

# Delta Lake

## 1. Qué es un Data Lake

Es un repositorio donde es posible almacenar tanto datos estructurados como datos no estructurados. Un data lake es el lugar hacia el que se ingestan datos de diversas fuentes y suele estar dividido en varias “capas” cuyos nombres varían.

Pero algo como:

- Landing – donde se almacena el dato “crudo”.
- Staging – donde se almacena el dato ya más limpio, normalmente en formato columnar.
- Analytics – donde se almacena el dato transformado, agregado, etc.

## 2. Ficheros Parquet

### Ventajas

- Formato columnar eficiente frente a archivos con formato row como csv.
- Menor tamaño y por tanto menor coste de almacenamiento.
- Mayor rapidez y por tanto menor coste de computación.

### Desventajas

- No soporta transacciones ACID.
- No soporta Time Travel (versionado de datos).
- No soporta updates.
- No tiene un historial de los cambios realizados.

## Formato de compresión Snappy

Es un formato de compresión cuyo objetivo no es el de comprimir al máximo, sino el de aportar muy altas velocidades con una compresión razonable.

## 3. Delta Table

### 3.1 Crear Delta Table (SQL)

**Importamos algunas librerías, borramos el warehouse y eliminamos las tablas si existen**

```
from delta.tables import *  
from pyspark.sql.types import *  
from pyspark.sql import functions as F
```

```
dbutils.fs.rm("/user/hive/warehouse", True)
```

```
Out[63]: True
```

```
%sql
```

```
DROP TABLE IF EXISTS tabla1;  
DROP TABLE IF EXISTS tabla2;  
DROP TABLE IF EXISTS tabla3;  
DROP TABLE IF EXISTS tabla1_parquet;  
DROP TABLE IF EXISTS vuelos;  
DROP TABLE IF EXISTS vuelos_copia;  
DROP TABLE IF EXISTS example;  
DROP TABLE IF EXISTS music;  
DROP TABLE IF EXISTS music_new;  
DROP TABLE IF EXISTS muebles1;  
DROP TABLE IF EXISTS muebles2;  
DROP TABLE IF EXISTS perro;  
DROP TABLE IF EXISTS perro3;  
DROP TABLE IF EXISTS perro_shallow;  
DROP TABLE IF EXISTS profesores;  
DROP TABLE IF EXISTS profesores_sc;  
DROP TABLE IF EXISTS profesores_dc;
```

```
OK
```

**Creamos una tabla delta con dos columnas, nombre y edad, e insertamos 3 registros diferentes**

```
spark.sql("CREATE OR REPLACE TABLE tabla1(nombre STRING, edad STRING) USING
DELTA")
spark.sql('INSERT INTO tabla1 VALUES("Pepe", "dieciséis"), ("Juan",
"veinticinco"), ("Antonia", "ochenta y cinco"), ("Margarita", "veintiuno")')
spark.sql("SELECT * FROM tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
| Antonia|ochenta y cinco|
|Margarita|      veintiuno|
|      Juan|    veinticinco|
|      Pepe|    dieciséis|
+-----+-----+
```

**Creamos otra tabla delta con dos columnas, nombre y edad, e insertamos 3 registros. Dos de ellos diferentes a los de la tabla1. En el caso del tercer registro, tenemos que actualizar la edad de Antonia, pues era un error**

```
spark.sql("CREATE OR REPLACE TABLE tabla2(nombre STRING, edad STRING) USING
DELTA")
spark.sql('INSERT INTO tabla2 VALUES("Juanita", "quince"), ("Alfredo",
"treinta y dos"), ("Antonia", "setenta y cinco")')
spark.sql("SELECT * FROM tabla2").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
| Antonia|setenta y cinco|
|Alfredo| treinta y dos|
|Juanita|      quince|
+-----+-----+
```

## 3.2 Ver forma de almacenamiento y delta\_log

**Observa los archivos que se han creado. ¿Cuántos hay?  
¿Qué hay en el directorio \_delta\_log?**

```
%fs
ls dbfs:/user/hive/warehouse/tabla1
```

	path
1	dbfs:/user/hive/warehouse/tabla1/_delta_log/
2	dbfs:/user/hive/warehouse/tabla1/part-00000-1b3ccfe8-d438-4a32-aba0-3732899270d1-c000
3	dbfs:/user/hive/warehouse/tabla1/part-00001-9eaa2ecd-bac6-4588-b454-d8aaac831e07-c000
4	dbfs:/user/hive/warehouse/tabla1/part-00002-030f883b-90f7-402a-8d4b-680faea2d996-c000.
5	dbfs:/user/hive/warehouse/tabla1/part-00003-b5ee460d-1c65-4c17-a0be-e5ea296361ed-c000

Showing all 5 rows.



%fs

```
ls dbfs:/user/hive/warehouse/tabla1/_delta_log
```

	path	name
1	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-0	.s3-opti
2	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-1	.s3-opti
3	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-2	.s3-opti
4	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000.crc	000000
5	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000.json	000000
6	dbfs:/user/hive/warehouse/tabla1/_delta_log/000000000000000000001.crc	000000
7	dbfs:/user/hive/warehouse/tabla1/_delta_log/000000000000000000001.json	000000

Showing all 7 rows.



### 3.3 Convertir tabla Parquet a Delta con SQL

**Vamos a crear una TempView llamada vuelos\_temp a partir de un csv**

```
dfVuelos = spark.read.format("csv").option("header",
True).load("dbfs:/FileStore/tables/flights.csv")
dfVuelos.createOrReplaceTempView("vuelos_temp")
```

**A continuación, vamos a guardar la tabla en formato parquet a partir de vuelos\_temp**

```
spark.sql("CREATE OR REPLACE TABLE vuelos STORED AS PARQUET AS SELECT * FROM vuelos_temp")
```

```
Out[67]: DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
```

