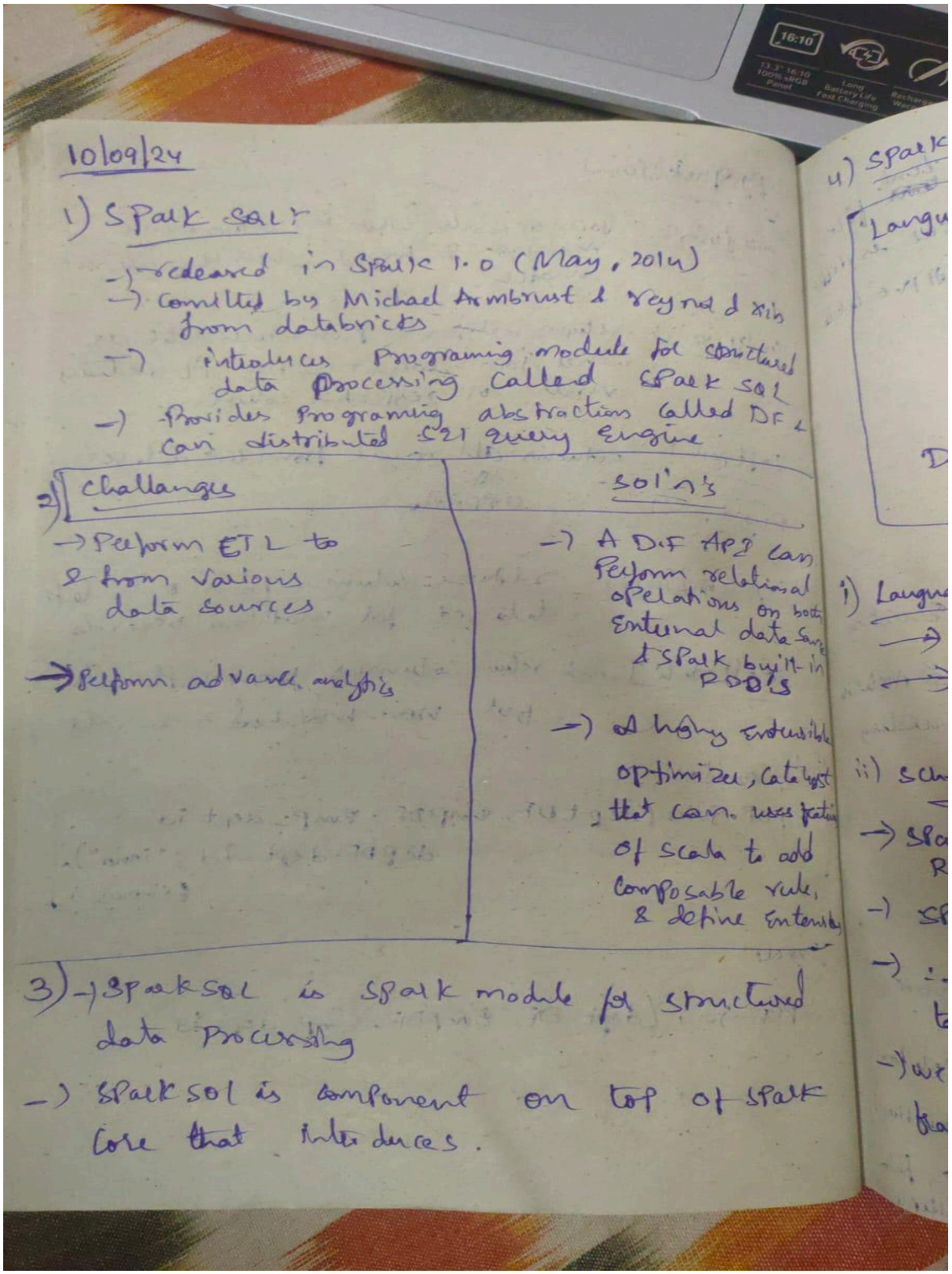
NAME : AKULA SHARATH CHANDRA
BATCH: DATA ENGINEERING
DATE : 10-02-2024
TOPICS : CONCEPTS OF SPARK SQL AND HANDS ON OF SPARK SQL

CONCEPTS OF SPARK SQL:



10/09/24

1) Spark SQL?

- -) declared in Spark 1.0 (May, 2014)
- -) Committed by Michael Armbrust & Reynold Xin from databricks
- -) introduces Programing module for structured data Processing called Spark SQL
- -) Provides Programing abstractions called DF & can distributed SQL query Engine

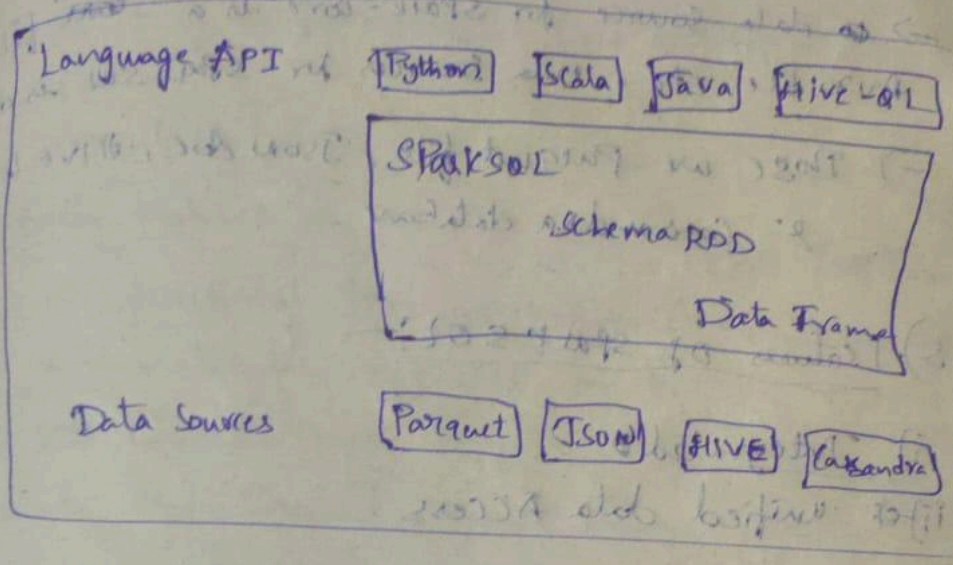
2)

