}'
    response = requests.get(url)
    data = response.json()
    temperature = data['main']['temp']
    timestamp = data['dt']
    date_time = datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')
    return round(float(temperature) - 273.15, 2), date_time


    # Функция для получения температуры и даты со временем из API Yandex
def get_yandex_weather_data(api_key, lat, lon):
    url = f"https://api.weather.yandex.ru/v2/informers?lat={lat}&lon={lon}"
    headers = {'X-Yandex-API-Key': api_key}
    response = requests.get(url, headers=headers)
    data = response.json()

    try:
        temperature = data['fact']['temp']
        timestamp = data['now']
        date_time = datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')
        return temperature, date_time
    except KeyError as e:
        print(f"KeyError: {e}")
        return None, None

    # API ключи и координаты для запросов
openweather_api_key = '2cd78e55c423fc81cebc1487134a6300'
yandex_weather_api_key = '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
city = 'Moscow'
lat = '55.75396'
lon = '37.620393'

openweather_temperature, openweather_time = get_openweather_data(openweather_api_key, city)
yandex_temperature, yandex_time = get_yandex_weather_data(yandex_weather_api_key, lat, lon)


    # Подключение к БД, создание таблицы, получение данных и сохранение их в базу данных
con = create_engine("mysql://Airflow:1@localhost/spark")
metadata = MetaData()

weather_table = Table(
    'Weather_Moscow',
    metadata,
```

```python
        return None, None

    # API ключи и координаты для запросов
openweather_api_key = '2cd78e55c423fc81cebc1487134a6300'
yandex_weather_api_key = '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
city = 'Moscow'
lat = '55.75396'
lon = '37.620393'

openweather_temperature, openweather_time = get_openweather_data(openweather_api_key, city)
yandex_temperature, yandex_time = get_yandex_weather_data(yandex_weather_api_key, lat, lon)


    # Подключение к БД, создание таблицы, получение данных и сохранение их в базу данных
con = create_engine("mysql://Airflow:1@localhost/spark")
metadata = MetaData()

weather_table = Table(
    'Weather_Moscow',
    metadata,
    Column('source', String(50), nullable=True),
    Column('temperature', Float, nullable=True),
    Column('date_time', TIMESTAMP, nullable=True)
)

metadata.create_all(con)

ins = weather_table.insert()
values = [
    {"source": "OpenWeatherMap", "temperature": openweather_temperature, "date_time": openweather_time},
    {"source": "Yandex.Weather", "temperature": yandex_temperature, "date_time": yandex_time}
]

with con.connect() as connection:
    connection.execute(ins, values)
```

Airflow

DAGs   Cluster Activity   Datasets   Security ▾   Browse ▾   Admin ▾   Docs ▾   08:53 UTC ▾   AA ▾

DAG: **HW_8_weather-telegram**

Schedule: @once ⓘ   Next Run: None

⊞ Grid   🔗 Graph   📅 Calendar   ⧗ Task Duration   ⇄ Task Tries   📈 Landing Times   ☰ Gantt   ⚠ Details   <> Code   🔍 Audit Log   ▶ 🗑

| 29.04.2024 08:53:06 📅 | 25 ▾ | All Run Types ▾ | All Run States ▾ | **Clear Filters** | | Auto-refresh ⬤ |

Press `shift` + `/` for Shortcuts

deferred   failed   queued   removed   restarting   running   scheduled   skipped   success   up_for_reschedule   up_for_retry   upstream_failed   no_status

《   》 DAG
HW_8_weather-telegram

⚠ Details   🔗 Graph   ☰ Gantt   <> Code

DAG Runs Summary

Total Runs Displayed	1
■ Total success	1
First Run Start	2024-04-27, 05:39:07 UTC
Last Run Start	2024-04-27, 05:39:07 UTC
Max Run Duration	00:00:08
Mean Run Duration	00:00:08
Min Run Duration	00:00:08

DAG Summary

Duration

00:00:08

00:00:04

00:00:00

yandex_weather
open_weather
python_weather
send_message_telegram

```python
import datetime
import os
import requests
import pendulum
from airflow.decorators import dag, task
from airflow.providers.telegram.operators.telegram import TelegramOperator

os.environ["no_proxy"]="*"

