Stock data processing in Delta Lake

CSCI E-63 Big Data Analytics Spring 2021, Astakhov Danil

Problem statement

Architecture construction of an ACID-compliant storage and streaming processing of stock market data based on Databricks Delta Lake.

Problem Description

Designing the storage architecture is an essential part in big data solutions engineering. The central issue of these tasks is the choice of persistent storage. Even though Data Lake and Data Warehouse are powerful tools, the classic solutions based on them have a high complexity of application architecture. Just as difficult is the implementation of CRUD scripts in storage. Architectures based on Delta Lake are designed to eliminate these drawbacks. This paper presents the implementation of Delta Lake architecture for stock market data. The stock market is a suitable data model for data streaming, moreover, there is a wide scope for Business Intelligence.

Lakehouse architecture

"We define a Lakehouse as a data management system based on low-cost and directly-accessible storage that also provides traditional analytical DBMS management and performance features such as ACID transactions, data versioning, auditing, indexing, caching, and query optimization. Lakehouses thus combine the key benefits of data lakes and data warehouses: low-cost storage in an open format accessible by a variety of systems from the former, and powerful management and optimization features from the latter." - Armbrust Michael, Ghodsi Ali, Xin Reynold, Zaharia Matei, Databricks, UC Berkeley, Stanford University “Lakehouse: A New Generation of Open Platforms That Unify Data Warehousing and Advanced Analytics.” The Conference on Innovative Data Systems Research (CIDR), 2021, p. 3, cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf.

The idea that lays under Lakehouse is to have low-cost store with a well-defined Apache Parquet format, and add a layer with transactional metadata.

Delta lake Delta Lake is extremely popular, and "now used in about half of Databricks’ workload, by thousands of customers" as said on https://delta.io

Key features:

Stock data processing in Delta Lake

Your report will start with class name, year and semester, your name and the project title.

Problem statement

Architecture construction of an ACID-compliant storage and streaming processing of stock market data based on Databricks Delta Lake.

Problem Description

Architecture construction and building the storage are important design steps in Big Data solutions. Data Lake and Data Warehouse, while powerful tools, increase the complexity of the overall application architecture, as well as make consistent modification of data outside of the pipelines more complicated. Stock market data is a good data model for data streaming, moreover, opens a wide field for Business Intelligence.

Lakehouse architecture

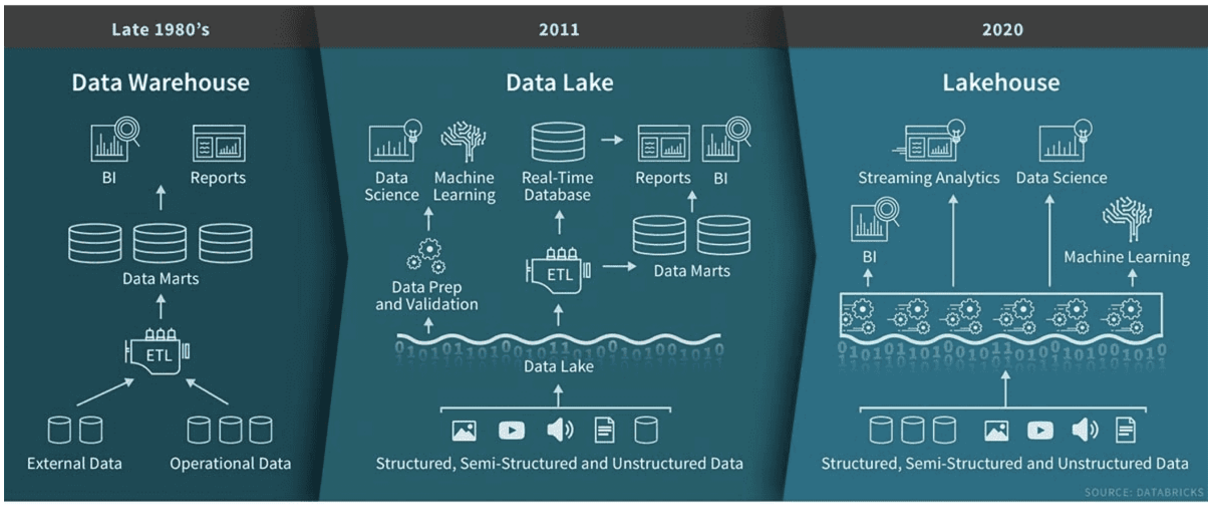
"We define a Lakehouse as a data management system based on low-cost and directly-accessible storage that also provides traditional analytical DBMS management and performance features such as ACID transactions, data versioning, auditing, indexing, caching, and query optimization. Lakehouses thus combine the key benefits of data lakes and data warehouses: low-cost storage in an open format accessible by a variety of systems from the former, and powerful management and optimization features from the latter." - Armbrust Michael, Ghodsi Ali, Xin Reynold, Zaharia Matei, Databricks, UC Berkeley, Stanford University “Lakehouse: A New Generation of Open Platforms That Unify Data Warehousing and Advanced Analytics.” The Conference on Innovative Data Systems Research (CIDR), 2021, p. 3, cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf.

The idea that lays under Lakehouse is to have low-cost store with a well-defined Apache Parquet format, and add a layer with transactional metadata.

Delta lake Delta Lake is extremely popular, and "now used in about half of Databricks’ workload, by thousands of customers" as said on https://delta.io

Key features:

- Simple integration with Business Intelligence and Machine Learning
- Structure Streaming and Batch Processing compatibility
- Transaction management
- Schema enforcement
- Data in parquet
- DataFrame API
- Metadata API
- SQL API



Delta Lake

Delta Lake is an open source storage layer that brings reliability to data lakes. Delta Lake provides ACID transactions, scalable metadata handling, and unifies streaming and batch data processing. Delta Lake runs on top of your existing data lake and is fully compatible with Apache Spark APIs.

Specifically, Delta Lake offers:

