

# ETL: автоматизация подготовки данных

## Урок 8. Специфика применения ETL в различных предметных сферах

### Задание

1. Используя материалы семинара и s8dag.py нужно доработать задачу в части записи данных в mysql по погоде яндекса и open weather (поля - метка текущего времени и температура).

2. Создать еще одну задачу по отправке данных в телеграмм. За основу взять данные таблиц платежей из 4-го семинара (все 360 периодов), конвертировать их в текстовый формат и отправить их в telegram.


К ДЗ приложите код и скриншоты отработанных задач аирфлоу, а также отправленный слепок из базы данных в вашем чатботе. Рассмотрите возможность применения разметки html либо markdown. Нужно выслать одну основную таблицу. Есть лимит по сообщениям, можно ограничить количество строк таблицы. Можете использовать функцию limit в sql запросе.

```
1 import datetime
2 import os
3 import requests
4 import pendulum
5 from airflow.decorators import dag, task
6 from airflow.providers.telegram.operators.telegram import TelegramOperator
7 from sqlalchemy import create_engine
8 import pandas as pd
9 from tabulate import tabulate
10
11
12 os.environ["no_proxy"] = "*"
13
14 @dag(
15     dag_id="wether-telegram-home",
16     schedule="@once",
17     start_date=pendulum.datetime(2024, 11, 23, tz="UTC"),
18     catchup=False,
19     dagrun_timeout=datetime.timedelta(minutes=60),
20 )
21
22 def WetherETL():
23
24     send_message_telegram_task = TelegramOperator(
25         task_id='send_message_telegram',
26         telegram_conn_id='telegram_default',
27         token='7672923175:AAE48c-RP3lrUOP96hEaFuKvIRwGwmdPTaE',
28         chat_id='5174647902',
29         text='Wether in Baranovichi \nYandex: ' + "{{ ti.xcom_pull(task_ids='yandex_wether',key='wether')[0]['temperature'] }}" + " degrees at " +
30         "{{ ti.xcom_pull(task_ids='yandex_wether',key='wether')[0]['datetime'] }}" +
31         "\nOpen wether: " + "{{ ti.xcom_pull(task_ids='open_wether',key='open_wether')[0]['temperature'] }}" + " degrees at " + "{{ ti.xcom_pull(
32         task_ids='open_wether',key='open_wether')[0]['datetime'] }}",
33     )
34
35     @task(task_id='yandex_wether')
36     def get_yandex_wether(**kwargs):
37         ti = kwargs['ti']
38         url = "https://api.weather.yandex.ru/v2/informers/?lat=53.132363&lon=26.017618"
39
40         payload={}
41         headers = {
42             'X-Yandex-API-Key': '33f45b91-bcd4-46e4-adc2-33cfd8b88e'
43         }
44         response = requests.request("GET", url, headers=headers, data=payload)
45         print("test")
46         temperature = response.json()['fact']['temp']
47         current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
48         a=response.json()['fact']['temp']
49         print(a)
50         ti.xcom_push(key='wether', value={'temperature': temperature, 'datetime': current_datetime})
51
52     @task(task_id='open_wether')
53     def get_open_wether(**kwargs):
54         ti = kwargs['ti']
55         url = "https://api.openweathermap.org/data/2.5/weather?lat=53.132363&lon=26.017618&appid=2cd78e55c423fc81cebc1487134a6300"
56
57         payload={}
58         headers = {}
59
60         response = requests.request("GET", url, headers=headers, data=payload)
```

```

59 print("test")
60 temperature = round(float(response.json()['main']['temp']) - 273.15, 2)
61 current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
62 a=round(float(response.json()['main']['temp']) - 273.15, 2)
63 print(a)
64 ti.xcom_push(key='open_wether', value={'temperature': temperature, 'datetime': current_datetime})
65
66
67 @task(task_id='save_weather')
68 def get_save_weather(**kwargs):
69     yandex_data = kwargs['ti'].xcom_pull(task_ids='yandex_wether', key='wether')
70     open_weather_data = kwargs['ti'].xcom_pull(task_ids='open_wether', key='open_wether')
71
72     temperature_yandex = yandex_data['temperature']
73     datetime_yandex = yandex_data['datetime']
74     service_yandex = 'Yandex'
75
76     temperature_open_weather = open_weather_data['temperature']
77     datetime_open_weather = open_weather_data['datetime']
78     service_open_weather = 'OpenWeather'
79
80     engine = create_engine("mysql://root:1@localhost:33061/spark")
81
82     with engine.connect() as connection:
83         connection.execute("""DROP TABLE IF EXISTS spark.`Temperature_Weather`""")
84         connection.execute("""CREATE TABLE IF NOT EXISTS spark.`Temperature_Weather` (
85             Service VARCHAR(255),
86             Date_time TIMESTAMP,
87             City VARCHAR(255),
88             Temperature FLOAT,
89             PRIMARY KEY (Date_time, Service)
90             )COLLATE='utf8mb4_general_ci' ENGINE=InnoDB""")
91         connection.execute(f"""INSERT INTO spark.`Temperature_Weather` (Date_time, City, Temperature, Service) VALUES ('{datetime_yandex}',
92             'Baranovich', {temperature_yandex}, '{service_yandex}')""")
93         connection.execute(f"""INSERT INTO spark.`Temperature_Weather` (Date_time, City, Temperature, Service) VALUES ('{datetime_open_weather}',
94             'Baranovich', {temperature_open_weather}, '{service_open_weather}')""")
95
96 @task(task_id='python_wether')
97 def get_wether(**kwargs):
98     print("Yandex "+str(kwargs['ti'].xcom_pull(task_ids=['yandex_wether'],key='wether')[0])+ " Open "+str(kwargs['ti'].xcom_pull(task_ids=[
99         'open_wether'],key='open_wether')[0]))
100
101     get_yandex_wether() >> get_open_wether() >> get_wether() >> send_message_telegram_task >> get_save_weather()
102
103 dag = WetherETL()

