

PRÁCTICA 4

ALMACÉNS E MINARÍA DE DATOS

Álex Baquero Domínguez
alex.baquero@rai.usc.es

Hugo Gómez Sabucedo
hugo.gomez.sabucedo@rai.usc.es

Alejandro Aybar Cifuentes
alejandro.aybar@rai.usc.es

Tabla de contenidos

1. Importación de los archivos .csv mezclados en una base de datos PostgreSQL usando Apache Airflow.	3
2. Ejercicio anterior utilizando Talend.	14
3. Escenario y reto que supone la copia incremental de datos.	17
4. Ejecución del ejecutable de Talend desde Airflow.	19

1. Importación de los archivos .csv mezclados en una base de datos PostgreSQL usando Apache Airflow.

1. Ejecución manual

1.1. Recibir ficheros CSV y mezclarlos

En primer lugar, creamos un programa Python donde recibimos los ficheros CSV, los mezclamos y guardamos la mezcla en un nuevo CSV:

```
cases_per_country.py x deaths_per_country.csv x cases_per_country.csv x
1 import pandas as pd
2 import psycopg2
3
4 def read_csv_cases():
5     return pd.read_csv("~/Documentos/AMD/p4/cases_per_country.csv",
6                        sep=';', encoding='latin-1')
7
8 def read_csv_deaths():
9     return pd.read_csv("~/Documentos/AMD/p4/deaths_per_country.csv",
10                        sep=";", encoding='latin-1')
11
12 def merge_data(data_cases, data_deaths):
13     return pd.merge(data_cases, data_deaths, on='country', how='inner')
14
```

```
72 data_cases = read_csv_cases()
73 data_deaths = read_csv_deaths()
74 merged_data = merge_data(data_cases, data_deaths)
75 merged_data.to_csv('~/.Documentos/AMD/p4/daily_covid19.csv', sep=';', index=False)
76
```

Comprobamos que los ficheros se han leído correctamente, y la mezcla se ha realizado con éxito:

```
alejandro@alejandro-ubuntu:~/Documentos/AMD/p4$ python cases_per_country.py
alejandro@alejandro-ubuntu:~/Documentos/AMD/p4$ cat ./daily_covid19.csv
country;cases;deaths
United States Of America;76841644;983637
India;42269212;518389
Brazil;31171102;665976
France;29910679;162775
Germany;26854388;139910
United Kingdom;18893211;175702
South Korea;18132312;23093
Russia;17927694;363694
Italy;17758795;168738
Turkey;14254816;95518
Spain;12551089;107272
Vietnam;10357897;38118
Japan;8996557;30859
Argentina;8307556;128380
Netherlands;8106145;22336
```

Consulta del fichero CSV de la mezcla tras ejecutar nuestro programa Python.

```

alejandro@alejandro-ubuntu:~/Documentos/AMD/p4$ cat ./cases_per_country.csv
country;cases
United States Of America;76841644
India;42269212
Brazil;31171102
France;29910679
Germany;26854388
United Kingdom;18893211
South Korea;18132312
Russia;17927694
Italy;17758795
Turkey;14254816
Spain;12551089
Vietnam;10357897
Japan;8996557
Argentina;8307556
Netherlands;8106145
Iran;7196330

```

Contenido del fichero CSV de casos recibido

```

alejandro@alejandro-ubuntu:~/Documentos/AMD/p4$ cat deaths_per_country.csv
country;deaths
United States Of America;983637
Brazil;665976
India;518389
Russia;363694
Mexico;322776
Peru;212536
United Kingdom;175702
Italy;168738
France;162775
Indonesia;156525
Iran;140518
Germany;139910
Colombia;138986
Argentina;128380
Poland;116744
Spain;107272
Ukraine;104542

```

Contenido del fichero CSV de muertes recibido

Nota: para leer los ficheros y mezclarlos hemos hecho uso del paquete 'pandas'.

1.2. Conexión a PostgreSQL

El siguiente paso es conectarnos al servidor de Postgres, y guardar ahí la mezcla. Implementamos esto al programa Python:

1.2.1. Crear tabla en PostgreSQL

En primer lugar, ejecutamos el comando CREATE TABLE para crear la tabla 'daily_covid19', que tendrá los atributos: (country, cases, deaths). Incluiremos la cláusula IF NOT EXISTS, para evitar errores cuando la tabla ya esté creada.

El procedimiento consiste en crear el comando, y pasárselo al cursor de la conexión PostgreSQL, para que ejecute dicho comando.

```

16 def command_create_table():
17     return """
18         CREATE TABLE IF NOT EXISTS daily_covid19(
19             country VARCHAR(50) PRIMARY KEY,
20             cases INT,
21             deaths INT
22         )
23     """
24
25 def execute_create_postgres(create_table):
26
27     connection = None
28
29     try:
30         connection = psycopg2.connect(host="localhost", database="covid19",
31                                     user="postgres", password="dogui2013")
32         cursor = connection.cursor()
33
34         cursor.execute(create_table)
35
36         cursor.close()
37         connection.commit()
38
39     except (Exception, psycopg2.DatabaseError) as error:
40         print(error)
41
42     finally:
43         if connection is not None:
44             connection.close()