## Finalmente, convertimos la tabla a delta

```
spark.sql("CONVERT TO DELTA vuelos")
```

```
Out[68]: DataFrame[]
```

## 3.4 Guardar los datos

### SQL – INSERT INTO SELECT...

**De forma similar a CREATE TABLE AS SELECT, podemos usar INSERT INTO SELECT, aunque debemos crear una tabla con el esquema apropiado antes**

```
spark.sql("CREATE OR REPLACE TABLE vuelos_copia(Year STRING, Month STRING, DayOfMonth STRING, DayOfWeek STRING, DepTime STRING, CRSDepTime STRING, ArrTime STRING, CRSArrTime STRING, UniqueCarrier STRING, FlightNum STRING, TailNum STRING, ActualElapsedTime STRING, CRSElapsedTime STRING, AirTime STRING, ArrDelay STRING, DepDelay STRING, Origin STRING, Dest STRING, Distance STRING, TaxiIn STRING, TaxiOut STRING, Cancelled STRING, CancellationCode STRING, Diverted STRING, CarrierDelay STRING, WeatherDelay STRING, NASDelay STRING, SecurityDelay STRING, LateAircraftDelay STRING)")
```

```
spark.sql("INSERT INTO vuelos_copia SELECT * FROM vuelos_temp")
```

```
spark.sql("SELECT * FROM vuelos_copia").show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Year|Month|DayOfMonth|DayOfWeek|DepTime|CRSDepTime|ArrTime|CRSArrTime|UniqueCarrier|FlightNum|TailNum|ActualElapsedTime|CRSElapsedTime|AirTime|ArrDelay|DepDelay|Origin|Dest|Distance|TaxiIn|TaxiOut|Cancelled|CancellationCode|Diverted|CarrierDelay|WeatherDelay|NASDelay|SecurityDelay|LateAircraftDelay|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|2008|1|3|4|2003|1955|2211|2225|WN|335|N712SW|128|150|116|-14|8|IAD|TPA|810|4|8|0|null|0|
```

NA	NA	NA	NA	NA
2008	1	3	4	754
WN	3231	N772SW	128	145
19	IAD	TPA	810	5
NA	NA	NA	NA	NA
12008	1	3	4	628
				620
				804
				750

## Usando Spark DataFrame/Dataset API – saveAsTable

Otra forma de guardar una tabla es usando el método `saveAsTable()`.

Guardamos una tabla a partir del `dfVuelos`

```
dfVuelos.write.format("delta").mode("overwrite").saveAsTable("vuelos")
```

## Usando Spark DataFrame/Dataset API – save

Del mismo modo, podemos usar el método `save()`. Guardamos una tabla a partir de `dfVuelos`, en este caso con el modo 'overwrite'

```
dfVuelos.write.format("delta").mode("overwrite").save("/user/hive/warehouse/vuelos")
```

Comparemos la forma de guardar una tabla en formato delta con guardar una tabla en formato parquet

```
#dfVuelos.write.format("parquet").mode("overwrite").save("/user/hive/warehouse/vuelos")
```

## 4. Ejercicios

### 4.1 Append Data

Vamos a añadir a nuestra tabla1 un nuevo registro. Pero antes vamos a ver cuántos archivos hay en la tabla de nuestro warehouse y en el directorio `_delta_log`

```
%fs
ls dbfs:/user/hive/warehouse/tabla1
```

	path
1	dbfs:/user/hive/warehouse/tabla1/_delta_log/
2	dbfs:/user/hive/warehouse/tabla1/part-00000-1b3ccfe8-d438-4a32-aba0-3732899270d1-c000
3	dbfs:/user/hive/warehouse/tabla1/part-00001-9eaa2ecd-bac6-4588-b454-d8aaac831e07-c000
4	dbfs:/user/hive/warehouse/tabla1/part-00002-030f883b-90f7-402a-8d4b-680faea2d996-c000.
5	dbfs:/user/hive/warehouse/tabla1/part-00003-b5ee460d-1c65-4c17-a0be-e5ea296361ed-c000

Showing all 5 rows.



```
%fs
```

```
ls dbfs:/user/hive/warehouse/tabla1/_delta_log
```

	path	name
1	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-0	.s3-opti
2	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-1	.s3-opti
3	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-2	.s3-opti
4	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000.crc	000000
5	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000.json	000000
6	dbfs:/user/hive/warehouse/tabla1/_delta_log/000000000000000000001.crc	000000
7	dbfs:/user/hive/warehouse/tabla1/_delta_log/000000000000000000001.json	000000

Showing all 7 rows.



## Usamos DESCRIBE HISTORY para ver cuál es la versión actual de la tabla1

```
spark.sql("DESCRIBE HISTORY tabla1").show()
```

```

+-----+-----+-----+-----+-----+
|version|timestamp|userId|userName|operation|operationParameters|job|notebook|clusterId|readVersion|isolationLevel|isBlindAppend|operationMetrics|userMetadata|
+-----+-----+-----+-----+-----+
|1|2021-07-21 07:32:25|1958270740196902|marcos.rc91@gmail...|

```

```
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
0|WriteSerializable|          true|{numFiles -> 4, n...|          null|
|          0|2021-07-21 07:32:20|1958270740196902|marcos.rc91@gmail...|CREATE OR
REPLACE...|{isManaged -> tru...|null|{3369027357220086}|0721-065836-dizzy95
0|          null|SnapshotIsolation|          true|          {}|          n
ull|
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+
---+
```

## Introducimos el nuevo registro

```
spark.sql("INSERT INTO tabla1 VALUES('Federica', 'cuarenta y dos')")
```

```
Out[74]: DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
```

## ¿Ha cambiado algo en los archivos que se encuentran en el warehouse? ¿Y en el directorio \_delta\_log?

```
%fs
ls dbfs:/user/hive/warehouse/tabla1
```

	path
1	dbfs:/user/hive/warehouse/tabla1/_delta_log/
2	dbfs:/user/hive/warehouse/tabla1/part-00000-1b3ccfe8-d438-4a32-aba0-3732899270d1-c000
3	dbfs:/user/hive/warehouse/tabla1/part-00000-f1195e73-1423-4709-b8f6-6d261c5d5033-c000
4	dbfs:/user/hive/warehouse/tabla1/part-00001-9eaa2ecd-bac6-4588-b454-d8aaac831e07-c000
5	dbfs:/user/hive/warehouse/tabla1/part-00002-030f883b-90f7-402a-8d4b-680faea2d996-c000.
6	dbfs:/user/hive/warehouse/tabla1/part-00003-b5ee460d-1c65-4c17-a0be-e5ea296361ed-c000

Showing all 6 rows.



```
%fs
ls dbfs:/user/hive/warehouse/tabla1/_delta_log
```



[illegible]

Showing all 9 rows.