```


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
16:51 UTC
AA

## DAGs

All 7
Active 7
Paused 0
Running 0
Failed 0

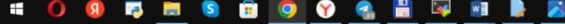

Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
AGanshin003	Valerik	0  1  0 0 8 ***		2024-11-23, 05:00:00	2024-11-24, 05:00:00	1		...
Baranovich_check_temperature_warm_or_cold	airflow	3  1  1 1 day, 0:00:00		2024-11-23, 00:00:00	2024-11-24, 00:00:00	3  1		...
HomeWork6_Task2	Valerik	2  2  0 7 ***		2024-11-23, 04:00:00	2024-11-24, 04:00:00	1		...
HomeWork7_Task1	Valerik	3  3  0 7 ***		2024-11-23, 04:00:00	2024-11-24, 04:00:00	1		...
hello_world	airflow	4  1  0 12 ***		2024-11-23, 12:00:00	2024-11-24, 12:00:00	2  1		...
wether-tiegram-home	airflow	1  0  @once		2024-11-23, 00:00:00		1		...
wether-tiegram-home-sql-exel	airflow	1  1  @once		2024-11-24, 16:21:51		1		...

1

Showing 1-7 of 7 DAGs

Version: v2.10.3  
Git Version: \_release:c99887ec11ce3e1a43f2794fc36d27555140f00


ENG
19:51
24.11.2024

The screenshot shows the Airflow web interface for the DAG 'wether-tlegram-home'. The DAG is in a 'Success' state, having completed its run on 2024-11-24 at 16:56:43 UTC. The task list includes 'yandex\_wether', 'open\_wether', 'python\_wether', 'send\_message\_telegram', and 'save\_weather'. A task instance for 'save\_weather' is highlighted. The DAG code is visible, showing imports for datetime, os, requests, pendulum, airflow.decorators, airflow.providers.telegram.operators.telegram, sqlalchemy, pandas, and tabulate. The code defines a DAG with a single task 'WetherETL()' and a Telegram bot interface. The Telegram bot interface shows a chat with 'Valeria28\_10\_bot' where the bot sends weather information for Moscow and Baranovichi. The bot's messages are: '/start 22:20', 'привет 22:21', 'Wether in Moscow Yandex: -1 degrees Open wether: -0.06 degrees 22:30', and 'Wether in Baranovichi Yandex: -1 degrees at 2024-11-23 20:08:28 Open wether: -2.64 degrees at 2024-11-23 20:08:33 20:08'. The bot's messages are also visible in the Telegram chat interface.

