Examen Final

Alumno: Federico Bonino
Fecha: 24 de Junio de 2024

Ejercicio 2 – Car Rental

Consigna

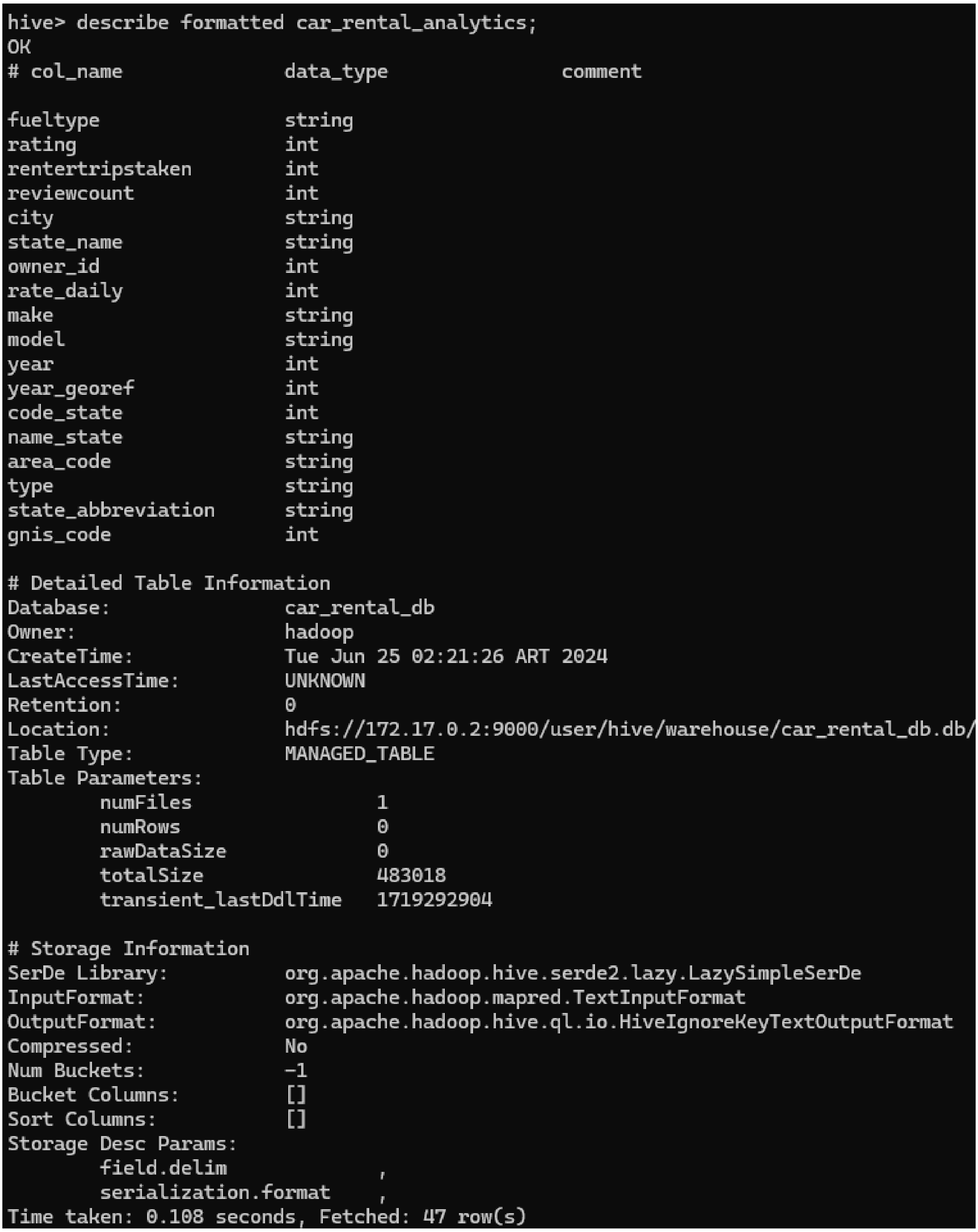
Una de las empresas líderes en alquileres de automóviles solicita una serie de dashboards y reportes para poder basar sus decisiones en datos. Entre los indicadores mencionados se encuentran total de alquileres, segmentación por tipo de combustible, lugar, marca y modelo de
automóvil, valoración de cada alquiler, etc.
Como Data Engineer debe crear y automatizar el pipeline para tener como resultado los datos listos para ser visualizados y responder las preguntas de negocio.