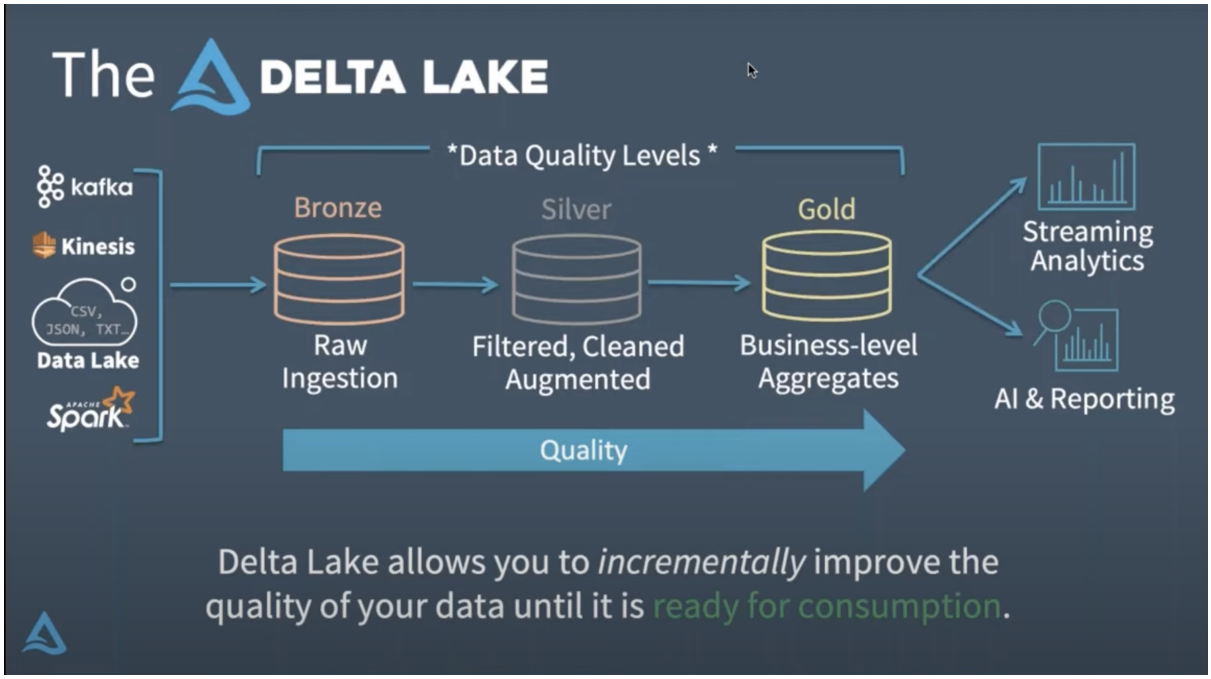
- ACID transactions on Spark: Serializable isolation levels ensure that readers never see inconsistent data.

- Scalable metadata handling: Leverages Spark's distributed processing power to handle all the metadata for petabyte-scale tables with billions of files at ease.
- Streaming and batch unification: A table in Delta Lake is a batch table as well as a streaming source and sink. Streaming data ingest, batch historic backfill, interactive queries all just work out of the box.
- Schema enforcement: Automatically handles schema variations to prevent insertion of bad records during ingestion.
- Time travel: Data versioning enables rollbacks, full historical audit trails, and reproducible machine learning experiments.
- Upserts and deletes: Supports merge, update and delete operations to enable complex use cases like change-data-capture, slowly-changing-dimension (SCD) operations, streaming upserts, and so on.

Source: "Introduction." Delta.lo, Databricks, docs.delta.io/latest/delta-intro.html. Accessed 10 May 2021.

Pipeline

According to "Making Apache Spark™ Better with Delta Lake" YouTube video (https://www.youtube.com/watch?v=LJtShrQqYZY) on channel Databricks, it's good architectural approach to organise delta lake from layers, with separation data on quality and processing logic.



Software environment

macOS Big Sur Version 11.3.1 Darwin Kernel Version 20.4.0

Apache Spark: spark-3.1.1-bin-hadoop3.2

Python 3.9.2, Clang 12.0.0 (clang-1200.0.32.29) on darwin

Python packages:

- deltalake 0.4.7
- pandas 1.2.4
- plotly 4.14.3

Installations

Apache Spark and Python were installed according to the process completed in the lab work.

deltalake installation:

```
python3 -m pip install deltalake
```

Dataset

Dataset is a CSV Kaggle dataset downloaded from `https://www.kaggle.com/camnugent/sandp500` that represents historical stock data for all current S&P 500 companies.

`wc` on `datasource/all_stocks_5yr.csv` shows 620 thousand rows:

```
➜  solution git:(master) ✗ wc datasource/all_stocks_5yr.csv
619041 619041 29580549 datasource/all_stocks_5yr.csv
```

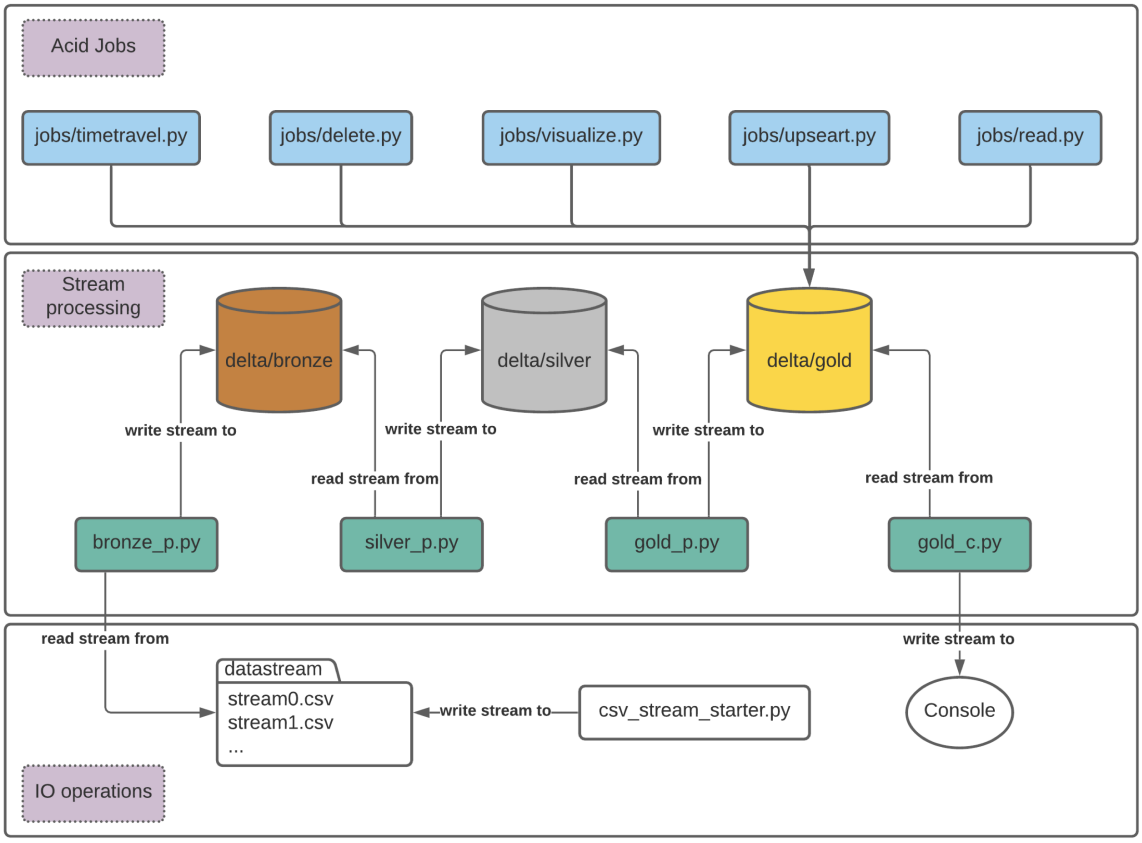
schema of CSV:

```
date,open,high,low,close,volume,Name
```

data source sample::

```
2013-02-08,15.07,15.12,14.63,14.75,8407500,AAL
```

Solution architecture



At first need to build a data producer. This is done with the stream server `csv_stream_starter.py`

csv_stream_starter.py

This module is a `csv` stream server which sorts the input dataset by date, and sequentially sends a set of lines to the stream in the `datastream` directory.