## ¿Cuál es ahora la versión actual de la tabla1? Compruébalo

```
spark.sql("DESCRIBE HISTORY tabla1").show()
```

```
+-----+-----+-----+-----+
|version|          timestamp|        userId|           userName|
operation| operationParameters| job|         notebook|      clusterId|
readVersion| isolationLevel|isBlindAppend|    operationMetrics|userMetadata|
a|
+-----+-----+-----+-----+
|                |
+-----+-----+-----+-----+
|                |
+-----+-----+-----+-----+
---+
|            2|2021-07-21 07:33:39|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
1|WriteSerializable|              true|{numFiles -> 1, n...|             null|
|            1|2021-07-21 07:32:25|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
0|WriteSerializable|              true|{numFiles -> 4, n...|             null|
|            0|2021-07-21 07:32:20|1958270740196902|marcos.rc91@gmail...|CREATE OR
REPLACE...|{isManaged -> tru...|null|{3369027357220086}|0721-065836-dizzy95
0|       null|SnapshotIsolation|               true|                  {}|                 n
ull|
+-----+-----+-----+-----+
|                |
+-----+-----+-----+-----+
|                |
+-----+-----+-----+-----+
---+
```

## 4.2 Append, Update Data

### Vamos a añadir un nuevo registro a tabla1: Dora, de dieciocho años

```
spark.sql("INSERT INTO tabla1 VALUES ('Dora', 'dieciocho')")
spark.sql("SELECT * FROM tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
|  Antonia|ochenta y cinco|
| Federica| cuarenta y dos|
|Margarita|      veintiuno|
|    Juan|    veinticinco|
|    Pepe|     dieciséis|
|    Dora|     dieciocho|
+-----+-----+
```

### Ahora vamos a actualizar la edad de Dora, pues en realidad tiene diecinueve

```
spark.sql("UPDATE tabla1 SET edad = 'diecinueve' WHERE nombre = 'Dora'")
spark.sql("SELECT * FROM tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
|  Antonia|ochenta y cinco|
| Federica| cuarenta y dos|
|Margarita|      veintiuno|
|    Juan|    veinticinco|
|    Pepe|     dieciséis|
|    Dora|    diecinueve|
+-----+-----+
```

### Vamos a crear una tabla parquet a partir de tabla1

```
spark.sql("CREATE TABLE tabla1_parquet STORED AS PARQUET AS SELECT * FROM tabla1")
spark.sql("SELECT * FROM tabla1_parquet").show()
```

nombre	edad
Antonia	ochenta y cinco
Federica	cuarenta y dos
Margarita	veintiuno
Juan	veinticinco
Pepe	dieciséis
Dora	diecinueve

## Insertamos un nuevo registro en tabla1\_parquet

```
spark.sql("INSERT INTO tabla1_parquet VALUES ('Ágata', 'treinta')")
```

```
Out[79]: DataFrame[]
```

**Si intentásemos actualizar la edad de Ágata en tabla1\_parquet, de treinta a veinte años. ¿Cuál sería el resultado?**

```
#spark.sql("UPDATE tabla1_parquet SET edad = 'veinte' WHERE nombre = 'Ágata'")
```

## 4.3 Overwrite Data

**Creamos dos tablas llamadas music y music\_new, con dos columnas tipo STRING**

```
spark.sql("CREATE OR REPLACE TABLE music(cancion STRING, genero STRING)
USING DELTA")
spark.sql("CREATE OR REPLACE TABLE music_new(cancion STRING, genero STRING)
USING DELTA")
```

```
Out[81]: DataFrame[]
```

**Insertamos algunos registros diferentes para cada una de las tablas**

```
spark.sql("INSERT INTO music VALUES('let it be', 'pop rock'), ('some say', 'pop punk'), ('einmal um die welt', 'rap')")
spark.sql("INSERT INTO music_new VALUES('here comes the sun', 'pop rock'), ('still waiting', 'pop punk'), ('bye bye', 'rap')")
```

```
Out[82]: DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
```

```
spark.sql("SELECT * FROM music").show()
```

```
+-----+-----+
|      cancion| genero|
+-----+-----+
|einmal um die welt|    rap|
|      let it be|pop rock|
|      some say|pop punk|
+-----+-----+
```

```
spark.sql("SELECT * FROM music_new").show()
```

```
+-----+-----+
|      cancion| genero|
+-----+-----+
|here comes the sun|pop rock|
|    still waiting|pop punk|
|      bye bye|    rap|
+-----+-----+
```

## Sobreescribimos la tabla music con el contenido de music\_new

```
spark.sql("INSERT OVERWRITE TABLE music SELECT * FROM music_new")
spark.sql("SELECT * FROM music").show()
```

```
+-----+-----+
|      cancion| genero|
+-----+-----+
|here comes the sun|pop rock|
|    still waiting|pop punk|
|      bye bye|    rap|
+-----+-----+
```

## Vamos a hacer una consulta para ver la versión anterior

#Esto no funciona en local: <https://github.com/delta-io/delta/issues/634>  
 spark.sql("SELECT \* FROM music VERSION AS OF 1").show()

```
+-----+-----+
|      cancion| genero|
+-----+-----+
|einmal um die welt|    rap|
|      let it be|pop rock|
|      some say|pop punk|
+-----+-----+
```

## 4.4 MERGE

**Hacemos un merge entre la tabla1 y la tabla2. Cuando el nombre sea igual en ambas tablas, tomará el valor de edad correspondiente a tabla2. Si los nombres no coinciden entre las dos tablas, simplemente se insertarán como registros nuevos.**

```
spark.sql("MERGE INTO tabla1 USING tabla2 ON tabla1.nombre = tabla2.nombre
WHEN MATCHED THEN UPDATE SET edad = tabla2.edad WHEN NOT MATCHED THEN INSERT
(nombre, edad) VALUES(tabla2.nombre, tabla2.edad)")
```

```
spark.sql("SELECT * FROM tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
| Federica| cuarenta y dos|
| Margarita|    veintiuno|
|      Juan|   veinticinco|
|      Pepe|   dieciséis|
|      Dora|   diecinueve|
| Alfredo| treinta y dos|
| Antonia| setenta y cinco|
| Juanita|      quince|
+-----+-----+
```

**¿Qué ha sucedido en el directorio `_delta_log`?**

```
%fs
```

```
ls dbfs:/user/hive/warehouse/tabla1/_delta_log
```

	path	name
1	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-0	.s3-opti
2	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-1	.s3-opti
3	dbfs:/user/hive/warehouse/tabla1/_delta_log/.s3-optimization-2	.s3-opti
4	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000000000000000.crc	000000
5	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000000000000000.json	000000
6	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000000000000001.crc	000000
7	dbfs:/user/hive/warehouse/tabla1/_delta_log/00000000000000000000000000000001.json	000000

Showing all 15 rows.