Desarrollo

1. Para crear las tablas en el datawarehouse se utiliza el siguiente script ubicado en `/home/hadoop/scripts/create_database_rental.hql`
   El script se corrió manualmente por consola antes de realizar la ingesta mediante `hive -f /home/hadoop/scripts/create_database_rental.hql`

```
-- Create the database
create database car_rental_db;
use car_rental_db;

-- create tables with appropiate schema
create table
    car_rental_analytics (
        fuelType string,
        rating int,
        renterTripsTaken int,
        reviewCount int,
        city string,
        state_name string,
        owner_id int,
        rate_daily int,
        make string,
        model string,
        year int,
        -- Geo_Point string,
        -- Geo_Shape string,
        year_georef string,
        Code_State int,
        Name_State string,
        Area_Code string,
        Type string,
        state_abbreviation string,
        GNIS_Code int
        )
    row format delimited
    fields terminated by ',';
```

Chequeamos que el esquema de las tablas sea el correcto, con sus nombres y tipos.



```
hive> describe formatted car_rental_analytics;
OK
# col_name              data_type               comment

fueltype                string
rating                  int
rentertripstaken        int
reviewcount             int
city                    string
state_name              string
owner_id                int
rate_daily              int
make                    string
model                   string
year                    int
year_georef             int
code_state              int
name_state              string
area_code               string
type                    string
state_abbreviation      string
gnis_code               int

# Detailed Table Information
Database:               car_rental_db
Owner:                  hadoop
CreateTime:             Tue Jun 25 02:21:26 ART 2024
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://172.17.0.2:9000/user/hive/warehouse/car_rental_db.db/
Table Type:             MANAGED_TABLE
Table Parameters:
        numFiles                1
        numRows                 0
        rawDataSize             0
        totalSize               483018
        transient_lastDdlTime   1719292904

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        serialization.format    ,
Time taken: 0.108 seconds, Fetched: 47 row(s)
```

2. Para realizar la ingesta se utiliza el siguiente script ubicado en `/home/hadoop/scripts/ingest_rental.sh`

```
rm -f /home/hadoop/landing/*.*

wget -O /home/hadoop/landing/CarRentalData.csv
"https://edvaibucket.blob.core.windows.net/data-engineer-edvai/CarRental
Data.csv?sp=r&st=2023-11-06T12:52:39Z&se=2025-11-06T20:52:39Z&sv=2022-
11-02&sr=c&sig=J4Ddi2c7Ep23OhQLPisbYaerlH472iigPwc1%2FkG80EM%3D"

wget -O /home/hadoop/landing/georef.csv
"https://dataengineerpublic.blob.core.windows.net/data-engineer/georef-
united-states-of-america-state.csv"

/home/hadoop/hadoop/bin/hdfs dfs -rm /ingest/*.*

/home/hadoop/hadoop/bin/hdfs dfs -put /home/hadoop/landing/*.* /ingest
```

3. Para orquestar el realizar las transformaciones en PySpark se utiliza el siguiente script ubicado en `/home/hadoop/scripts/transform_rental.py`

```python
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
sc = SparkContext('local')
spark = SparkSession(sc)
from pyspark.sql.functions import *
from pyspark.sql import HiveContext
hc = HiveContext(sc)

### Inicio del Script ###
# Leemos los csv desde HDFS y cargamos en dataframes
df_rental = spark.read.option("header", "true").option("sep",
",").csv("hdfs://172.17.0.2:9000/ingest/CarRentalData.csv")
df_georef = spark.read.option("header", "true").option("sep",
";").csv("hdfs://172.17.0.2:9000/ingest/georef.csv")

# Dropeamos columnas que no utilizaremos
df_rental = df_rental.drop('location.country', 'location.latitude',
'location.longitude', 'vehicle.type')
df_georef = df_georef.drop('State FIPS Code')

# Normalizamos nombres de columnas
# Importante mantener linea vacia debajo de cada bucle for
for column in df_rental.columns:
    df_rental = df_rental.withColumnRenamed(column,
column.replace('.','_'))
```