Read from original dataset implemented with small generator-like Source object:

```
class Source():
    def __init__(self, filename):
        self.pos = 0
        self.data = []

        with open(filename, 'r') as rf:
            reader = csv.reader(rf)
            self.data = sorted([row for row in reader], key=lambda row: row[0]) # sort
```

```
by date

    def read(self, strings_amount):
            new_pos = self.pos + strings_amount
            data = self.data[self.pos:new_pos]
            self.pos = new_pos
            return '\n'.join([','.join(row) for row in data])
```

Read is made with `strings_amount` parameter, which value currently is random range:

```
data = source.read(random.randint(30, 70)).strip('\n')
```

`wait_time` needed to control pauses between messages.

```
def start_csv_stream():
    idx = 0
    source = Source("datasource/all_stocks_5yr.csv")
    while True:
        produce_file(source, 'datastream', idx)
        idx += 1
        # reduce time
        wait_time = 1
        time.sleep(wait_time)
```

server started with

```
python3 csv_stream_starter.py
```

Each line indicates a successful stream writing, and the value represents length of the recorded message.

```
len(data)=531
len(data)=285
len(data)=326
len(data)=284
len(data)=670
len(data)=424
len(data)=370
```

`datastream` (stream directory) looks like:

```
➜  solution git:(master) ✗ ls datastream
source0.csv source1.csv source2.csv source3.csv source4.csv source5.csv source6.csv
➜  solution git:(master) ✗ cat datastream/source0.csv
2013-02-08,15.07,15.12,14.63,14.75,8407500,AAL
2013-02-08,67.7142,68.4014,66.8928,67.8542,158168416,AAPL
2013-02-08,78.34,79.72,78.01,78.9,1298137,AAP
2013-02-08,36.37,36.42,35.825,36.25,13858795,ABBV
2013-02-08,46.52,46.895,46.46,46.89,1232802,ABC
2013-02-08,34.39,34.66,34.29,34.41,10237828,ABT
2013-02-08,73.01,73.71,72.82,73.31,2000477,ACN
2013-02-08,38.31,39.45,38.145,39.12,5104545,ADBE
2013-02-08,44.72,45.9,44.45,45.7,2962576,ADI
2013-02-08,30.31,30.48,30.16,30.22,4789484,ADM
2013-02-08,60.71,61.0,60.51,60.925,1813162,ADP%
```

Bronze level

Bronze is the first level of the deltalake. All incoming messages go to the bronze storage. Bronze - is a level that in general is unprocessed.

In this case, writing into delta table comes from stream, but also there could be just batch and stream + batch solution.

A `delta` folder has been created. The delta folder will be a "database" and store the "tables". Each table in the delta folder consists of the regular parquet files and transaction log (`_delta_log`). Transaction log allows the ACID support to exist and solves the problem of the parquet files database growth.

```
➜  solution git:(master) ✗ mkdir -p delta/events delta/events_processed datasource
➜  solution git:(master) ✗
```

Here and in next Spark constructed with Delta extensions:

```
spark = pyspark.sql.SparkSession.builder.master('spark://127.0.0.1:7077').appName("MyA
pp") \
    .config("spark.jars.packages", "io.delta:delta-core_2.12:0.8.0") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.Del
taCatalog") \
    .config("spark.sql.streaming.forceDeleteTempCheckpointLocation", True) \
    .getOrCreate()
```

Here and in next scripts started with Delta extensions:

```
spark-submit --packages io.delta:delta-core_2.12:0.8.0,org.apache.spark:spark-sql-kafk
a-0-10_2.12:3.0.1 --conf "spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtensio
n" --conf "spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCat
alog" bronze_p.py
```

schema of incoming data is defined:

```
import pyspark.sql.types as stypes
schema = stypes.StructType().add('date', stypes.DateType()) \
    .add('open', stypes.FloatType()).add('high', stypes.FloatType()) \
    .add('low', stypes.FloatType()).add('close', stypes.FloatType()) \
    .add('volume', stypes.FloatType()).add('name', stypes.StringType())
```

csv stream read:

```
df = spark.readStream.option("host","localhost").option("port","9999") \
    .option('includeTimestamp', 'true').schema(schema).csv('datastream')
```

schema checked:

```
df.printSchema()
```

```
root
 |-- date: date (nullable = true)
 |-- open: float (nullable = true)
 |-- high: float (nullable = true)
 |-- low: float (nullable = true)
 |-- close: float (nullable = true)
 |-- volume: float (nullable = true)
 |-- name: string (nullable = true)
```

Data stream written to `delta/bronze`:

Checkpoints are needed to be able to reduce transaction log expansion.

```
query = df.writeStream.format("delta").outputMode("append") \
    .option("checkpointLocation", "checkpoints/etl-from-csv") \
    .option("mergeSchema", "true") \
    .start("delta/bronze/")
query.awaitTermination()
```

Silver level

`silver_p` is the second layer of data processing and storage. A streaming job has been created that loads data from the `bronze` table into the `silver` table. The architecture assumes that primary data processing, e.g. removing `NULL` values, can also be performed while this layer is running.

The `deltatable`, from which the stream reading is performed, can be worked with in the same way as a regular DF:

```
df = spark.readStream.format("delta").load("delta/bronze/")
```

The schema is unnecessary: the schema was set when the data was written and will now be inherited.

`NULL` values have been removed from df. For each column a `NULL` value is replaced by a value from another column (`column_pairs`). This is done in view of the further logic of the application (aggregations).

There could also be a different logic of data processing.

```
column_pairs = [('open', F.col('close')),
                ('high', F.col('low')),
                ('low', F.col('high')),
                ('close', F.col('open')),
                ('volume', 0)
                ]

for column, target in column_pairs:
    df = df.withColumn(
        column,
        F.when(
            F.isnan(column) |
            F.col(column).isNull() |
            (F.col(column) == "NA") |
            (F.col(column) == "NULL"),
            target).otherwise(F.col(column)).cast(stypes.DoubleType()))
```

Clean data written to `delta/silver`.

```
query = df.writeStream.format("delta").outputMode("append") \
    .option("checkpointLocation", "checkpoints/bronze-to-silver") \
    .option("mergeSchema", "true") \
    .start("delta/silver/")
query.awaitTermination()
```