**Abre los archivos .json y compáralos. ¿Hay algún tipo de información interesante en ellos?**

## 4.5. Time Travel, versionado

### Vamos a eliminar a Alfredo de tabla1

```
spark.sql("DELETE FROM tabla1 WHERE nombre = 'Alfredo'")
```

```
spark.sql("SELECT * FROM tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
| Federica| cuarenta y dos|
| Margarita|      veintiuno|
|      Juan|    veinticinco|
|      Pepe|    dieciséis|
|      Dora|    diecinueve|
|  Antonia| setenta y cinco|
|  Juanita|      quince|
+-----+-----+
```

**¿Cuál es ahora la versión actual de la tabla1? Compruébalo**

```
spark.sql("DESCRIBE HISTORY tabla1").show()
```

```

+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+
---+
|version|      timestamp|      userId|      userName|
operation| operationParameters| job|      notebook|      clusterId|
readVersion| isolationLevel|isBlindAppend| operationMetrics|userMetadata|
a|
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+
---+
|      6|2021-07-21 07:34:43|1958270740196902|marcos.rc91@gmail...|
DELETE|{predicate -> ["(...|null|{3369027357220086}|0721-065836-dizzy950|
5|WriteSerializable|      false|{numRemovedFiles ...|      null|
|      5|2021-07-21 07:34:36|1958270740196902|marcos.rc91@gmail...|
MERGE|{predicate -> (sp...|null|{3369027357220086}|0721-065836-dizzy950|
4|WriteSerializable|      false|{numTargetRowsCop...|      null|
|      4|2021-07-21 07:33:54|1958270740196902|marcos.rc91@gmail...|
UPDATE|{predicate -> (no...|null|{3369027357220086}|0721-065836-dizzy950|
3|WriteSerializable|      false|{numRemovedFiles ...|      null|
|      3|2021-07-21 07:33:46|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
2|WriteSerializable|      true|{numFiles -> 1, n...|      null|
|      2|2021-07-21 07:33:39|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
1|WriteSerializable|      true|{numFiles -> 1, n...|      null|
|      1|2021-07-21 07:32:25|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
0|WriteSerializable|      true|{numFiles -> 4, n...|      null|
|      0|2021-07-21 07:32:20|1958270740196902|marcos.rc91@gmail...|CREATE OR
REPLACE...|{isManaged -> tru...|null|{3369027357220086}|0721-065836-dizzy95
0|      null|SnapshotIsolation|      true|      {}|      n
ull|
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+
---+

```

**Utilizando VERSION AS OF, cargamos una versión de la tabla1 anterior a la eliminación de Alfredo y la mostramos por pantalla para asegurarnos de que aparece**

#Esto no funciona en local: <https://github.com/delta-io/delta/issues/634>  
 spark.sql("SELECT \* FROM tabla1 VERSION AS OF 1").show()

#También es posible:

```
#spark.read.format("delta").option("versionAsOf", 1).load("/user/hive/warehouse/tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
|  Antonia|ochenta y cinco|
|Margarita|      veintiuno|
|    Juan|    veinticinco|
|    Pepe|    dieciséis|
+-----+-----+
```

## Comprueba si esto último ha sustituido la tabla1 por una versión anterior

```
spark.sql("SELECT * FROM tabla1").show()
```

```
+-----+-----+
| nombre|      edad|
+-----+-----+
| Federica|cuarenta y dos|
|Margarita|      veintiuno|
|    Juan|    veinticinco|
|    Pepe|    dieciséis|
|    Dora|    diecinueve|
|  Antonia|setenta y cinco|
|  Juanita|      quince|
+-----+-----+
```

## Restaurar una versión anterior

#Esto no funciona en local  
 spark.sql("RESTORE tabla1 VERSION AS OF 1")

#También es posible:

```
#deltaTable = DeltaTable.forName(spark, "tabla1")
#deltaTable.restoreToVersion(1)
```

```
spark.sql("SELECT * FROM tabla1").show()
```



```

+-----+-----+
| nombre|      edad|
+-----+-----+
|  Antonia|ochenta y cinco|
|Margarita|    veintiuno|
|    Juan|    veinticinco|
|    Pepe|    dieciséis|
+-----+-----+

```

## 5. API DeltaTable

<https://docs.delta.io/latest/api/python/index.html> (<https://docs.delta.io/latest/api/python/index.html>)

### Cargar tabla por ruta con forPath()

```
dtTabla1 = DeltaTable.forPath(spark, "/user/hive/warehouse/tabla1")
dtTabla1.toDF().show()
```

```

+-----+-----+
| nombre|      edad|
+-----+-----+
|  Antonia|ochenta y cinco|
|Margarita|    veintiuno|
|    Juan|    veinticinco|
|    Pepe|    dieciséis|
+-----+-----+

```

### Cargar tabla por nombre con forName()

```
dtTabla2 = DeltaTable.forName(spark, "tabla2")
dtTabla2.toDF().show()
```

```

+-----+-----+
| nombre|      edad|
+-----+-----+
|Antonía|setenta y cinco|
|Alfredo|  treinta y dos|
|Juanita|    quince|
+-----+-----+