```

```

81 create_table = command_create_table()
82 execute_create_postgres(create_table)

```

1.2.2. Insertar datos en PostgreSQL

A continuación, con la tabla ya creada, ejecutamos el comando COPY FROM, con el que insertamos la información del CSV:

```

46 def execute_insert_postgres():
47
48     connection = None
49
50     try:
51         connection = psycopg2.connect(host="localhost", database="covid19",
52                                     user="postgres", password="dogui2013")
53         cursor = connection.cursor()
54
55         cursor.execute('DELETE FROM daily_covid19')
56
57         with open('./daily_covid19.csv', 'r') as csv_file:
58             next(csv_file)
59             cursor.copy_from(csv_file, 'daily_covid19', sep=';')
60
61         cursor.close()
62         connection.commit()
63
64     except (Exception, psycopg2.DatabaseError) as error:
65         print(error)
66
67     finally:
68         if connection is not None:
69             connection.close()
70

```

1.3. Comprobación en PostgreSQL

Ejecutamos nuestro programa Python, y acudimos a PostgreSQL para comprobar si la información se ha almacenado correctamente:



The screenshot shows a PostgreSQL query interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:

```
1 SELECT * FROM public.daily_covid19
2 ORDER BY country ASC
```

Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table of results. The table has four columns: 'country' (character varying (50)), 'cases' (integer), and 'deaths' (integer). The results are ordered by country in ascending order. The table shows 26 rows of data, with the first row being Afghanistan and the last row being Bosnia And Herzegovina. At the bottom of the table, it says 'Total rows: 221 of 221' and 'Query complete 00:00:00.689'.

	country [PK] character varying (50)	cases integer	deaths integer
1	Afghanistan	180523	7672
2	Albania	262812	3423
3	Algeria	259430	6737
4	American Samoa	6190	31
5	Andorra	36298	149
6	Angola	76454	1819
7	Anguilla	2990	7
8	Antigua And Barbuda	7670	137
9	Argentina	8307556	128380
10	Armenia	420733	8524
11	Aruba	28745	217
12	Australia	6895936	8736
13	Austria	4318626	16245
14	Azerbaijan	781625	9457
15	Bahamas	29799	808
16	Bahrain	579820	1486
17	Bangladesh	1941623	29077
18	Barbados	77315	458
19	Belarus	959398	6653
20	Belgium	4170786	31301
21	Belize	55459	662
22	Benin	26335	163
23	Bermuda	13459	134
24	Bhutan	59518	21
25	Bolivia	804523	21433
26	Bosnia And Herzegovina	362753	15378

Total rows: 221 of 221 Query complete 00:00:00.689

Como podemos comprobar, la información de la mezcla ha sido almacenada en el servidor PostgreSQL con éxito.

Habiendo logrado la tarea de manera manual, vamos a proceder a automatizar el proceso con Apache Airflow.

2. Ejecución automatizada con Apache Airflow

Para trabajar sobre Airflow, vamos a abrir dos terminales. En una ejecutamos 'airflow scheduler', para actualizar los DAGs según los creamos/modifiquemos. En otra, ejecutamos 'airflow webserver', para disponer de la interfaz web en 'localhost:8080'.

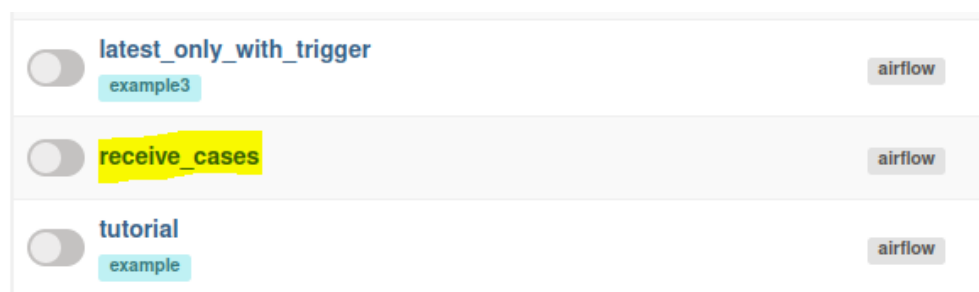
2.1. Recibir CSV de casos

En primer lugar, vamos a automatizar la tarea de recibir el CSV que contiene los casos semanales por país. A modo de prueba, vamos a configurar el 'schedule_interval' de nuestro DAG para que se ejecute cada minuto. Así, podremos comprobar si el DAG funciona correctamente, antes de configurarlo para las 1 a.m.