When jobs run, `delta/silver` files are printed:

```
→  delta git:(master) ✗ ls silver/_delta_log
00000000000000000000.json               00000000000000000110.json
00000000000000000001.json               00000000000000000111.json
00000000000000000002.json               00000000000000000112.json
00000000000000000003.json               00000000000000000113.json
00000000000000000004.json               00000000000000000114.json
00000000000000000005.json               00000000000000000115.json
00000000000000000006.json               00000000000000000116.json
00000000000000000007.json               00000000000000000117.json
00000000000000000008.json               00000000000000000118.json
00000000000000000009.json               00000000000000000119.json
00000000000000000010.checkpoint.parquet 00000000000000000120.checkpoint.parquet
00000000000000000010.json               00000000000000000120.json
00000000000000000011.json               00000000000000000121.json
00000000000000000012.json               00000000000000000122.json
00000000000000000013.json               00000000000000000123.json
00000000000000000014.json               00000000000000000124.json
00000000000000000015.json               00000000000000000125.json
00000000000000000016.json               00000000000000000126.json
00000000000000000017.json               00000000000000000127.json
00000000000000000018.json               00000000000000000128.json
00000000000000000019.json               00000000000000000129.json
00000000000000000020.checkpoint.parquet 00000000000000000130.checkpoint.parquet
```

```
→  delta git:(master) ✗ cat silver/_delta_log/00000000000000000000.json
{"commitInfo":{"timestamp":1620868097267,"operation":"STREAMING UPDATE","operationParameters":{"outputMo
de":"Append","queryId":"f9497d19-946e-47f5-b9f6-6dbde13184c3","epochId":"0"},"isBlindAppend":true,"opera
tionMetrics":{"numRemovedFiles":"0","numOutputRows":"5314","numOutputBytes":"186218","numAddedFiles":"8"
}}}
{"protocol":{"minReaderVersion":1,"minWriterVersion":2}}
{"metaData":{"id":"2fc208a9-bc2b-42b7-b7b9-605130451870","format":{"provider":"parquet","options":{}},"s
chemaString":"{\"type\":\"struct\",\"fields\":[{\"name\":\"date\",\"type\":\"date\",\"nullable\":true,\"
metadata\":{}},{\"name\":\"open\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{\"name\":\"hig
h\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{\"name\":\"low\",\"type\":\"double\",\"nulla
ble\":true,\"metadata\":{}},{\"name\":\"close\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{
\"name\":\"volume\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{\"name\":\"name\",\"type\":\
"string\",\"nullable\":true,\"metadata\":{}}]}","partitionColumns":[],"configuration":{},"createdTime":1
620868095608}}
{"txn":{"appId":"f9497d19-946e-47f5-b9f6-6dbde13184c3","version":0,"lastUpdated":1620868097241}}
{"add":{"path":"part-00000-44f80cac-89ed-42a9-8a54-592267517c73-c000.snappy.parquet","partitionValues":{
},"size":55209,"modificationTime":1620868097000,"dataChange":true}}
{"add":{"path":"part-00001-9ab6f8ac-038b-4721-8217-e8ff88c753c4-c000.snappy.parquet","partitionValues":{
},"size":44856,"modificationTime":1620868097000,"dataChange":true}}
{"add":{"path":"part-00002-0bddd477-e467-4d60-adbe-86bae286f1ee-c000.snappy.parquet","partitionValues":{
},"size":34655,"modificationTime":1620868097000,"dataChange":true}}
{"add":{"path":"part-00003-c11f62fd-274b-4285-ba7e-a435458efba0-c000.snappy.parquet","partitionValues":{
},"size":20693,"modificationTime":1620868097000,"dataChange":true}}
```

Gold level

`gold_p` is the third layer of data processing and storage. A streaming job has been created that loads data from the silver table into the gold table. The architecture assumes that high level data processing - (like aggregation) can also be performed while this layer is running.

Silver Delta table read:

```
silver = spark.readStream.format("delta").load("delta/silver/")
```

To implement `output mode` = `append`, `timestamp` is added to the grouping expression. Aggregation is by `name` field, aggregated values:

- `max_range` - maximum subtraction between high and low columns
- `volume_sum` - sum of the volume column
- `delta_total_percents` - how many percent change in open

```
import pyspark.sql.functions as F

query = silver.withColumn('timestamp', F.unix_timestamp(F.col('date'), "yyyy-MM-dd").c
ast(stypes.TimestampType())) \
    .withWatermark("timestamp", "1 minutes") \
    .select('*').groupby(silver.name, "timestamp").agg(
    F.max(F.col('high') - F.col('low')).alias('max_range'),
    F.sum(silver.volume).alias('volume_sum'),
    F.sum((F.col('open') - F.col('close')) / F.col('open') * 100).alias('delta_total_p
ercents')
)
```

```
+----+-------------------+-------------------+-----------+--------------------+
|name|          timestamp|          max_range| volume_sum|delta_total_percents|
+----+-------------------+-------------------+-----------+--------------------+
| NEE|2013-02-12 00:00:00|  0.529998779296875|  1779632.0| -0.4990997144873778|
| GLW|2013-02-11 00:00:00|0.20999908447265625|1.0352591E7| -0.9740250692497617|
|  GD|2013-02-12 00:00:00|    0.4949951171875|  1950788.0|  0.3735245762140945|
| CCI|2013-02-15 00:00:00|   1.58489990234375|  4525800.0| -1.8296358955001792|
|  PH|2013-02-08 00:00:00| 0.7900009155273438|   566484.0| -0.5222187511535973|
|KLAC|2013-02-14 00:00:00| 0.6599998474121094|  2182742.0|-0.16068586910364147|
| CVS|2013-02-13 00:00:00|              0.375|  4837994.0| 0.13690532634835145|
|  PH|2013-02-14 00:00:00|                1.5|  1044842.0| -0.5638518684927185|
| KMI|2013-02-14 00:00:00|0.40000152587890625|  2910025.0|  0.1329060034028074|
| AYI|2013-02-13 00:00:00| 1.2600021362304688|   251358.0| -1.5125535543636806|
|  HP|2013-02-12 00:00:00| 0.9200057983398438|   961950.0| -0.4144230801885063|
| LLY|2013-02-11 00:00:00| 0.5800018310546875|1.8307288E7| -0.6348023597277948|
| KSU|2013-02-08 00:00:00|     1.989501953125|   640499.0| 0.47101169491781236|
| NEE|2013-02-14 00:00:00| 0.7957000732421875|  1779659.0| 0.19352971625063228|
|  HD|2013-02-13 00:00:00|0.48999786376953125|  3668687.0| 0.10367253895799323|
|EQIX|2013-02-11 00:00:00|                2.5|   447442.0|-0.11938949429906232|
|   L|2013-02-13 00:00:00| 0.3899993896484375|   663122.0| -0.7542898995535714|
| KEY|2013-02-08 00:00:00|0.12000083923339844|  7594244.0| -0.7368388928865132|
|   D|2013-02-12 00:00:00|  0.470001220703125|  1564869.0| -0.7190254011427634|
| MYL|2013-02-11 00:00:00|0.20000076293945312|  2686982.0| -0.2432233952541217|
+----+-------------------+-------------------+-----------+--------------------+
```