@dag(
    dag_id="HW_8_weather-telegram",
    schedule="@once",
    start_date=pendulum.datetime(2023, 1, 1, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
)

def WeatherETL():

    send_message_telegram_task = TelegramOperator(
        task_id='send_message_telegram',
        telegram_conn_id='telegram_default',
        token='6739587766:AAEyp8y2-LzFLASv7Z6FZPXb1riR15vKsTA',
        chat_id='1420330277',
        text='Weather in Vladivostok \nYandex: ' + "{{ ti.xcom_pull(task_ids=['yandex_weather'],key='weather')[0]}}" + " degrees" +
        "\nOpen weather: " + "{{ ti.xcom_pull(task_ids=['open_weather'],key='open_weather')[0]}}" + " degrees",
    )

    @task(task_id='yandex_weather')
    def get_yandex_weather(**kwargs):
        ti = kwargs['ti']
        url = "https://api.weather.yandex.ru/v2/forecast?lat=43.1056&lon=131.8735"

        payload={}
        headers = {
        'X-Yandex-API-Key': '2dd736ea-1e89-4aef-a412-43a50eaeec54'
        }
        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        a=response.json()['fact']['temp']
        print(a)
        ti.xcom_push(key='weather', value=response.json()['fact']['temp'])
#        return str(a)
    @task(task_id='open_weather')
    def get_open_weather(**kwargs):
        ti = kwargs['ti']
        url = "https://api.openweathermap.org/data/2.5/weather?lat=43.1056&lon=131.8735&appid=7f1baa1d74ab9d393b97eab4b600ef44"

        payload={}
        headers = {}

        response = requests.request("GET", url, headers=headers, data=payload)
```

```python
     catchup=False,
     dagrun_timeout=datetime.timedelta(minutes=60),
)

def WeatherETL():

    send_message_telegram_task = TelegramOperator(
        task_id='send_message_telegram',
        telegram_conn_id='telegram_default',
        token='6739587766:AAEyp8y2-LzFLASv7Z6FZPXb1riR15vKsTA',
        chat_id='1420330277',
        text='Weather in Vladivostok \nYandex: ' + "{{ ti.xcom_pull(task_ids=['yandex_weather'],key='weather')[0]}}" + " degrees" +
        "\nOpen weather: " + "{{ ti.xcom_pull(task_ids=['open_weather'],key='open_weather')[0]}}" + " degrees",
    )

    @task(task_id='yandex_weather')
    def get_yandex_weather(**kwargs):
        ti = kwargs['ti']
        url = "https://api.weather.yandex.ru/v2/forecast?lat=43.1056&lon=131.8735"

        payload={}
        headers = {
        'X-Yandex-API-Key': '2dd736ea-1e89-4aef-a412-43a50eaeec54'
        }
        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        a=response.json()['fact']['temp']
        print(a)
        ti.xcom_push(key='weather', value=response.json()['fact']['temp'])
#        return str(a)
    @task(task_id='open_weather')
    def get_open_weather(**kwargs):
        ti = kwargs['ti']
        url = "https://api.openweathermap.org/data/2.5/weather?lat=43.1056&lon=131.8735&appid=7f1baa1d74ab9d393b97eab4b600ef44"

        payload={}
        headers = {}

        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        a=round(float(response.json()['main']['temp']) - 273.15, 2)
        print(a)
        ti.xcom_push(key='open_weather', value=round(float(response.json()['main']['temp']) - 273.15, 2))
#        return str(a)
    @task(task_id='python_weather')
    def get_weather(**kwargs):
        print("Yandex "+str(kwargs['ti'].xcom_pull(task_ids=['yandex_weather'],key='weather')[0])+" Open "+str(kwargs['ti'].xcom_pull(task_ids=['open_weather'],key='open_weather')[0]))

    get_yandex_weather() >> get_open_weather() >> get_weather()>> send_message_telegram_task

dag = WeatherETL()
```

27 апреля

/start 10:46

/mybots 11:10

Weather in Vladivostok
Yandex: 17 degrees
Open weather: 19.01 degrees 15:39

Написать сообщение...