Situamos este ejecutable en la carpeta airflow/dags.

```
receive_cases.py x cases_per_country.py x cases_covid19.csv x deaths_per_country.csv x
1 from datetime import datetime
2 from airflow import DAG
3 from airflow.operators.dummy_operator import DummyOperator
4 from airflow.operators.python_operator import PythonOperator
5 from pandas import pandas as pd
6
7 def receive_cases():
8     cases_data = pd.read_csv("~/Documentos/AMD/p4/cases_per_country.csv",
9                             sep=';', encoding='latin-1')
10    cases_data.to_csv('~/.airflow/data/cases_covid19.csv', sep=';', index=False)
11
12    dag = DAG('receive_cases',
13             description='Receive cases CSV DAG',
14             schedule_interval='* * * * *',
15             start_date=datetime(year=2022, month=11, day=5),
16             catchup=False)
17
18    receive_cases_operator = PythonOperator(task_id='receive_task',
19                                           python_callable=receive_cases, dag=dag)
20
21    receive_cases_operator
22
```

Acudimos a la interfaz de Airflow, y buscamos el DAG que acabamos de crear.

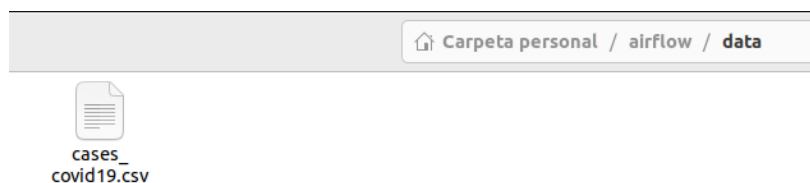


Lo activamos y esperamos unos minutos para ver si la tarea automatizada se realiza correctamente.

DAG receive_cases	
DAG Details	
DAG Runs Summary	
Total Runs Displayed	25
■ Total success	25
First Run Start	2022-11-06, 16:32:01 UTC
Last Run Start	2022-11-06, 16:56:01 UTC
Max Run Duration	00:00:17
Mean Run Duration	00:00:03
Min Run Duration	00:00:01
DAG Summary	
Total Tasks	1
PythonOperator	1

Transcurridos 25 minutos, observamos cómo el DAG se ha ejecutado exitosamente 25 veces.

Acudimos a la carpeta airflow/data, que establecimos como destino del fichero CSV recibido.



Como podemos observar, se ha recibido correctamente el fichero CSV. Lo abrimos para ver su contenido:

```
~/airflow/data/cases_covid19.csv - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
cases_covid19.csv x receive_cases.py x cases_per_country.py x deaths_per_country.csv x
1 country;cases
2 United States Of America;76841644
3 India;42269212
4 Brazil;31171102
5 France;29910679
6 Germany;26854388
7 United Kingdom;18893211
8 South Korea;18132312
9 Russia;17927694
10 Italy;17758795
11 Turkey;14254816
12 Spain;12551089
13 Vietnam;10357897
14 Japan;8996557
15 Argentina;8307556
16 Netherlands;8106145
17 Iran;7196330
18 Australia;6895936
19 Indonesia;6055428
20 Poland;6018262
```


2.2. Recibir CSV de muertes

Hacemos el mismo proceso para el CSV de muertes:

```
receive_deaths.py  x  cases_covid19.csv  x  receive_cases.py  x  cases_per_country.py  x
1  from datetime import datetime
2  from airflow import DAG
3  from airflow.operators.dummy_operator import DummyOperator
4  from airflow.operators.python_operator import PythonOperator
5  from pandas import pandas as pd
6
7  def receive_deaths():
8      cases_data = pd.read_csv("~/Documentos/AMD/p4/deaths_per_country.csv",
9                             sep=';', encoding='latin-1')
10     cases_data.to_csv('~/.airflow/data/deaths_covid19.csv', sep=';', index=False)
11
12     dag = DAG('receive_deaths',
13              description='Receive deaths CSV DAG',
14              schedule_interval='* * * * *',
15              start_date=datetime(year=2022, month=11, day=5),
16              catchup=False)
17
18     receive_deaths_operator = PythonOperator(task_id='receive_deaths_task',
19                                             python_callable=receive_deaths, dag=dag)
20
21     receive_deaths_operator
22
```

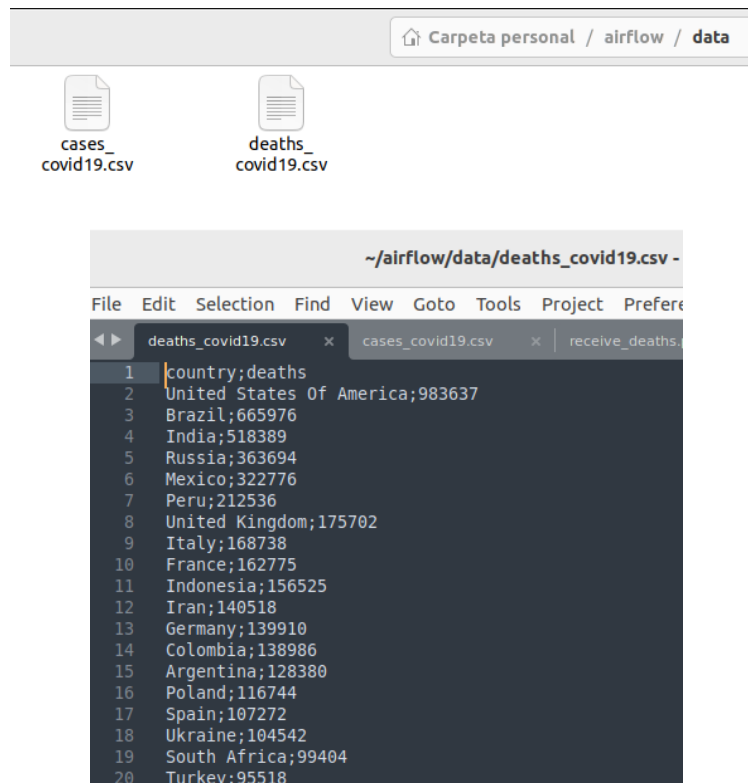
Buscamos el DAG creado:



Lo activamos y comprobamos si se ha ejecutado correctamente:

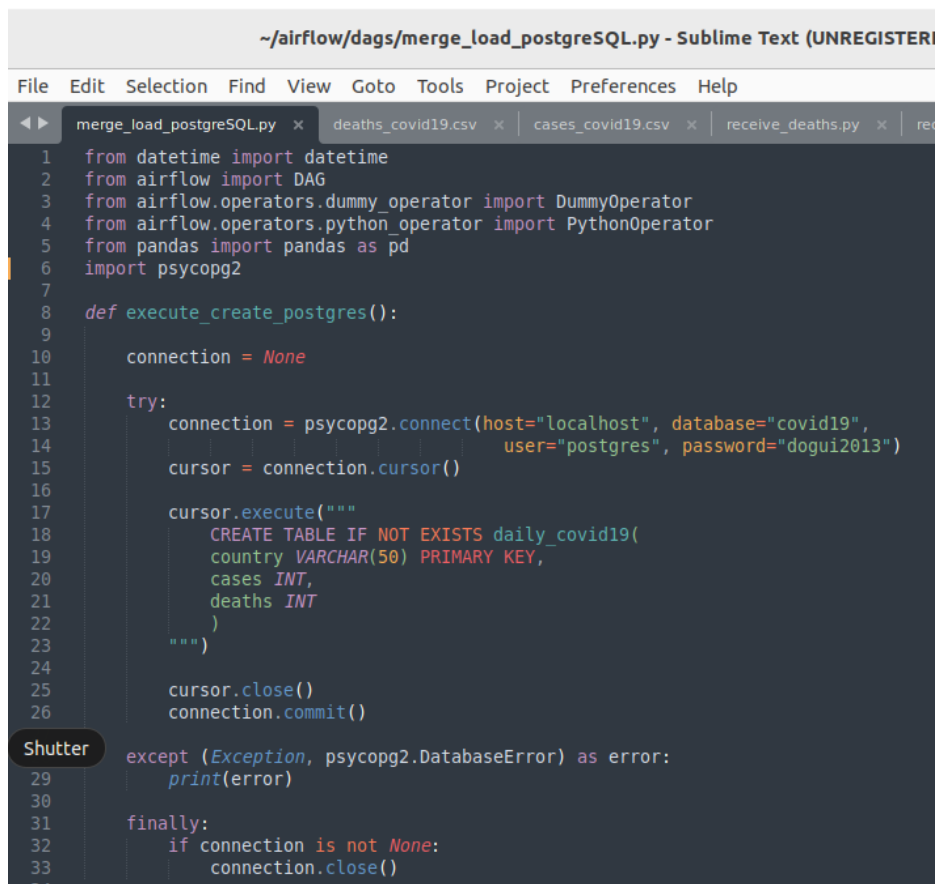
DAG receive_deaths	
DAG Details	
DAG Runs Summary	
Total Runs Displayed	2
■ Total success	2
First Run Start	2022-11-06, 17:20:44 UTC
Last Run Start	2022-11-06, 17:21:01 UTC
Max Run Duration	00:00:04
Mean Run Duration	00:00:04
Min Run Duration	00:00:03
DAG Summary	
Total Tasks	1
PythonOperator	1

Buscamos el archivo CSV recibido en airflow/data y consultamos su contenido:



2.3. Mezclar y guardar en PostgreSQL

El siguiente DAG se encargará de mezclar el contenido de los 2 ficheros CSV recibidos (casos y muertes), y guardar el resultado en PostgreSQL:



```

34
35 def execute_insert_postgres():
36
37     data_cases = pd.read_csv("~/airflow/data/cases_covid19.csv", sep=';', encoding='latin-1')
38     data_deaths = pd.read_csv("~/airflow/data/deaths_covid19.csv", sep=';', encoding='latin-1')
39     data_merge = pd.merge(data_cases, data_deaths, on='country', how='inner')
40     data_merge.to_csv("~/airflow/data/daily_covid19.csv", sep=';', index=False)
41
42
43     connection = None
44
45     try:
46         connection = psycopg2.connect(host="localhost", database="covid19",
47                                     user="postgres", password="dogui2013")
48         cursor = connection.cursor()
49
50         cursor.execute('DELETE FROM daily_covid19')
51
52         with open("~/airflow/data/daily_covid19.csv", 'r') as csv_file:
53             next(csv_file)
54             cursor.copy_from(csv_file, 'daily_covid19', sep=';')
55
56         cursor.close()
57         connection.commit()
58
59     except (Exception, psycopg2.DatabaseError) as error:
60         print(error)
61
62     finally:
63         if connection is not None:
64             connection.close()
65
66
67 dag = DAG('merge_load_postgreSQL',
68         description='Merge cases and deaths and load to PostgreSQL',
69         schedule_interval='* * * * *',
70         start_date=datetime(year=2022, month=11, day=5),
71         catchup=False)
72
73 create_table_operator = PythonOperator(task_id = 'create_table_task',
74 python_callable = execute_create_postgres, dag = dag)
75
76 insert_data_operator = PythonOperator(task_id = 'insert_data_task',
77 python_callable = execute_insert_postgres, dag = dag)
78
79 create_table_operator >> insert_data_operator
80

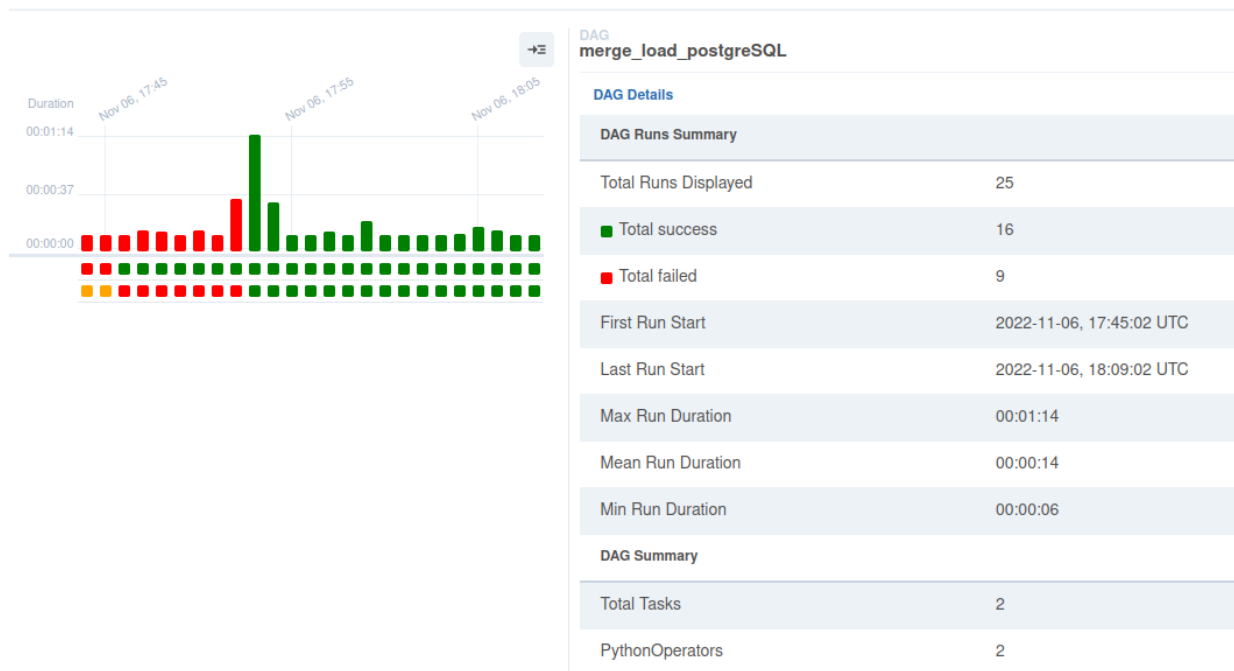
```

Como podemos observar, hemos creado dos operadores:

- El primero se encarga de crear la tabla en PostgreSQL, con la cláusula IF NOT EXISTS para evitar errores cuando la tabla ya existe. El segundo mezcla.
- El segundo mezcla el contenido de los dos ficheros CSV, almacena el resultado y lo guarda en PostgreSQL

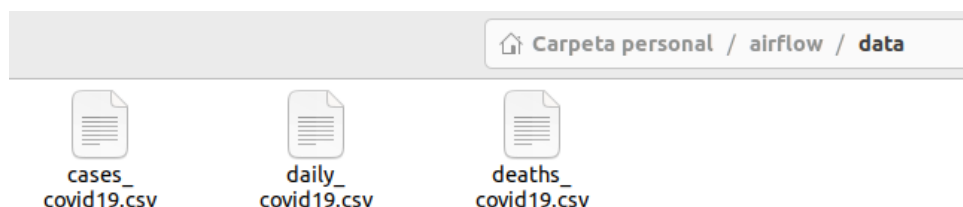
Buscamos el DAG en la interfaz, y lo activamos:





Tras solucionar problemas de compilación al inicio, conseguimos que el DAG cumpla su función y se ejecute correctamente.

Buscamos el fichero CSV con la mezcla en airflow/data y consultamos su contenido:



```
~/airflow/data/daily_covid19.csv -
le Edit Selection Find View Goto Tools Project Prefe
daily_covid19.csv x merge_load_postgreSQL.py x deaths_covid
1 country;cases;deaths
2 United States Of America;76841644;983637
3 India;42269212;518389
4 Brazil;31171102;665976
5 France;29910679;162775
6 Germany;26854388;139910
7 United Kingdom;18893211;175702
8 South Korea;18132312;23093
9 Russia;17927694;363694
10 Italy;17758795;168738
11 Turkey;14254816;95518
12 Spain;12551089;107272
13 Vietnam;10357897;38118
14 Japan;8996557;30859
15 Argentina;8307556;128380
16 Netherlands;8106145;22336
17 Iran;7196330;140518
18 Australia;6895936;8736
19 Indonesia;6055428;156525
20 Poland;6018262;116744
```

Por último, acudimos a PostgreSQL, y comprobamos si la información se ha almacenado exitosamente.