This query is written to `gold` level.

```
query = query.writeStream.format("delta").outputMode("append") \
    .option("checkpointLocation", "checkpoints/etl-third-to-fourth") \
```

```
    .option("mergeSchema", "true") \
    .start("delta/gold/")
query.awaitTermination()
```

Gold level console consumer

`gold_c` is the part of the pipeline that streams data from the `gold` level to the console.

```
df = spark.readStream.format("delta").load("delta/gold/")
query = df.writeStream.outputMode('update') \
    .format('console').option('truncate', 'false').start()
```

Streaming output:

```
-------------------------------------------
Batch: 58
-------------------------------------------
+----+-------------------+-------------------+------------+--------------------+
|name|timestamp          |max_range          |volume_sum  |delta_total_percents|
+----+-------------------+-------------------+------------+--------------------+
|KIM |2013-06-07 00:00:00|0.5                |2660709.0   |-0.1810729194314875 |
|STZ |2013-06-07 00:00:00|0.9010009765625    |2679510.0   |-1.4261266359077838 |
|SO  |2013-06-07 00:00:00|0.6800994873046875 |6141049.0   |0.2693059759508     |
|RE  |2013-06-07 00:00:00|1.8300018310546875 |376024.0    |0.1562598262209997  |
|IT  |2013-06-07 00:00:00|1.0499992370605469 |351507.0    |-0.27826060419497284|
|BAC |2013-06-07 00:00:00|0.24000072479248047|1.20915168E8|-0.3750952064594014 |
|GT  |2013-06-07 00:00:00|0.25999927520751953|2675855.0   |0.2020248207060221  |
|STX |2013-06-07 00:00:00|0.8299980163574219 |3291695.0   |0.856051077585392   |
|HUM |2013-06-07 00:00:00|1.3700027465820312 |902021.0    |-0.7210270811250076 |
|CI  |2013-06-07 00:00:00|1.4799957275390625 |1405529.0   |-1.4074028862847223 |
|SYY |2013-06-07 00:00:00|0.3600006103515625 |3953825.0   |-0.5959384387908775 |
|BF.B|2013-06-07 00:00:00|0.6550025939941406 |856138.0    |-0.18634926811513602|
|KMI |2013-06-07 00:00:00|0.9500007629394531 |2961857.0   |-0.9446075156454211 |
|WEC |2013-06-07 00:00:00|0.6300010681152344 |837179.0    |-0.6341422476419588 |
|CVS |2013-06-07 00:00:00|0.7100028991699219 |3914124.0   |0.0                 |
|FISV|2013-06-07 00:00:00|0.470001220703125  |1070018.0   |-0.4878027600332078 |
|SPG |2013-06-07 00:00:00|3.44000244140625   |1348037.0   |0.13732818603168095 |
|CRM |2013-06-07 00:00:00|1.3699989318847656 |1.0148789E7 |-3.204792881723416  |
|GLW |2013-06-07 00:00:00|0.3099994659423828 |1.501403E7  |-1.8543028205138246 |
|EA  |2013-06-07 00:00:00|0.43000030517578125|3476129.0   |-0.4680877036236702 |
+----+-------------------+-------------------+------------+--------------------+
only showing top 20 rows

-------------------------------------------
Batch: 59
-------------------------------------------
+-----+-------------------+-------------------+-----------+--------------------+
|name |timestamp          |max_range          |volume_sum |delta_total_percents|
+-----+-------------------+-------------------+-----------+--------------------+
|CSX  |2013-06-10 00:00:00|0.279998779296875  |6823939.0  |0.31796472254066116 |
```

```
|GILD |2013-06-10 00:00:00|1.4500007629394531 |8744157.0  |2.0728303009635902  |
|STZ  |2013-06-10 00:00:00|0.9500007629394531 |1377841.0  |1.2715039472630223  |
|BLL  |2013-06-10 00:00:00|0.1849994659423828 |908044.0   |0.5040119427548982  |
|DOV  |2013-06-10 00:00:00|1.28900146484375   |920249.0   |1.1665834055886366  |
|ESS  |2013-06-10 00:00:00|2.9399871826171875 |298738.0   |0.08326707150882279 |
|PM   |2013-06-10 00:00:00|1.3199996948242188 |3888447.0  |0.6828580839205263  |
|ROP  |2013-06-10 00:00:00|1.0                |301264.0   |0.37817929298324066 |
|APC  |2013-06-10 00:00:00|1.0999984741210938 |1886003.0  |0.04539368393164887 |
|YUM  |2013-06-10 00:00:00|1.69000244140625   |3975062.0  |1.2670303621831502  |
|KMX  |2013-06-10 00:00:00|0.7462005615234375 |1458434.0  |0.2330521368082962  |
|AET  |2013-06-10 00:00:00|0.8987998962402344 |2197540.0  |-1.1446623347365277 |
|ANSS |2013-06-10 00:00:00|1.410003662109375  |259039.0   |1.1772538507263797  |
|PFE  |2013-06-10 00:00:00|0.31999969482421875|6.6492012E7|0.03523016853018487 |
|WY   |2013-06-10 00:00:00|0.7399997711181641 |5194843.0  |1.3559309102721133  |
|VMC  |2013-06-10 00:00:00|1.9799995422363281 |501905.0   |-2.022896045280164  |
|DISCA|2013-06-10 00:00:00|2.2099990844726562 |1497131.0  |2.5940040736747814  |
|XLNX |2013-06-10 00:00:00|0.6399993896484375 |3002325.0  |0.7132339316666315  |
|ETR  |2013-06-10 00:00:00|1.1299972534179688 |1607675.0  |-0.40409694847345495|
|IDXX |2013-06-10 00:00:00|0.6899986267089844 |744852.0   |0.17828824602544624 |
+-----+-------------------+-------------------+-----------+--------------------+
```

Gold jobs

Because deltalake has the power of ACID transactions, additional queries to the database can be made while the pipelines are running, without having to worry about consistency of data.

Delete

This job removes all values from the gold table for which `delta_total_percents` > `2`.

