13. Spark Streaming con Delta Lake

Utilización del formato de almacenamiento de datos Delta Lake para crear un lago de datos con un stream de datos aleatorios.

Para el ejercicio se parte de un conjunto de datos de prestamos (loans) y su situación (pagado, pendiente, retrasado, al día…)

13.1. Cargar los datos como Delta Lake

Cargamos los datos de ejemplo en el Delta Lake "loans_delta" usando formato parquet

```
spark.sql("set spark.sql.shuffle.partitions = 1")

sourcePath = "/databricks-datasets/learning-spark-v2/loans/loan-
risks.snappy.parquet"

deltaPath = "/tmp/loans_delta"

dbutils.fs.rm(deltaPath, recurse=True)

(spark.read.format("parquet").load(sourcePath)
  .write.format("delta").save(deltaPath))

spark.read.format("delta").load(deltaPath).createOrReplaceTempView("loans_de
lta")
print("Vista creada 'loans_delta'")
```

Verificamos que están los datos cargados

```
spark.sql("SELECT count(*) FROM loans_delta").show()
1
spark.sql("SELECT * FROM loans_delta LIMIT 5").show()
```

13.2. Creación de un stream de datos simulados

Las siguientes funciones generan un flujo de datos aleatorios de IDs de préstamos y de importes. Estos datos se agregan a la tabla de datos.

Estas funciones mientras están en marcha, agregan datos a la tabla por lo que definimos también las funciones para parar la generación y carga de datos "stop_all_streams()"

```
import random
import os
from pyspark.sql.functions import *
from pyspark.sql.types import *

def random_checkpoint_dir():
  return "/tmp/chkpt/%s" % str(random.randint(0, 10000))



states = ["CA", "TX", "NY", "WA"]

@udf(returnType=StringType())
def random_state():
  return str(random.choice(states))


def generate_and_append_data_stream():
```

```
  newLoanStreamDF = (spark.readStream.format("rate").option("rowsPerSecond",
5).load()
    .withColumn("loan_id", 10000 + col("value"))
    .withColumn("funded_amnt", (rand() * 5000 + 5000).cast("integer"))
    .withColumn("paid_amnt", col("funded_amnt") - (rand() * 2000))
    .withColumn("addr_state", random_state())
    .select("loan_id", "funded_amnt", "paid_amnt", "addr_state"))

  checkpointDir = random_checkpoint_dir()

  streamingQuery = (newLoanStreamDF.writeStream
    .format("delta")
    .option("checkpointLocation", random_checkpoint_dir())
    .trigger(processingTime = "10 seconds")
    .start(deltaPath))

  return streamingQuery

def stop_all_streams():

  print("Stopping all streams")
  for s in spark.streams.active:
    s.stop()
  print("Stopped all streams")
  print("Deleting checkpoints")
  dbutils.fs.rm("/tmp/chkpt/", True)
  print("Deleted checkpoints")
```

Para iniciar el proceso de generación de datos aleatoris ejecutar

```
streamingQuery = generate_and_append_data_stream()
```

Comprobar que se están agregando filas con

```
spark.sql("SELECT count(*) FROM loans_delta").show()
```

Detener la generación de datos

```
stop_all_streams()
```