Query

Query History









1 SELECT * FROM public.daily_covid19

2 ORDER BY country ASC

Data Output

Messages

Notifications



	country [PK] character varying (50)	cases integer	deaths integer
1	Afghanistan	180523	7672
2	Albania	262812	3423
3	Algeria	259430	6737
4	American Samoa	6190	31
5	Andorra	36298	149
6	Angola	76454	1819
7	Anguilla	2990	7
8	Antigua And Barbuda	7670	137
9	Argentina	8307556	128380
10	Armenia	420733	8524

2.4. Automatización con la hora correcta

Por último, vamos a automatizar la tarea con las horas requeridas en el ejercicio. Es decir, marcaremos el DAG de recibir casos para la 1 a.m., y el de recibir muertes al de 2 a.m., junto con el de mezclar y guardar en PostgreSQL.

Para ello, tenemos que ajustar la sentencia cron de los DAGs:

- receive_cases: 0 1 * * *
- receive_deaths: 0 2 * * *
- merge_load: 5 2 * * *

<input type="checkbox"/> merge_load_postgreSQL	airflow	<div><div>49</div><div>11</div></div>	5 2 * * *
<input type="checkbox"/> receive_cases	airflow	<div><div>85</div><div></div></div>	0 1 * * *
<input type="checkbox"/> receive_deaths	airflow	<div><div>36</div><div></div></div>	0 2 * * *

Con esto, nuestras tareas quedan programadas a la hora indicada.

2. Ejercicio anterior utilizando Talend.

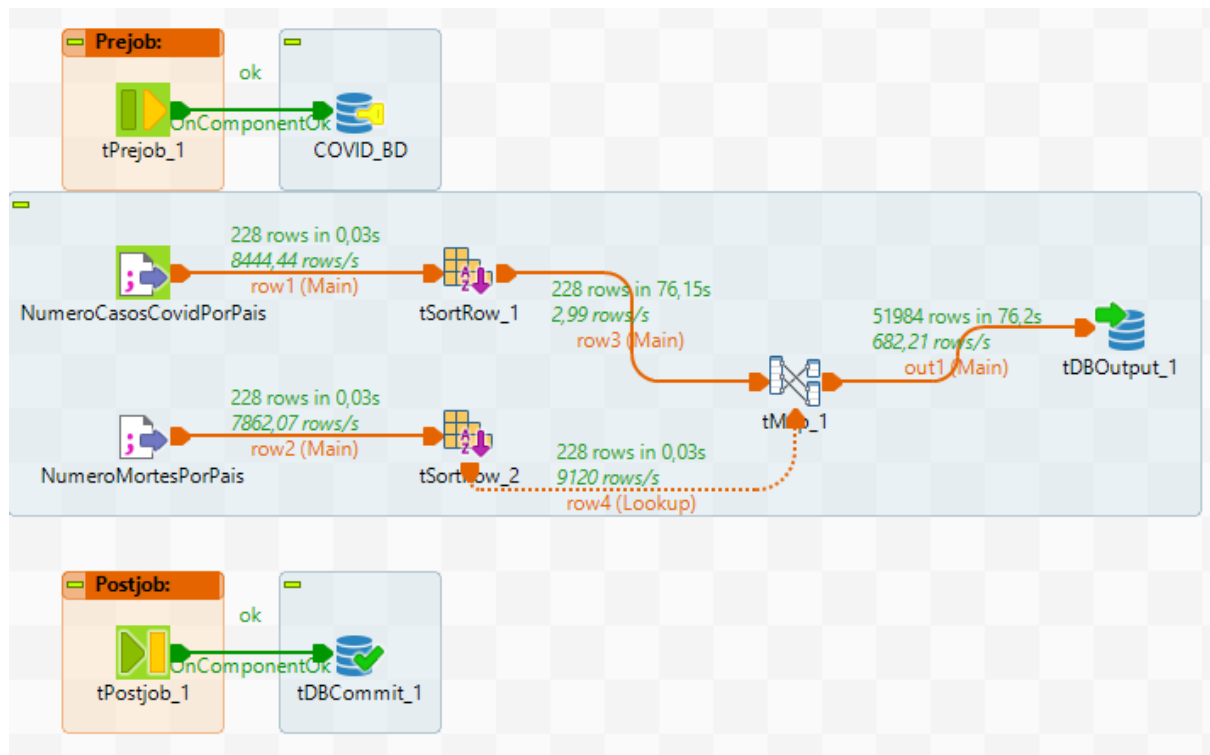
Partimos de dos archivos .csv en los que uno especifica el número de casos COVID por país y otro el número de fallecidos por país:

NumeroMortesPorPais.csv: caderno de notas	NumeroCasosCovidPorPais.csv: caderno...
Ficheiro Editar Formato Ver Axuda	Ficheiro Editar Formato Ver Axuda
Indonesia;156557	Indonesia;6058455
Bangladesh;29117	Bangladesh;1951303
Venezuela;5686	Venezuela;521907
Brunei Darussalam;164	Brunei Darussalam;153008
Kiribati;13	Kiribati;3204
Cameroon;1915	Cameroon;118242
Luxembourg;1277	Luxembourg;278235
Sweden;19049	Sweden;2511512
Uganda;3570	Uganda;148446
Montenegro;2681	Montenegro;225735
Jordan;13626	Jordan;1671943
Dominican Republic;4354	Dominican Republic;579652
Ireland;6290	Ireland;1549848
Cambodia;3048	Cambodia;136168
United States Of America;991649	United States Of America;81248793
Singapore;1381	Singapore;1316766
Papua New Guinea;657	Papua New Guinea;44476
Sri Lanka;16252	Sri Lanka;655710
San Marino;109	San Marino;15797
Laos;677	Laos;196961
American Samoa;31	American Samoa;6191
Uzbekistan;1613	Uzbekistan;237358
Bosnia And Herzegovina;15519	Bosnia And Herzegovina;370932
Portugal;23664	Portugal;4962131
Finland;4714	Finland;1114573
< Liña 1	< Liña 1, Columna 100% Windows (CRLF) UTF-8

Antes de empezar con Talend, creamos la tabla donde almacenaremos los datos de los .csv mezclados en PostgreSQL:

```
CREATE TABLE covid_p4(  
    country varchar(40),  
    COVIDCases int,  
    deaths int  
);
```

Para realizar este ejercicio en el Talend, lo primero que tenemos que hacer es conectarnos a la base de datos donde creamos nuestra tabla anterior utilizando un tDBConnection.

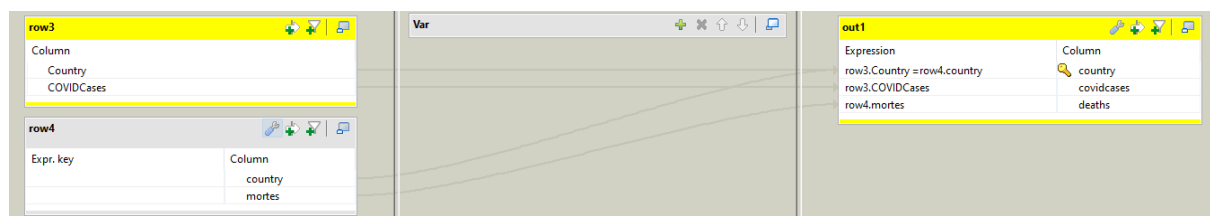


Creamos dos `tFileInputDelimited` que hacen referencia a ambos .csv. Estos archivos se pasan por unos `tSortRows` para asegurarse de que la columna *country* entra en el `tMap` ordenada en ambos archivos. De esta manera, el mapeo será más fácil.

Criteria			
Schema column	sort num or alpha?	Order asc or desc?	
Country	alpha	asc	

En el `tMap`, juntamos las columnas de ambos archivos haciendo que la columna *country* del primer archivo .csv sea igual a la columna *country* del segundo archivo .csv:

```
row3.Country = row4.country
```



Almacenamos el resultado en un `tDBOutput` especificando que sea en la tabla creada anteriormente (si no está creada, hacemos que se cree).

tDBOutput_1(PostgreSQL)

Database: PostgreSQL Apply

☒ Use an existing connection

Component List: tDBConnection_1 - COVID_BD *

Table: "covid_p4" *

Action on table: Create table if does not exist

Action on data: Insert or update

Schema: Built-In Edit schema Sync columns

☐ Use spatial options

☐ Die on error

Finalmente, hacemos un commit en la base de datos y los datos se almacenan en la base de datos de PostgreSQL. El resultado obtenido es el siguiente:

COVID/postgres@PostgreSQL 14

Query Editor Query History

```
1 select * from covid_p4
```

Data Output Explain Messages Notifications

	country character varying (40)	covidcases integer	deaths integer
1	Afghanistan	232073	7684
2	Africa (total)	232073	251517
3	Albania	232073	3443
4	Algeria	232073	6801
5	America (total)	232073	2733254
6	American Samoa	232073	31
7	Andorra	232073	150
8	Angola	232073	1866
9	Anguilla	232073	8
10	Antigua And Barbuda	232073	139
11	Argentina	232073	128679
12	Armenia	232073	8545
13	Aruba	232073	218
14	Asia (total)	232073	1292728

ol=true&sgid=1&sid=2&serv...