Gold DeltaTable loaded:

```
gold = DeltaTable.forPath(spark, "../delta/gold/")
```

delete query:

```
gold.delete(F.abs(F.col("delta_total_percents")) > 2)   # predicate using Spark
```

checked using toDF()

```
gold.toDF().select('*').where(F.col("delta_total_percents") > 1.7).show()
```

Read data before job run:

```
+----+-------------------+------------------+-----------+--------------------+
|name|          timestamp|         max_range| volume_sum|delta_total_percents|
+----+-------------------+------------------+-----------+--------------------+
| TWX|2013-02-25 00:00:00|1.7299995422363281|  6431525.0|  2.5660388874557785|
|ISRG|2013-02-28 00:00:00|28.169998168945312|  2746851.0|  11.082234764221248|
| CVX|2013-02-25 00:00:00| 3.839996337890625|  7522969.0|  2.5407717463284603|
| ECL|2013-02-25 00:00:00|2.5699996948242188|  1443306.0|   2.933329264322917|
|INCY|2013-02-20 00:00:00|0.7749996185302734|  1378199.0|  3.1078538569894696|
| WFC|2013-02-25 00:00:00|1.1800003051757812|  2.70318E7|   2.929685358789625|
| NSC|2013-02-25 00:00:00|2.0699996948242188|  2690751.0|   2.603948292362565|
| GPC|2013-02-19 00:00:00|2.6200027465820312|  2308667.0|   3.046119800531852|
|REGN|2013-02-19 00:00:00|6.8899993896484375|  1018693.0|  2.4624569962294176|
| APC|2013-02-25 00:00:00|3.9300003051757812|  3865033.0|  4.5089926343146205|
| DHI|2013-02-20 00:00:00|1.3299999237060547|1.2674507E7|   4.569187371525791|
| UNP|2013-02-20 00:00:00| 1.660003662109375|  7028462.0|  2.2067315674740113|
| HUM|2013-02-22 00:00:00|2.6700057983398438|  3795289.0|   2.794601867227108|
|  CF|2013-02-20 00:00:00| 2.128002166748047| 1.330919E7|   4.549642761183847|
| CNP|2013-02-25 00:00:00|0.6499996185302734|  3592201.0|  2.0057309722167442|
| MAS|2013-02-20 00:00:00|              1.25|  8109855.0|   5.784706574836333|
|  MS|2013-02-20 00:00:00|1.1199989318847656|1.7434108E7|   3.937650194908662|
| PVH|2013-02-20 00:00:00|3.8899993896484375|   956101.0|   2.703814026360912|
| ROP|2013-02-25 00:00:00| 4.055000305175781|   470345.0|  2.8122703112690335|
|KLAC|2013-02-20 00:00:00|1.6599998474121094|  1675158.0|   2.780236693617465|
+----+-------------------+------------------+-----------+--------------------+
only showing top 20 rows
```

Output:

```
+----+-------------------+------------------+-----------+--------------------+
|name|          timestamp|         max_range| volume_sum|delta_total_percents|
+----+-------------------+------------------+-----------+--------------------+
|  GS|2013-02-21 00:00:00| 4.149993896484375|  8318209.0|   1.847512565273212|
|  GT|2013-03-04 00:00:00|0.3399991989135742|  2881577.0|   1.934984508697554|
| TPR|2013-02-20 00:00:00|               1.5|  4446174.0|  1.8155842132544933|
| JEC|2013-02-20 00:00:00|1.0999984741210938|  1143897.0|   1.887168772466949|
| HRB|2013-02-25 00:00:00|0.5599994659423828|  3962992.0|  1.7408918700155076|
| FRT|2013-02-25 00:00:00|2.2300033569335938|   420451.0|  1.7129615501121238|
| CAT|2013-02-20 00:00:00|2.0499954223632812|  8571002.0|   1.873682925575658|
| ALK|2013-02-20 00:00:00|0.7549991607666016|  1035072.0|  1.9119679386123194|
| SLB|2013-02-20 00:00:00|1.7299957275390625|  6395447.0|  1.9233182482788127|
| NOV|2013-02-15 00:00:00| 1.970001220703125|  4415727.0|  1.9516295733074123|
| STI|2013-02-13 00:00:00|0.6750011444091797|  4857578.0|  1.7651003617617635|
| NEM|2013-03-13 00:00:00|0.9199981689453125|  7470774.0|  1.9612742527505005|
| MHK|2013-02-21 00:00:00| 4.040000915527344|  1759249.0|  1.7211547264685996|
| TXT|2013-02-21 00:00:00|0.8299999237060547|  3385823.0|  1.8543986949334155|
| AIZ|2013-02-20 00:00:00|0.7900009155273438|  1058179.0|  1.7606421252956095|
| HIG|2013-02-20 00:00:00|0.5500011444091797|  6051037.0|  1.9758055894853404|
|  MA|2013-02-20 00:00:00|1.1300010681152344|  5376220.0|  1.7413800449096555|
|SWKS|2013-02-15 00:00:00|0.6100006103515625|  2876718.0|  1.8036102171502972|
| ADM|2013-02-20 00:00:00|0.6599998474121094|  7775251.0|    1.78301646983144|
| USB|2013-03-05 00:00:00|1.1499977111816406|1.7875032E7|  1.8292603906476115|
+----+-------------------+------------------+-----------+--------------------+
only showing top 20 rows
```

The ability to remove from the table without fear opens up possibilities - such as convenient GDPR compliance.

Read job

The read job is set up to print gold layer values for which `delta_total_percents` > `2`;

```
df.select('*').where(F.col("delta_total_percents") > dtp_threshold).show()
show_df(gold.toDF(), 2)
```

Job started after completion of deletion:

```
+----+---------+---------+----------+--------------------+
|name|timestamp|max_range|volume_sum|delta_total_percents|
+----+---------+---------+----------+--------------------+
+----+---------+---------+----------+--------------------+
```

Upseart job

Data can be added or changed consistently in deltalake. Add data from upseart.csv dataset to gold table.

`upseart.csv` contains two entries.

`NFPK` name is not in the `gold` data, rows for the `SWKS` name exists.

```
name,timestamp,max_range,volume_sum,delta_total_percents
NFPK,2013-02-15 00:00:00,1.1206103515625,3875718.0,3.1
SWKS,2013-02-15 00:00:00,0.6100006103515625,2876718.0,2.80361
```

`upseart.csv` is loaded as `DF`:

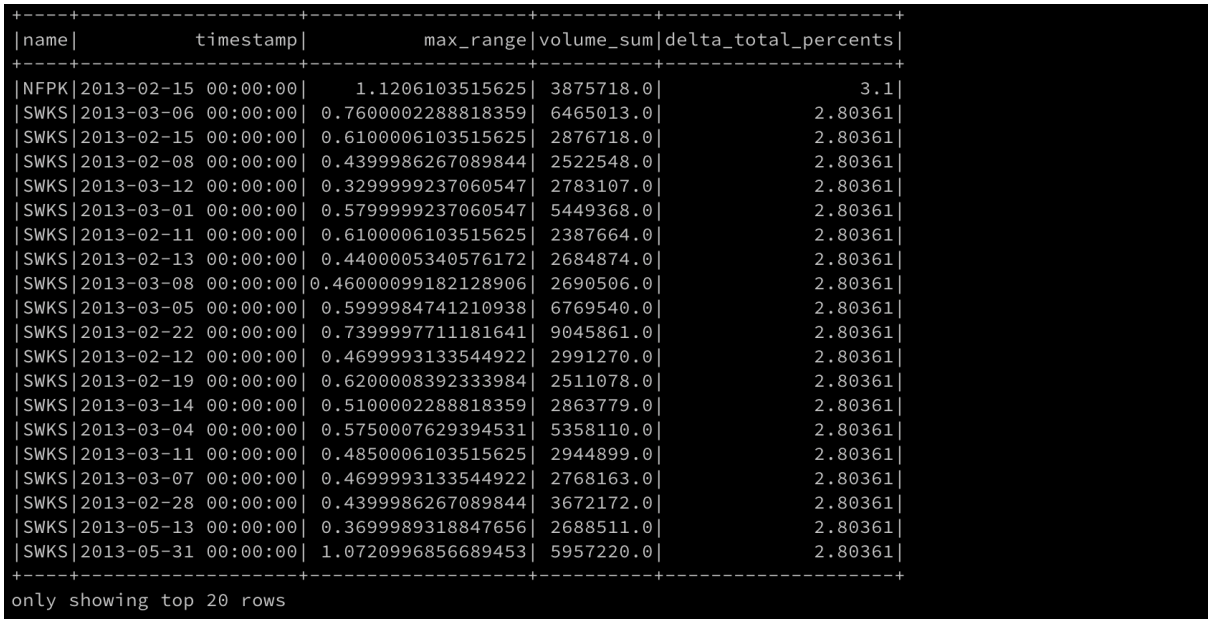
```
upseart_data = spark.read.format("csv").option("header", "true").load("../datasource/u
pseart.csv")
```