```

## Ver historia de la tabla

```
dtTabla1.history().show()
```

```
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+
|version|          timestamp|          userId|          userName|
operation| operationParameters| job|          notebook|          clusterId|
readVersion| isolationLevel|isBlindAppend| operationMetrics|userMetadata|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+
|          7|2021-07-21 07:35:38|1958270740196902|marcos.rc91@gmail...|
RESTORE|{version -> 1, ti...|null|{3369027357220086}|0721-065836-dizzy950|
6|          Serializable|          false|{numRestoredFiles...|          null|
|          6|2021-07-21 07:34:43|1958270740196902|marcos.rc91@gmail...|
DELETE|{predicate -> ["(...|null|{3369027357220086}|0721-065836-dizzy950|
5|WriteSerializable|          false|{numRemovedFiles ...|          null|
|          5|2021-07-21 07:34:36|1958270740196902|marcos.rc91@gmail...|
MERGE|{predicate -> (sp...|null|{3369027357220086}|0721-065836-dizzy950|
4|WriteSerializable|          false|{numTargetRowsCop...|          null|
|          4|2021-07-21 07:33:54|1958270740196902|marcos.rc91@gmail...|
UPDATE|{predicate -> (no...|null|{3369027357220086}|0721-065836-dizzy950|
3|WriteSerializable|          false|{numRemovedFiles ...|          null|
|          3|2021-07-21 07:33:46|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
2|WriteSerializable|          true|{numFiles -> 1, n...|          null|
|          2|2021-07-21 07:33:39|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
1|WriteSerializable|          true|{numFiles -> 1, n...|          null|
|          1|2021-07-21 07:32:25|1958270740196902|marcos.rc91@gmail...|
WRITE|{mode -> Append, ...|null|{3369027357220086}|0721-065836-dizzy950|
0|WriteSerializable|          true|{numFiles -> 4, n...|          null|
|          0|2021-07-21 07:32:20|1958270740196902|marcos.rc91@gmail...|CREATE OR
REPLACE...|{isManaged -> tru...|null|{3369027357220086}|0721-065836-dizzy95
0|          null|SnapshotIsolation|          true|          {}|          n
ull|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+
```

**Si añadimos un número n como argumento al método history() podemos ver únicamente las últimas n versiones de la tabla. Compruébalo**

```
dtTabla1.history(2).show()
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
--+-+-----+-----+-----+-----+
|version|          timestamp|      userId|      userName|operatio
n| operationParameters| job|      notebook|      clusterId|readVers
ion|  isolationLevel|isBlindAppend|  operationMetrics|userMetadata|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
--+-+-----+-----+-----+-----+
|      7|2021-07-21 07:35:38|1958270740196902|marcos.rc91@gmail...| RESTOR
E|{version -> 1, ti...|null|{3369027357220086}|0721-065836-dizzy950|
6|  Serializable|      false|{numRestoredFiles...|      null|
|      6|2021-07-21 07:34:43|1958270740196902|marcos.rc91@gmail...|  DELET
E|{predicate -> ["(...|null|{3369027357220086}|0721-065836-dizzy950|
5|WriteSerializable|      false|{numRemovedFiles ...|      null|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
--+-+-----+-----+-----+-----+
```

```
dtTabla1.history(1).show()
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
--+-+-----+-----+-----+-----+
|version|          timestamp|      userId|      userName|operatio
n| operationParameters| job|      notebook|      clusterId|readVers
ion|isolationLevel|isBlindAppend|  operationMetrics|userMetadata|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
--+-+-----+-----+-----+-----+
|      7|2021-07-21 07:35:38|1958270740196902|marcos.rc91@gmail...| RESTOR
E|{version -> 1, ti...|null|{3369027357220086}|0721-065836-dizzy950|
6|  Serializable|      false|{numRestoredFiles...|      null|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
--+-+-----+-----+-----+-----+
```

```
dtTabla1.history(0).show()
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
|version|timestamp|userId|userName|operation|operationParameters|job|noteboo
k|clusterId|readVersion|isolationLevel|isBlindAppend|operationMetrics|userMe
tadata|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+

```

## Restaurar tabla a versión específica

#No funciona en local

```

#deltaTable = DeltaTable.forName(spark, "tabla1")
#deltaTable.restoreToVersion(1)

```

## 5.1 DDL

### Crear tabla delta

#### create()

```

dtExample = DeltaTable.create(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()
dtExample.toDF().show()

```

```

+---+---+
| c1| c2|
+---+---+
+---+---+

```

También es posible pasar un schema al método `addColumns()`:

```
.addColumns(df.schema)
```

### **replace()**

```
'''dtExample = DeltaTable.replace(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()
dtExample.toDF().show()'''
```

```
Out[101]: 'dtExample = DeltaTable.replace(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()\ndtExample.toDF().show()'
```

### **createOrReplace()**

```
'''dtExample = DeltaTable.createOrReplace(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()
dtExample.toDF().show()'''
```

```
Out[102]: 'dtExample = DeltaTable.createOrReplace(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()\ndtExample.toDF().show()'
```

### **createIfNotExists()**

```
'''dtExample = DeltaTable.createIfNotExists(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()
dtExample.toDF().show()'''
```

```
Out[103]: 'dtExample = DeltaTable.createIfNotExists(spark).tableName("example") \
    .addColumn("c1", dataType = "INT", nullable = False) \
    .addColumn("c2", dataType = IntegerType(), generatedAlwaysAs = "c1 + 1") \
    .partitionedBy("c1") \
    .execute()\ndtExample.toDF().show()'
```

## 5.2 DML

### Insertamos unos registros a la tabla2

```
%sql
```

```
DELETE FROM tabla2;
```

```
INSERT INTO tabla2 VALUES('Juanita', 'trece'), ('Manolita', 'catorce'),  
('Lolita', 'quince');
```

```
SELECT * FROM tabla2;
```

	nombre ▲	edad ▲
1	Manolita	catorce
2	Juanita	trece
3	Lolita	quince

Showing all 3 rows.



### Borrar registros de una tabla con delete()

```
dtTabla2.delete("nombre = 'Juanita'")
```

#También es posible:

```
#dtTabla2.delete(F.col("nombre") == "Juanita")
```

```
dtTabla2.toDF().show()
```

```
+-----+-----+
| nombre|  edad|
+-----+-----+
|Manolita|catorce|
|  Lolita| quince|
+-----+-----+
```

### Borrar todos los registros de una tabla con delete() sin pasar ningún argumento

```
dtTabla2.delete()
dtTabla2.toDF().show()
```

```
+-----+-----+
| nombre|edad|
+-----+-----+
+-----+-----+
```

## ¿Qué hay en tabla2?

```
spark.sql("SELECT * FROM tabla2").show()
```

```
+-----+-----+
| nombre|edad|
+-----+-----+
+-----+-----+
```

## Actualizar registros con el método update()

### Cargamos la tabla vuelos en un deltaTable

```
dtVuelos = DeltaTable.forName(spark, "vuelos")
```

### Sin usar las funciones de Spark SQL

```
dtVuelos.update(
  condition = "Dest != 'TPA'",
  set = { "Dest": "'TPA'" } )
```