3. Escenario y reto que supone la copia incremental de datos.

En ambos casos comentados anteriormente, tenemos un escenario donde, partiendo de una base de datos con una cierta cantidad de información sobre los casos y muertes por COVID por cada país, recibimos cada día dos archivos con los datos de ese día. Para importar estos datos a la base de datos, podríamos hacerlo de dos formas: una de ellas sería que, cada día, según nos llegasen los datos, borrásemos todos los datos previos en la base de datos, y los cargásemos de nuevo, junto con los nuevos datos. Esto es lo que se conoce como copia completa. Sin embargo, en nuestro caso, la mejor opción es hacer una copia incremental de los datos. De esta forma, tendríamos en la base de datos toda la información previa sobre los casos y muertes por covid y, cada día, según nos llegan los datos, cargamos estos nuevos datos a la base de datos.

La principal ventaja que presenta la copia incremental de datos es que es mucho más rápida que realizar una copia completa, puesto que la cantidad de datos que tendremos es significativamente menor (aproximadamente, tenemos un total de 230 países, y para cada uno de ellos sólo tenemos que guardar, además de la fecha, dos valores: casos y muertes).

Además, ya que siempre importamos una cantidad similar de datos, este proceso durará lo mismo todos los días, por lo que es más sencillo establecer una ventana de tiempo para realizar la importación de datos de forma segura y sin causar ningún problema.

Por otra parte, al tener que importar menos datos, también se reduce la posibilidad de que un fallo afecte a la copia de los mismos, con la posibilidad de pérdida de datos que esto conlleva.

Por el contrario, la copia incremental de datos presenta también algunas desventajas y retos. En primer lugar, puede ser necesario almacenar información adicional a la que tienen los datos, para poder saber cuándo fue importado cada dato, y poder tener un histórico con el cual realizar o bien consultas o bien una restauración de los datos, en caso de que estos se perdieran.

Por otra parte, podría darse la situación de que, en un día concreto, los datos vinieran incompletos, o que su formato fuera incorrecto. Si esto no se detecta a tiempo, podría causar problemas en el proceso de importación de los datos, así como inconsistencias en la base de datos. Este es un riesgo que tenemos que tener en cuenta y minimizar.

También es importante asegurarnos de que los datos se reciben siempre en orden. En nuestro caso, a priori, no tendremos muchos problemas, puesto que sabemos

que recibimos los datos de una fuente a una hora determinada, y los de la otra una hora después. Además, queremos introducir los datos ya combinados, por lo que sólo sería cuestión de esperar hasta recibir todos los datos, e importarlos. Sin embargo, en otros casos, esto podría ser un reto.

Si que es importante analizar y plantearnos qué ocurriría si, por cualquier motivo, no recibimos los datos de una de las fuentes. De ocurrir esto, tendríamos que decidir si, bien importamos los datos de la otra fuente y, cuando recibamos los que faltan, los importamos; o bien si esperamos a tener todos los datos para importarlos. Esto último sería lo más lógico pero, ¿qué ocurriría si no llegamos a recibir algunos de los datos? ¿Deberíamos importar los del día siguiente de cualquier forma, o seguiríamos esperando hasta que se reciban los primeros datos, y luego importamos los datos restantes?

Tenemos que tener en cuenta, además, que se podrían producir cambios en la estructura de los datos, de forma que tuviésemos que almacenar nuevas columnas con nueva información, o cambiar el formato de datos ya existentes. Si esto ocurriese, podríamos tener, de nuevo, los datos en un estado inconsistente, o mismo errores que pudieran ser fatales para la base de datos. Aparte, si esto ocurriese y no lo detectásemos a tiempo, tendríamos que analizar la base de datos en busca del día en que dicha estructura cambió, y cambiar la estructura de los datos anteriores, así como la de los nuevos datos.

Por último, cabe destacar la gran cantidad de espacio que ocupará nuestra base de datos. Si suponemos que tenemos cuatro columnas: una para la fecha, un varchar de tamaño 40 para el nombre del país, y dos ints para los casos y muertes, cada registro ocupará 53 bytes (3 para la fecha, suponiendo que es de tipo date, 2+40 para el varchar y 4 para cada int). Teniendo en cuenta que tenemos datos para 230 países, cada día importaremos, aproximadamente, 12.2kB de datos. Es necesario disponer de espacio suficiente para almacenar tanto todos estos datos, como más información que se quisiera añadir a posteriori, y la información necesaria para encontrar esos datos en la base de datos.

4. Ejecución del ejecutable de Talend desde Airflow.

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime
import os
import sys

bib_app = "/home/P4/job.sh"
default_args = {
    'owner': 'hugo',
    'depends_on_past': False,
    'start_date': datetime(2022, 11, 6),
    'provide_context': True}

dag = DAG('run_talend',
          default_args=default_args,
          description='Dag for Talend job')

t1 = BashOperator(
    task_id='dependency',
    bash_command= bib_app,
    dag=dag)

t2 = BashOperator(
    task_id = 't2',
    dag = dag,
    bash_command = 'bash /home/P4/job.sh'
)

t1.set_upstream(t2)
```

Para ejecutar el job de Talend desde Airflow, crearemos un archivo .py que guardaremos en el directorio AIRFLOW_HOME/dags (si no existe dicho directorio, lo crearemos). El contenido de dicho archivo será el que se ve en la imagen superior. Sólo tendríamos que ejecutar el DAG desde Airflowy, de esta forma, podríamos ejecutar el job de Talend desde Airflow.