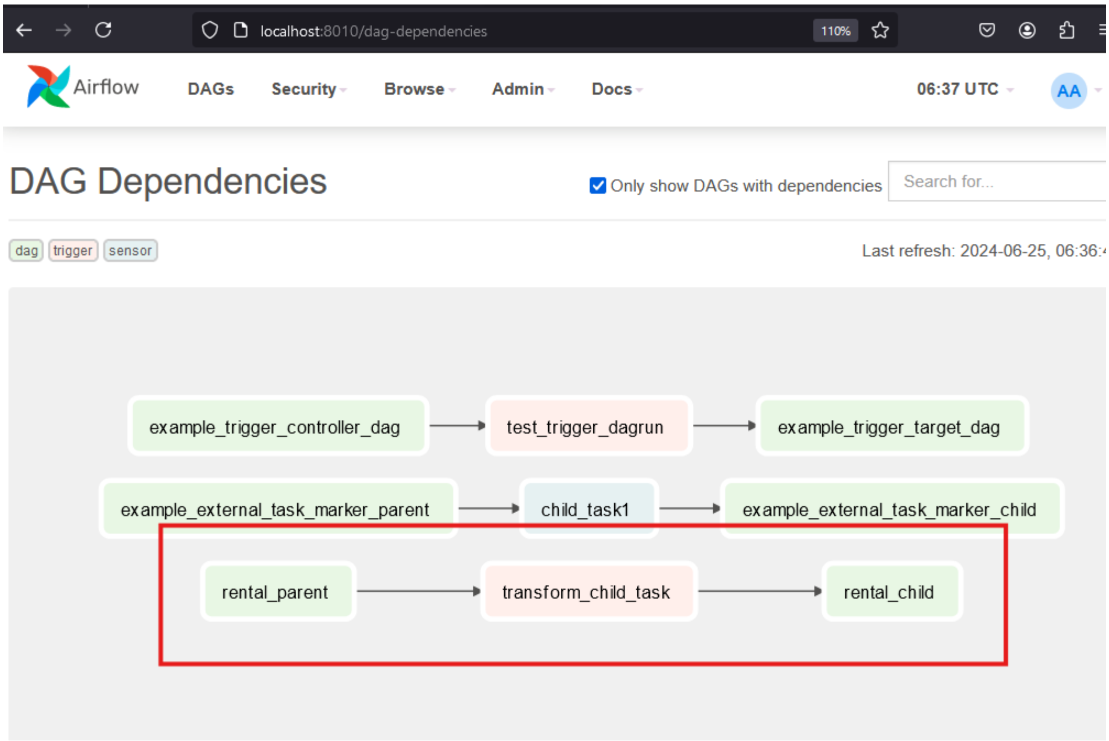
```
df_rental =
df_rental.withColumnRenamed("location_city","city").withColumnRenamed("loc
ation_state","state_name")


for column in df_rental.columns[-3:]:
    df_rental = df_rental.withColumnRenamed(column, column.split('_')[-1])

for column in df_georef.columns:
    df_georef = df_georef.withColumnRenamed(column,
column.replace('.','_').replace(' ','_'))

for column in df_georef.columns:
    df_georef =
df_georef.withColumnRenamed(column,'_'.join(column.split('_')[-2:]))

df_georef = df_georef.withColumnRenamed("Year", "year_georef")

# Redondeamos los float de 'rating' y castear a int
# Vuelvo a cargar sql.functinos porque levanta error
from pyspark.sql.functions import *
df_rental = df_rental.filter(column("rating").isNotNull())

# Eliminamos 'Texas'
df_rental = df_rental.filter("state_name != 'TX'")

# Joineamos las tablas en 'state_name'
df_rental = df_rental.join( df_georef,
df_rental.state_name==df_georef.state_abbreviation, 'left')

# Casteamos las variables no-string
cast_cols = ['renterTripsTaken', 'reviewCount', 'owner_id', 'rate_daily',
'year', 'year_georef', 'Code_State', 'GNIS_Code']
for column in cast_cols:
    df_rental = df_rental.withColumn( column, col(column).cast('int'))

# Redondeamos los float de 'rating' y castear a int
df_rental = df_rental.withColumn( 'rating',
round('rating').cast('integer') )

# Mayusculas por minusculas en 'fuelType'
df_rental = df_rental.withColumn('fuelType', lower('fuelType') )

# Dropeo columna 'Geo_Point' y 'Geo_Shape'
#  --AVERIGUAR COMO TRATAR CON LISTAS GEO-SHAPE
#  EL OBJETO DE LISTAS SE CARGA INCORRECCTAMENTE EN HIVE--
df_rental = df_rental.drop('Geo_Point', 'Geo_Shape')

# Creamos vistas con la data filtrada
df_rental.createOrReplaceTempView("df_rental_vista")

# Insertamos DFs filtrados en tablas de Hive
hc.sql("insert into car_rental_db.car_rental_analytics select * from
df_rental_vista;")

### Fin del Script ###
```