### Usando las funciones de Spark SQL

```
'''dtVuelos.update(
  condition = F.col("Dest") != "TPA",
  set = { "Dest": F.lit("TPA") } )'''
```

```
Out[109]: 'dtVuelos.update(\n    condition = F.col("Dest") != "TPA",\n    se
t = { "Dest": F.lit("TPA") } )'
```

## Upsert con merge()

**Creamos dos tablas delta. Las columnas serán “nombre” y “descripcion”, de tipo STRING**

```
dtMuebles1 = DeltaTable.createOrReplace(spark).tableName("muebles1") \
    .addColumn("nombre", dataType = "STRING", nullable = False) \
    .addColumn("descripcion", dataType = "STRING", nullable = False) \
    .execute()
```

```
dtMuebles2 = DeltaTable.createOrReplace(spark).tableName("muebles2") \
    .addColumn("nombre", dataType = "STRING", nullable = False) \
    .addColumn("descripcion", dataType = "STRING", nullable = False) \
    .execute()
```

**Insertamos algunos registros en las tablas. Nos aseguramos de que algunos nombres son iguales, aunque las descripciones serán diferentes**

```
spark.sql("INSERT INTO muebles1 VALUES('silla', 'grande'), ('mesita', 'azul'), ('sofá', 'cuero'), ('escritorio', 'caoba')")
spark.sql("INSERT INTO muebles2 VALUES('silla', 'pequeña'), ('mesa', 'blanca'), ('cama', 'doble'), ('escritorio', 'plástico')")
```

```
Out[111]: DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
```

```
dtMuebles1.toDF().show()
```

```
+-----+-----+
|  nombre|descripcion|
+-----+-----+
|escritorio|      caoba|
|   silla|      grande|
|  mesita|        azul|
|   sofá|      cuero|
+-----+-----+
```

```
dtMuebles2.toDF().show()
```

```
+-----+-----+
|  nombre|descripcion|
+-----+-----+
|escritorio|  plástico|
|   silla|  pequeña|
|   mesa|   blanca|
|   cama|   doble|
+-----+-----+
```



```
+-----+-----+
```

**Al hacer merge, una de las dos tablas tiene que ser un objeto tipo DataFrame, así que convertimos muebles2 en un df**

```
dfMuebles2 = dtMuebles2.toDF()
```

### Sin usar las funciones de Spark SQL

```
dtMuebles1.alias("muebles1").merge(
  source = dfMuebles2.alias("muebles2"),
  condition = "muebles1.nombre = muebles2.nombre"
).whenMatchedUpdate(set =
  {
    "muebles1.descripcion": "muebles2.descripcion"
  }
).whenNotMatchedInsertAll() \
.execute()
```

```
dtMuebles1.toDF().show()
```

```
+-----+-----+
|  nombre|descripcion|
+-----+-----+
|   cama|      doble|
|escritorio| plástico|
|   mesa|     blanca|
|   silla|   pequeña|
|  mesita|       azul|
|   sofá|     cuero|
+-----+-----+
```

### Usando las funciones de Spark SQL

```
'''dtMuebles1.alias("muebles1").merge(
    source = dfMuebles2.alias("muebles2"),
    condition = F.expr("muebles1.nombre = muebles2.nombre")
).whenMatchedUpdate(set =
    {
        "descripcion": F.col("muebles2.descripcion")
    }
).whenNotMatchedInsertAll() \
.execute()

dtMuebles1.toDF().show()
'''
```

```
Out[116]: 'dtMuebles1.alias("muebles1").merge(\n    source = dfMuebles2.alia
s("muebles2"),\n    condition = F.expr("muebles1.nombre = muebles2.nombre")\
n ).whenMatchedUpdate(set =\n    {\n        "descripcion": F.col("muebles2.de
scripcion")\n    }\n ).whenNotMatchedInsertAll() .execute()\n\ndtMuebles
1.toDF().show()\n'
```

## 6. Clone Table

Podemos clonar las tablas, pero cada una de ellas mantendrá su propio transaction log. Hay dos formas de clonar las tablas delta:

- shallow: los archivos originales no se duplican, la tabla clonada utiliza los archivos de la original, pero tiene su propio transaction log.
- deep: los archivos originales se copian en la nueva tabla clonada.

### 6.1 Shallow

**Creamos una tabla llamada profesores e insertamos algunos valores**

```
%sql
CREATE OR REPLACE TABLE profesores(nombre STRING, asignatura STRING) USING
DELTA;
INSERT INTO profesores VALUES('Arturo', 'música'), ('Lola', 'historia'),
('Mercedes', 'lengua');
SELECT * FROM profesores;
```

	nombre ▲	asignatura ▲	
1	Mercedes	lengua	
2	Arturo	música	
3	Lola	historia	

Showing all 3 rows.



## Clonamos la tabla profesores a profesores\_sc con SHALLOW CLONE

```
%sql
```

```
CREATE TABLE profesores_sc SHALLOW CLONE profesores;
SELECT * FROM profesores_sc;
```

	nombre ▲	asignatura ▲	
1	Mercedes	lengua	
2	Arturo	música	
3	Lola	historia	

Showing all 3 rows.



## ¿Qué (no) hay en el directorio profesores\_sc?

```
%fs
```

```
ls /user/hive/warehouse/profesores_sc
```

	path ▲	name ▲	size ▲
1	dbfs:/user/hive/warehouse/profesores_sc/_delta_log/	_delta_log/	0

Showing all 1 rows.



## 6.2 Deep

### Clonamos la tabla profesores a profesores\_dc con DEEP CLONE

```
%sql
```

```
CREATE TABLE profesores_dc DEEP CLONE profesores;  
SELECT * FROM profesores_dc;
```

	nombre ▲	asignatura ▲
1	Mercedes	lengua
2	Arturo	música
3	Lola	historia

Showing all 3 rows.



### ¿Qué hay en el directorio profesores\_dc?

```
%fs
```

```
ls /user/hive/warehouse/profesores_dc/
```

	path
1	dbfs:/user/hive/warehouse/profesores_dc/_delta_log/
2	dbfs:/user/hive/warehouse/profesores_dc/part-00000-8aa217bb-92b8-4638-8a98-5e7bc39469
3	dbfs:/user/hive/warehouse/profesores_dc/part-00001-7a80394e-4984-443c-96cf-75cd9f17aa
4	dbfs:/user/hive/warehouse/profesores_dc/part-00002-d0c4b755-5723-4068-91a3-1aea0b3ba

Showing all 4 rows.