Upseart is performed on the `name` field - as a result, an additional record is created with the name `NFPK` and `delta_total_percents` value is changed in all `SWKS` records.

```
gold.alias("gold").merge(
    upseart_data.alias("upseart"),
    "gold.name = upseart.name") \
  .whenMatchedUpdate(set = { "delta_total_percents" : "upseart.delta_total_percents" }
) \
  .whenNotMatchedInsert(values =
    {
        "name": "upseart.name",
        "timestamp": "upseart.timestamp",
        "max_range": "upseart.max_range",
        "volume_sum": "upseart.volume_sum",
        "delta_total_percents": "upseart.delta_total_percents",
    }
  ) \
  .execute()
```

Execution checked:

```
gold.toDF().select('*').where(F.col("delta_total_percents") > 2).show()
```



```
+----+-------------------+-------------------+----------+-------------------+
|name|          timestamp|          max_range|volume_sum|delta_total_percents|
+----+-------------------+-------------------+----------+-------------------+
|NFPK|2013-02-15 00:00:00|    1.1206103515625| 3875718.0|                3.1|
|SWKS|2013-03-06 00:00:00| 0.7600002288818359| 6465013.0|            2.80361|
|SWKS|2013-02-15 00:00:00| 0.6100006103515625| 2876718.0|            2.80361|
|SWKS|2013-02-08 00:00:00| 0.4399986267089844| 2522548.0|            2.80361|
|SWKS|2013-03-12 00:00:00| 0.3299999237060547| 2783107.0|            2.80361|
|SWKS|2013-03-01 00:00:00| 0.5799999237060547| 5449368.0|            2.80361|
|SWKS|2013-02-11 00:00:00| 0.6100006103515625| 2387664.0|            2.80361|
|SWKS|2013-02-13 00:00:00| 0.4400005340576172| 2684874.0|            2.80361|
|SWKS|2013-03-08 00:00:00|0.46000099182128906| 2690506.0|            2.80361|
|SWKS|2013-03-05 00:00:00| 0.5999984741210938| 6769540.0|            2.80361|
|SWKS|2013-02-22 00:00:00| 0.7399997711181641| 9045861.0|            2.80361|
|SWKS|2013-02-12 00:00:00| 0.4699993133544922| 2991270.0|            2.80361|
|SWKS|2013-02-19 00:00:00| 0.6200008392333984| 2511078.0|            2.80361|
|SWKS|2013-03-14 00:00:00| 0.5100002288818359| 2863779.0|            2.80361|
|SWKS|2013-03-04 00:00:00| 0.5750007629394531| 5358110.0|            2.80361|
|SWKS|2013-03-11 00:00:00| 0.4850006103515625| 2944899.0|            2.80361|
|SWKS|2013-03-07 00:00:00| 0.4699993133544922| 2768163.0|            2.80361|
|SWKS|2013-02-28 00:00:00| 0.4399986267089844| 3672172.0|            2.80361|
|SWKS|2013-05-13 00:00:00| 0.3699989318847656| 2688511.0|            2.80361|
|SWKS|2013-05-31 00:00:00| 1.0720996856689453| 5957220.0|            2.80361|
+----+-------------------+-------------------+----------+-------------------+
only showing top 20 rows
```

Timetravel job

`_delta_log` allows a database version from the past to be loaded as `DF`. Loading is possible either by time or by version.

Because the metadata in deltalake is also big data, it can be interacted with via the `DF` interface.

History of transactions printed:

```
gold = DeltaTable.forPath(spark, "delta/gold/")
gold_history = gold.history()
print('gold history:')
gold_history.show()
```

```
+-------+-------------------+------+--------+---------+--------------------+----+--------+--------
|version|          timestamp|userId|userName|operation| operationParameters| job|notebook|cluster
+-------+-------------------+------+--------+---------+--------------------+----+--------+--------
|     74|2021-05-13 04:37:56|  null|    null|    MERGE|[predicate -> (go...|null|    null|     nu
|     73|2021-05-13 04:35:10|  null|    null|   DELETE|[predicate -> ["(...|null|    null|     nu
|     72|2021-05-13 04:20:16|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     71|2021-05-13 04:20:10|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     70|2021-05-13 04:20:02|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     69|2021-05-13 04:19:54|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     68|2021-05-13 04:19:46|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     67|2021-05-13 04:19:39|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     66|2021-05-13 04:19:33|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     65|2021-05-13 04:19:26|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     64|2021-05-13 04:19:19|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     63|2021-05-13 04:19:13|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     62|2021-05-13 04:19:05|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     61|2021-05-13 04:18:59|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     60|2021-05-13 04:18:52|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     59|2021-05-13 04:18:44|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     58|2021-05-13 04:18:35|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     57|2021-05-13 04:18:28|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     56|2021-05-13 04:18:21|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
|     55|2021-05-13 04:18:14|  null|    null|STREAMING|[outputMode -> Ap...|null|    null|     nu
+-------+-------------------+------+--------+---------+--------------------+----+--------+--------
only showing top 20 rows
```