4. Para orquestar el proceso de ETL mediante Airflow se utilizaron dos scripts de manera que un proceso padre realice la ingesta y llame al proceso hijo para realizar las transformaciones. Los scripts que implementan los DAGs son:
   `/home/hadoop/airflow/dags/dag_rental_parent_TriggerDagRun.py`
   `/home/hadoop/airflow/dags/dag_rental_child_TriggerDagRun.py`

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime, timedelta
from airflow.operators.trigger_dagrun import TriggerDagRunOperator

args = {
    'owner' : 'airflow',
}

with DAG(
    dag_id="rental_parent",
    default_args=args,
    schedule_interval='@once',
    start_date=datetime(2020, 1, 1),
    dagrun_timeout=timedelta(minutes=60),
    tags=['ingest','parent'],
) as parent_dag:

    begin = EmptyOperator(
        task_id='begin_processing',
    )

    ingest = BashOperator(
        task_id='ingest_rental',
        # Don't drop the space at the end of the command or Jinja
will fail
        bash_command='/usr/bin/sh
/home/hadoop/scripts/ingest_rental.sh ',
    )

    transform_child_task = TriggerDagRunOperator(
        task_id="transform_child_task",
        trigger_dag_id='rental_child',
        wait_for_completion=False,
        reset_dag_run=True,
    )

    end = EmptyOperator(
        task_id='end_processing',
    )

    begin >> ingest >> transform_child_task >> end
```

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime, timedelta

args = {
    'owner' : 'airflow',
}

with DAG(
    dag_id="rental_child",
    default_args=args,
    schedule_interval=None,
    start_date=datetime(2020, 1, 1),
    dagrun_timeout=timedelta(minutes=60),
    tags=['transform','child'],
) as child_dag:

    begin = EmptyOperator(
        task_id='begin_processing',
    )

    transform_rental = BashOperator(
        task_id='transform_rental',
        # Don't drop the space at the end of the command or Jinja
will fail
        bash_command='ssh hadoop@172.17.0.2
/home/hadoop/spark/bin/spark-submit --files
/home/hadoop/hive/conf/hive-site.xml
/home/hadoop/scripts/transform_rental.py ',
    )

    end = EmptyOperator(
        task_id='end_processing',
    )

    begin >> transform_rental >> end
```

En la siguiente captura de pantalla se observa la dependencia entre DAGs generada por Airflow:


localhost:8010/dag-dependencies
110%
Airflow
DAGs
Security
Browse
Admin
Docs
06:37 UTC
AA