## 7. Z-ORDER

Z-ORDER funciona como un clustered index. Agrupa los datos por una columna X para que al hacer una query haya archivos que no sea necesario leer, ya que el valor buscado no estará en todos ellos. Eso se llama data skipping.

### Casos de uso

- Usamos Z-ORDER cuando queremos optimizar las consultas por una columna.
- Cuando queremos evitar el uso de IO innecesario, especialmente si hay un coste asociado.

### Vamos a añadir los registros de nombres.csv a una tabla nueva

```
%sql
```

```
CREATE OR REPLACE TABLE tabla3 (nombre STRING, edad STRING);
```

```
OK
```

```
dfTemp = spark.read.format("csv").load("dbfs:/FileStore/tables/nombres.csv")
dfTemp.createOrReplaceTempView("nombres_temp")
spark.sql("INSERT OVERWRITE tabla3 SELECT * FROM nombres_temp")
spark.sql("SELECT * FROM tabla3").show()
```

```
+-----+-----+
|  nombre|      edad|
+-----+-----+
|Margarita|treinta y uno|
|Margarita|treinta y uno|
|Margarita|treinta y uno|
|Margarita|treinta y uno|
|Margarita|treinta y uno|
|  Antonio|    cuarenta|
|   Lucía|sesenta y dos|
|Cecilia|veinticuatro|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
|  Perico|  diecinueve|
```

Perico	diecinueve
Perico	diecinueve

**Ahora vamos a hacer una consulta de todos los Pericos. ¿Cuánto ha tardado? Puedes ejecutar este código varias veces**

```
spark.sql("SELECT * FROM tabla3 WHERE nombre = 'Perico' AND edad = 'diecinueve').show(1000000000)
```

```
+-----+-----+
|nombre|    edad|
+-----+-----+
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
```

**Por una parte, tenemos OPTIMIZE, un proceso costoso que reorganiza las partes en que se divide una tabla. A OPTIMIZE se le puede añadir ZORDER. En este caso, añadimos ZORDER BY a la tabla1 por el nombre**

```
spark.sql("OPTIMIZE tabla3 ZORDER BY nombre")
```

```
Out[119]: DataFrame[path: string, metrics: struct<numFilesAdded:bigint,numFilesRemoved:bigint,filesAdded:struct<min:bigint,max:bigint,avg:double,totalFiles:bigint,totalSize:bigint>,filesRemoved:struct<min:bigint,max:bigint,avg:d
```

```
ouble,totalFiles:bigint,totalSize:bigint>,partitionsOptimized:bigint,zOrderStats:struct<strategyName:string,inputCubeFiles:struct<num:bigint,size:bigint>,inputOtherFiles:struct<num:bigint,size:bigint>,inputNumCubes:bigint,mergedFiles:struct<num:bigint,size:bigint>,numOutputCubes:bigint,mergedNumCubes:bigint>,numBatches:bigint,totalConsideredFiles:bigint,totalFilesSkipped:bigint,preserveInsertionOrder:boolean>]
```

## De nuevo hacemos una consulta de todos los Pericos. ¿Cuánto ha tardado esta vez? Puedes ejecutar este código varias veces

```
spark.sql("SELECT * FROM tabla3 WHERE nombre = 'Perico' AND edad = 'diecinueve']").show(1000000000)
```

```
+-----+-----+
|nombre|      edad|
+-----+-----+
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
|Perico|diecinueve|
```

```
tabla3_df = spark.read.format("delta").load("dbfs:/user/hive/warehouse/tabla3")
tabla3_df.write.format("delta").mode('overwrite').save("dbfs:/FileStore/tabla3")
```

## 8. Delta Lake

### 8.1 Un solo almacenamiento para Batch y Streaming

Ya hemos visto cómo las tablas delta admiten lecturas y escrituras en batch. Del mismo modo, Delta Lake está integrado con Spark Streaming.

### 8.2 Change Data Feed (CDF)

#### Casos de uso

- Tablas Silver y Gold: mejora el rendimiento procesando únicamente los cambios a nivel de fila.
- Vistas actualizadas: crea vistas actualizadas para BI y analytics sin tener que reprocesar las tablas.
- Transmite los cambios: puede enviar los cambios por ejemplo a Kafka o a un RDBMS.
- Genera un registro de auditoría: puede mostrar los cambios a través del tiempo.

(<https://docs.databricks.com/delta/delta-change-data-feed.html>)

(<https://docs.databricks.com/delta/delta-change-data-feed.html>))

**Change data feed no está activado por defecto. Vamos a crear una tabla que lo tenga activado**

```
%sql
```

```
CREATE OR REPLACE TABLE perro (nombre STRING, raza STRING)
TBLPROPERTIES (delta.enableChangeDataFeed = true)
```

```
OK
```

**Es posible activarlos para tablas existentes con ALTER TABLE**

```
%sql
```

```
--ALTER TABLE tabla2 SET TBLPROPERTIES (delta.enableChangeDataFeed = true)
```



## Otra forma de hacerlo es activarlo para todas las tablas nuevas

```
%sql
--SET spark.databricks.delta.properties.defaults.enableChangeDataFeed =
true;
```

## Insertamos nuevos datos a la tabla

```
%sql
INSERT INTO perro VALUES('Rex', 'pastor alemán'), ('Dexter', 'doberman'),
('Poppy', 'caniche');
```

	num_affected_rows ▲	num_inserted_rows ▲
1	3	3

Showing all 1 rows.



## Observamos el directorio de la tabla perro

```
%fs
ls dbfs:/user/hive/warehouse/perro
```

	path
1	dbfs:/user/hive/warehouse/perro/_delta_log/
2	dbfs:/user/hive/warehouse/perro/part-00000-1458907c-e5f5-4dc5-83bf-056a6413a5fd-c000.s
3	dbfs:/user/hive/warehouse/perro/part-00001-260f5548-bb4f-4879-8d78-3ec2fd90f83c-c000.sr
4	dbfs:/user/hive/warehouse/perro/part-00002-7437bd84-73c6-4aec-8cf5-1b076259603c-c000.:

Showing all 4 rows.



## Vamos a actualizar el nombre de Rex a Regex

```
%sql
```

```
UPDATE perro SET nombre='Regex' WHERE nombre='Rex';
```

```
SELECT * FROM perro;
```

	nombre ▲	raza ▲
1	Regex	pastor alemán
2	Dexter	doberman
3	Poppy	caniche

Showing all 3 rows.