Version without `upseart` operation chosen.

```
version = 73
```

Time travel initiated:

```
gold_timetravel = spark.read.format("delta").option("versionAsOf", version).load("delt
a/gold")
```

```
print(f"Chosen {version=}")
gold_timetravel.select('*').where(F.col('version') == version).show()
```

```
version=73
Chosen version=73
+-------+-------------------+------+--------+---------+--------------------+----+--------+---------+-----
|version|          timestamp|userId|userName|operation| operationParameters| job|notebook|clusterId|read
+-------+-------------------+------+--------+---------+--------------------+----+--------+---------+-----
|     73|2021-05-13 04:35:10|  null|    null|   DELETE|[predicate -> ["(...|null|    null|     null|
+-------+-------------------+------+--------+---------+--------------------+----+--------+---------+-----
```

```
print(f"Data for that version:")
gold_timetravel.show()
```

```
Data for that version:
+----+-------------------+-------------------+-----------+--------------------+
|name|          timestamp|          max_range| volume_sum|delta_total_percents|
+----+-------------------+-------------------+-----------+--------------------+
| AFL|2013-02-13 00:00:00| 1.0499992370605469|  6063028.0|  1.5779865961586932|
| SJM|2013-03-04 00:00:00| 1.1399993896484375|   391094.0| -0.5619156338204953|
| MTD|2013-02-08 00:00:00|  3.429901123046875|   166815.0| -0.8068141919393976|
|DISH|2013-02-08 00:00:00| 0.6399993896484375|  1449533.0| -1.1827919830241873|
| LLL|2013-03-06 00:00:00| 1.3899993896484375|   626860.0|  0.8958645054340371|
| WMT|2013-03-01 00:00:00| 1.1200027465820312|  8902516.0| -1.3563140732258048|
| CBG|2013-02-14 00:00:00| 0.5100002288818359|  3224715.0| -1.4373732073370962|
|  RF|2013-02-21 00:00:00|0.21000003814697266| 2.397351E7|  1.1718708041976822|
| MAR|2013-02-14 00:00:00|0.43000030517578125|  1381230.0| -0.5865143919995018|
| TSN|2013-02-14 00:00:00|0.45999908447265625|  4730889.0|  -1.508795866697278|
|  MU|2013-03-08 00:00:00|0.31999969482421875|3.4767488E7|  -1.882614411086026|
| HES|2013-02-08 00:00:00|  0.9899978637695312|  2760327.0| -1.0196435451823962|
| AVY|2013-03-05 00:00:00| 0.5900001525878906|   911836.0| -1.1127215354977695|
| JEC|2013-02-22 00:00:00| 0.5699996948242188|   711554.0|-0.47063545582823235|
| MTD|2013-02-11 00:00:00|   3.92999267578125|   165521.0|   0.742047613415832|
|PCAR|2013-02-28 00:00:00| 0.6300010681152344|  1543304.0|0.021066838866573526|
| WDC|2013-02-14 00:00:00| 0.8600006103515625|  2032534.0| -0.5148270035624527|
| MOS|2013-02-11 00:00:00| 0.5600013732910156|  1249907.0|  0.4072324523302649|
| AES|2013-02-14 00:00:00| 0.1700000762939453|  4892310.0|  0.8841766380156829|
|ROST|2013-03-06 00:00:00|0.37000083923339844|  4865492.0| -0.7935188660063585|
+----+-------------------+-------------------+-----------+--------------------+
only showing top 20 rows
```

`delta_total_percents` by threshold 2 printed:

```
print("DTP threshold 2 for that version:")
from read import show_df
show_df(gold_timetravel, 2)
```

```
DTP threshold 2 for that version:
+----+---------+---------+----------+--------------------+
|name|timestamp|max_range|volume_sum|delta_total_percents|
+----+---------+---------+----------+--------------------+
+----+---------+---------+----------+--------------------+
```

As can seen, upseart was cancelled.

Visualize job

Also, based on reading from the database, the data could be visualised. There could also be a machine learning mechanism.

The names for which plots will be plotted are selected:

```
names = ['WYNN', 'BLL', 'IT', 'BA']
```

The dependency of the delta_total_percents parameter on time is plotted.

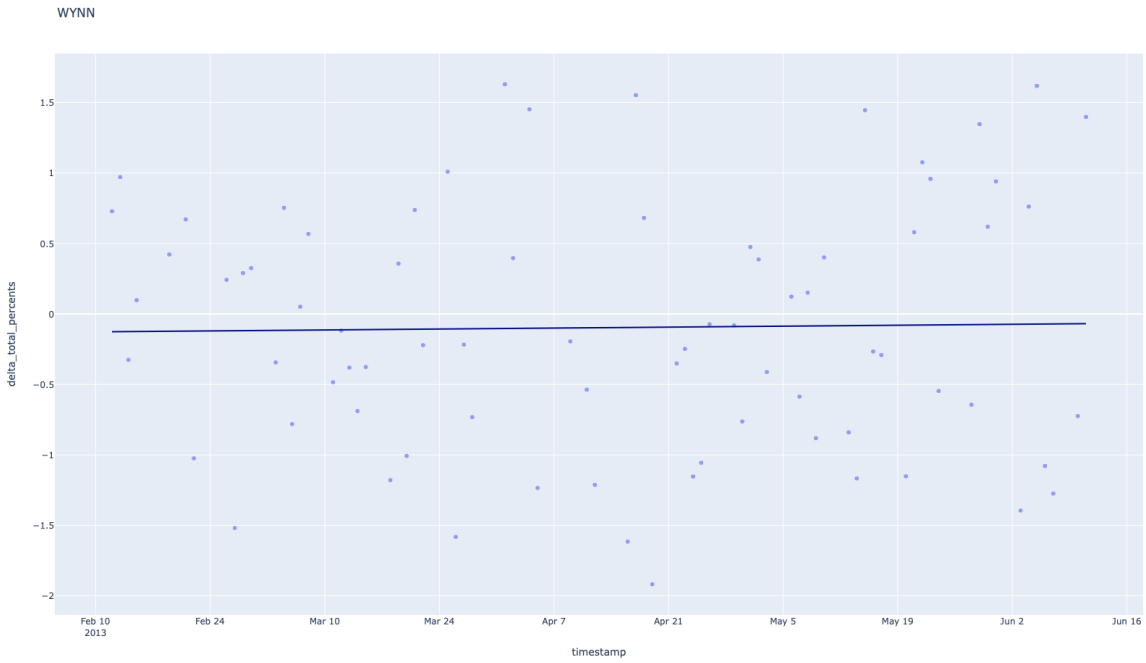
```
for name in names:
    plot_data = gold_df.select(F.col('timestamp'), F.col('delta_total_percents')).wher
```

```
e(F.col('name') == name)
    print(f'{name=}:')
    plot_data.show()

    import plotly.express as px

    fig = px.scatter(
        plot_data.toPandas(), x='timestamp', y='delta_total_percents', opacity=0.65,
        title=name,trendline='ols', trendline_color_override='darkblue'
    )
    fig.show()
```

Plot for `WYNN` name:



Summary

In this study, architecture construction of an ACID-compliant storage and streaming processing of stock market data based on Databricks Delta Lake was made.

Delta Lake is a new chapter in the development of Big Data solutions. Combination of convenient DataFrame interface, storage organisation transparency, ACID-compatibility, batch-stream processing, metadata as DataFrame opens up opportunities for building extremely large, scalable and durable storages and Big Data processing systems in financial and other fields.

Youtube

Github

Brief Demo https://youtu.be/DfgN8cHtpLg

Full Demo https://youtu.be/4nye3nRV1R4

Github: https://github.com/vo0xr0c/deltalake