DAG Dependencies

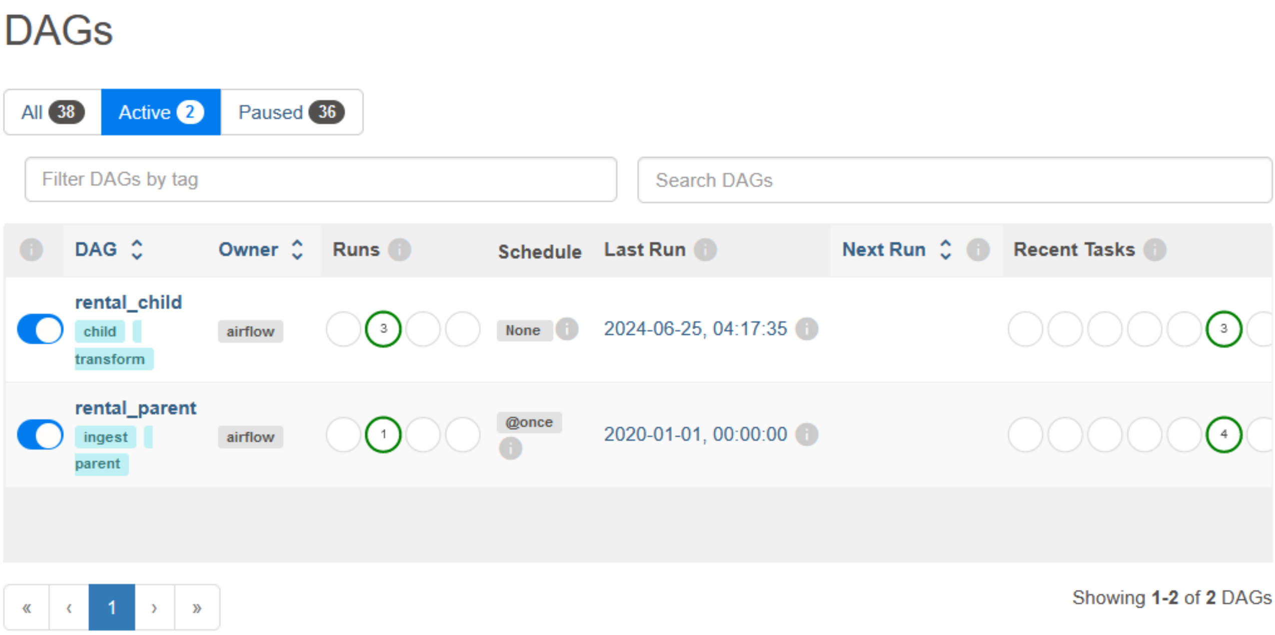
☑ Only show DAGs with dependencies

Search for...

Last refresh: 2024-06-25, 06:36:

example_trigger_controller_dag
test_trigger_dagrun
example_trigger_target_dag
example_external_task_marker_parent
child_task1
example_external_task_marker_child
rental_parent
transform_child_task
rental_child

Esta captura de pantalla muestra los DAGs orquestados por Airflow:



DAGs

All 38 | Active 2 | Paused 36

Filter DAGs by tag

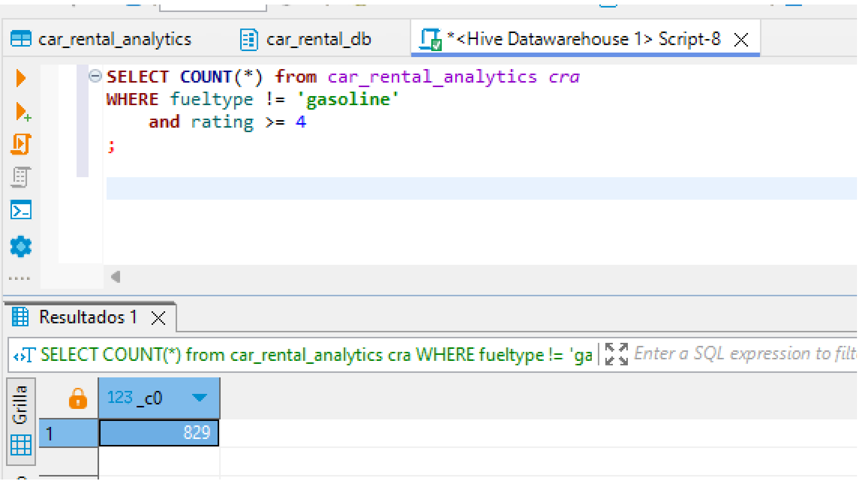
Search DAGs

	DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
	rental_child child transform	airflow	3	None	2024-06-25, 04:17:35		3
	rental_parent ingest parent	airflow	1	@once	2020-01-01, 00:00:00		4

Showing **1-2** of **2** DAGs

5. Querys

    a. 829 autos alquilados ecológicos con rating de al menos 4.



car_rental_analytics
car_rental_db
*<Hive Datawarehouse 1> Script-8
Resultados 1
Enter a SQL expression to filt
Grilla

```
SELECT COUNT(*) from car_rental_analytics cra
WHERE fueltype != 'gasoline'
    and rating >= 4
;
```

```
SELECT COUNT(*) from car_rental_analytics cra WHERE fueltype != 'ga
```

```
   123 _c0
1      829
```

    b.

6. _

7. _