## ¿Qué ha cambiado en el directorio de la tabla perro?

```
%fs
```

```
ls dbfs:/user/hive/warehouse/perro
```

	path
1	dbfs:/user/hive/warehouse/perro/_change_data/
2	dbfs:/user/hive/warehouse/perro/_delta_log/
3	dbfs:/user/hive/warehouse/perro/part-00000-0b7e3616-ac5e-4aca-b7e1-827f03480e0c-c000.s
4	dbfs:/user/hive/warehouse/perro/part-00000-1458907c-e5f5-4dc5-83bf-056a6413a5fd-c000.s
5	dbfs:/user/hive/warehouse/perro/part-00001-260f5548-bb4f-4879-8d78-3ec2fd90f83c-c000.sr
6	dbfs:/user/hive/warehouse/perro/part-00002-7437bd84-73c6-4aec-8cf5-1b076259603c-c000.sr

Showing all 6 rows.



## Formas de ver los cambios en las tablas con CDF en batch

Más información: <https://docs.databricks.com/delta/delta-change-data-feed.html>  
(<https://docs.databricks.com/delta/delta-change-data-feed.html>)

**SQL – Escribimos el nombre de la tabla, la versión o timestamp inicial y la versión o timestamp final (opcional) entre las que queremos ver los cambios**

```
%sql
```

```
SELECT * FROM table_changes('perro', 0)
```

	nombre ▲	raza ▲	_change_type ▲	_commit_version ▲	_commit_timestamp ▲
1	Rex	pastor alemán	update_preimage	2	2021-07-21 07:39:55
2	Regex	pastor alemán	update_postimage	2	2021-07-21 07:39:55
3	Rex	pastor alemán	insert	1	2021-07-21 07:39:49
4	Dexter	doberman	insert	1	2021-07-21 07:39:49
5	Poppy	caniche	insert	1	2021-07-21 07:39:49

Showing all 5 rows.



## En python / Scala (sin '\')

```
spark.read.format("delta") \
    .option("readChangeFeed", "true") \
    .option("startingVersion", 0) \
    .option("endingVersion", 10) \
    .table("perro").show()
```

#0 bien:

#ATENCIÓN: los timestamps deben ser correctos, de lo contrario dará error

```
'''spark.read.format("delta") \
    .option("readChangeFeed", "true") \
    .option("startingTimestamp", '2021-04-21 05:45:46') \
    .option("endingTimestamp", '2021-07-19 13:38:57.0') \
    .table("perro").show()'''
```

nombre	raza	_change_type	_commit_version	_commit_timestamp
Rex	pastor alemán	update_preimage	2	2021-07-21 07:39:55
Regex	pastor alemán	update_postimage	2	2021-07-21 07:39:55
Rex	pastor alemán	insert	1	2021-07-21 07:39:49
Dexter	doberman	insert	1	2021-07-21 07:39:49
Poppy	caniche	insert	1	2021-07-21 07:39:49


```
Out[122]: 'spark.read.format("delta")    .option("readChangeFeed", "true")
.option("startingTimestamp", \'2021-04-21 05:45:46\')    .option("endingTimes
tamp", \'2021-07-19 13:38:57.0\')    .table("perro").show()'
```

## Lectura de CDF en streaming

### En python / Scala (con 'val' y sin '\')

```
sStream = spark.readStream.format("delta") \
    .option("readChangeFeed", "true") \
    .option("startingVersion", 0) \
    .table("perro")
```

```
display(sStream)
```

►  display\_query\_1 (id: af545bdf-450c-4ea2-b91f-fd8bb56b7078)

*Last updated: 18 minutes ago*

	nombre ▲	raza ▲	_change_type ▲	_commit_version ▲	_commit_timestamp ▲
1	Rex	pastor alemán	update_preimage	2	2021-07-26 15:00:00
2	Regex	pastor alemán	update_postimage	2	2021-07-26 15:00:00
3	Rex	pastor alemán	insert	1	2021-07-26 15:00:00
4	Dexter	doberman	insert	1	2021-07-26 15:00:00
5	Poppy	caniche	insert	1	2021-07-26 15:00:00
6	Doc	San Bernardo	insert	3	2021-07-26 15:00:00
7	Doc	San Bernardo	update_preimage	4	2021-07-26 15:00:00

Showing all 8 rows.



### Cuando el comando anterior esté ejecutándose, insertamos un nuevo valor

```
%sql
INSERT INTO perro VALUES("Doc", "San Bernardo")
```

	num_affected_rows ▲	num_inserted_rows ▲
1	1	1

Showing all 1 rows.



### Actualizamos el nombre del registro nuevo, de Doc a Dog

```
%sql
UPDATE perro SET nombre='Dog' WHERE raza='San Bernardo'
```

	num_affected_rows ▲	
1	1	

Showing all 1 rows.



**Observa el `display(sStream)` y asegúrate de que está ordenado por `_commit_version` de manera descendente para ver los cambios**

## Notas (a 21/07/2021)

### Cosas que no funcionan en local pero sí en databricks

- Restaurar tabla a versión X, ni SQL ni DeltaTable API.
- Usar VERSION AS OF en SQL. Sí funciona en DeltaTable API.
- Optimize / ZORDER.
- VACUUM.
- Change Data Feed - se puede poner SET TBLPROPERTIES (delta.enableChangeDataFeed = true), pero no se puede usar con delta. Y si la tabla no es delta esa propiedad es inútil.
- CLONE.

### Otros

- Databricks crea las tablas delta por defecto, sin necesidad de especificar el formato.
- No es posible hacer un vacuum únicamente al Change Data Feed, pero al hacer vacuum a una tabla también lo hace al CDF.
- Z-ORDER
  - no funciona consistentemente en el ejemplo de este notebook en cuanto a la rapidez. Lo esperado es que la consulta sea más rápida después de usar OPTIMIZE ZORDER BY, pero la realidad es que a veces sí, a veces no.
  - ZORDER BY parece modificar los minvalues y los maxvalues dentro de los archivos .json del delta\_log. Parece que ahí es donde se almacena este índice.
- DeltaTable.convertToDelta() no ha funcionado en ningún momento, ni en databricks ni en local. Por el contrario, CONVERT TO DELTA funciona sin problemas tanto en databricks como en local.