## Задание 2.

В Телеграмм таблица выглядит не отформатированной, но при копировании сообщения, таблица получается ровной и это связано с маленькими окнами сообщений:

```
1 import datetime
2 import os
3 import requests
4 import pendulum
5 import markdown
6 from airflow.decorators import dag, task
7 from airflow.providers.telegram.operators.telegram import TelegramOperator
8 from sqlalchemy import create_engine
9 import pandas as pd
10 from tabulate import tabulate
11
12 os.environ["no_proxy"] = "*"
13
14 @dag(
15     dag_id="wether-tlegram-home-sql-exel",
16     schedule="@once",
17     start_date=pendulum.datetime(2024, 11, 23, tz="UTC"),
18     catchup=False,
19     dagrun_timeout=datetime.timedelta(minutes=60),
20 )
21 def WetherETL():
22
23     first_message_telegram = TelegramOperator(
24         task_id='weather_sql_message_telegram',
25         telegram_conn_id='telegram_default',
26         token='7672923175:AAE48c-RP3irUOP96hEaFuKvIRwGWmdPTaE',
27         chat_id='5174647902',
28         text='''
29         <b>*Weather in Baranovichi*</b>
30
31         |-----|
32         | Source | Degrees |
33         |-----|-----|
34         | 1 | Yandex | {{ '{:5}'.format(ti.xcom_pull(task_ids=["yandex_wether"], key="wether")[0]["temperature"]) }} |
35         | 2 | Open Weather | {{ '{:5}'.format(ti.xcom_pull(task_ids=["open_wether"], key="open_wether")[0]["temperature"]) }} |
36         |-----|-----|
37         '''
38     )
39
40     second_message_telegram = TelegramOperator(
41         task_id='exel_message_telegram',
42         telegram_conn_id='telegram_default',
43         token='7672923175:AAE48c-RP3irUOP96hEaFuKvIRwGWmdPTaE',
44         chat_id='5174647902',
45         text='*Кредитные платежи:*\\n\\n\\n{{ task_instance.xcom_pull(task_ids="data_from_excel") }}\\n\\n\\n',
46     )
47
48 @task(task_id='yandex_wether')
49 def get_yandex_wether(**kwargs):
50     ti = kwargs['ti']
51     url = "https://api.weather.yandex.ru/v2/informers/?lat=53.132363&lon=26.017618"
52
53     payload = {}
54     headers = {
55         'X-Yandex-API-Key': '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
56     }
57     response = requests.get(url, headers=headers, data=payload)
58     data = response.json()
59     temperature = data['temperature']
60     return temperature
```



```

61 response = requests.request("GET", url, headers=headers, data=payload)
62 print("test")
63 temperature = response.json()['fact']['temp']
64 current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
65 a=response.json()['fact']['temp']
66 print(a)
67 ti.xcom_push(key='wether', value={'temperature': temperature, 'datetime': current_datetime})
68
69
70 # return str(a)
71
72 @task(task_id='open_wether')
73 def get_open_wether(**kwargs):
74     ti = kwargs['ti']
75     url = "https://api.openweathermap.org/data/2.5/weather?lat=53.132363&lon=26.017618&appid
76     =2cd78e55c423fc81cebc1487134a6300"
77     payload={}
78     headers = {}
79
80     response = requests.request("GET", url, headers=headers, data=payload)
81     print("test")
82     temperature = round(float(response.json()['main']['temp']) - 273.15, 2)
83     current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
84     a=round(float(response.json()['main']['temp']) - 273.15, 2)
85     print(a)
86     ti.xcom_push(key='open_wether', value={'temperature': temperature, 'datetime': current_datetime})
87
88 #
89     return str(a)
90
91 @task(task_id='save_weather')
92 def get_save_weather(**kwargs):
93     ti = kwargs['ti']
94     yandex_data = kwargs['ti'].xcom_pull(task_ids='yandex_wether', key='wether')
95     open_weather_data = kwargs['ti'].xcom_pull(task_ids='open_wether', key='open_wether')
96
97     temperature_yandex = yandex_data['temperature']
98     datetime_yandex = yandex_data['datetime']
99     service_yandex = 'Yandex'
100     temperature_open_weather = open_weather_data['temperature']
101     datetime_open_weather = open_weather_data['datetime']
102     service_open_weather = 'OpenWeather'
103
104     engine = create_engine("mysql://root:1@localhost:33061/spark")
105
106     with engine.connect() as connection:
107         connection.execute("""DROP TABLE IF EXISTS spark.`Temperature_Weather`""")
108         connection.execute("""CREATE TABLE IF NOT EXISTS spark.`Temperature_Weather` (
109             Service VARCHAR(255),
110             Date_time TIMESTAMP,
111             City VARCHAR(255),
112             Temperature FLOAT,
113             PRIMARY KEY (Date_time, Service)
114             )COLLATE='utf8mb4_general_ci' ENGINE=InnoDB""")
115         connection.execute(f"""INSERT INTO spark.`Temperature_Weather` (Date_time, City, Temperature, Service) VALUES ('
116 {datetime_yandex}', 'Baranovichi', {temperature_yandex}, '{service_yandex}')""")
117         connection.execute(f"""INSERT INTO spark.`Temperature_Weather` (Date_time, City, Temperature, Service) VALUES ('
118 {datetime_open_weather}', 'Baranovichi', {temperature_open_weather}, '{service_open_weather}')""")

```

```

117
118
119 @task(task_id='python_wether')
120 def get_wether(**kwargs):
121     print("Yandex "+str(kwargs['ti'].xcom_pull(task_ids=['yandex_wether'],key='wether')[0])+ " Open "+str(kwargs['ti'].
122     .xcom_pull(task_ids=['open_wether'],key='open_wether')[0]))
123
124     def truncate_message(message, max_length=3959):
125         if len(message) > max_length:
126             return message[:max_length]
127         return message
128
129     @task(task_id='data_from_excel')
130     def get_data_from_excel():
131         excel_file_path = "/home/lera/s4.2.xlsx"
132         df = pd.read_excel(excel_file_path)
133         table_text = df.to_markdown(index=False)
134         truncated_text = truncate_message(table_text)
135         return truncated_text
136
137     get_yandex_wether() >> get_open_wether() >> get_wether() >> get_save_weather() >> first_message_telegram >>
138     get_data_from_excel() >> second_message_telegram
139
140 dag = WetherETL()

```