Challanges	sol'n's
-> Perform ETL to & from various data sources -> Perform advanced analytics	-) A DF API can Perform relational operations on both External data sources & Spark built-in RDD's -) A highly extensible optimizer, Catalyst that can use features of Scala to add Composable rules, & define Extensible

3) -) SparkSQL is Spark module for structured data Processing

-) SparkSQL is component on top of Spark Core that introduces.

4) Spark SQL Architecture :-



i) Language API :-

→ Spark is compatible with diff lang & SparkSQL

→ it also supported by python, scala, Java, Hive-QL

ii) Schema RDD :-

→ Spark core designed with special data structure called RDD

→ SparkSQL works on schemas, tables & records

→ ∴ we can use Schema RDD as temporary table

→ we can call this schema RDD as data frame.

ii) Data Sources :-

→ Data source for Spark-core is a text file, Avro file etc. Data sources for Spark SQL is different.

→ Those are Parquet file, JSON doc, Hive table & Cassandra database.

5) Features of Spark SQL :-

i) Integrated

ii) unified data Access.

iii) Hive compatibility :-

→ run unmodified hive queries on existing warehouses

→ Spark SQL reuses the hive frontend & metastore giving you full compatibility with existing hive data, queries & UDF's.

→ simply install it along

iv) Standard connectivity (drivers)

v) Scalability

6) User Defined functions :-

Plug in your own processing code & invoke it from a hive query

— UDF (Plain UDF) i/p - single row, o/p - single row

— UDAF (User defined aggregate function)

— UDTF (User defined table-generating function) i/p - single row o/p - multiple rows

Spark RDD's

- it is a fundamental data structure of spark.
- immutable distributed collections of objects.
- each dataset is divided into logical partitions
- I/P functional transformations (map, filter, ...)
- Automatically rebuilt on failure.
- contains any types of Python, Java or Scala obj, including user-defined classes.
- RDD is read only, Partitioned collection of records
- RDD can created through deterministic operations on either data on stable storage or other RDDs
- It is fault-tolerant collection of elements that can be operated on in I/P.
- 2 ways to create RDD
  1) Parallelizing an existing collection
  2) referencing a dataset in External Storage Systems.

8) Dataset & Dataframe:-

-) A distributed collection of data, which organized into named columns.

-> It is Equivalent to relational tables with good optimization techniques.

-) A DF can be constructed from array of diff source such as Hive tables, Data files, External databases Etc.

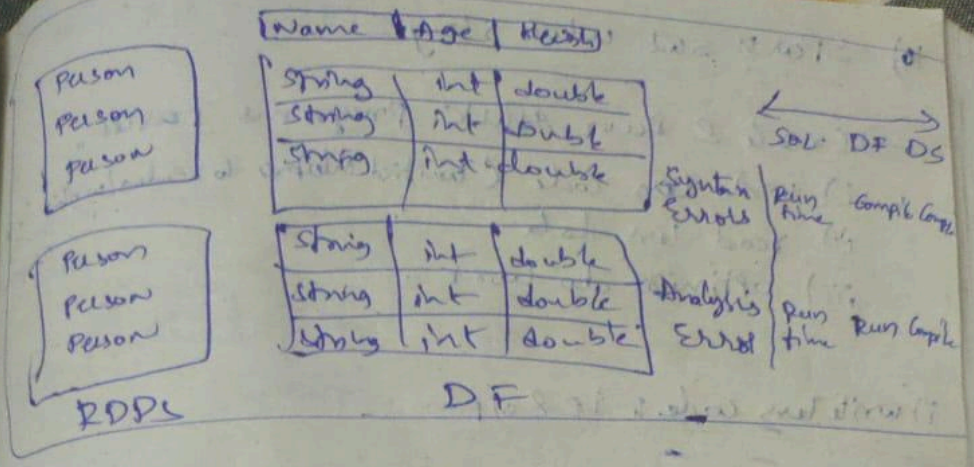
-) This API was designed for modern bigdata & data science applications taking inspiration from dataframe in R Programming & Pandas in Python.

-) Dataframe:-

- Data is organized into named colln like table in Relational database.

-) Dataset:- a distributed collection of d

- Static-typing & Runtime type-Sa
- new interface added in Spark 1.6



Features of DF :-

→ Process data in size of kb to Pb.

→ Supports diff. data formats & storage systems.

→ State of art optimization & code generation

→ Can be easily integrated with all Big data tools & frameworks via Spark-Core

→ Provides API's for Python, Java, Scala & R Programming

10) Spark SQL whole story

→ Create & Run spark programs faster:

i) write less code & unified interface to reading/writing data in variety of formats

ii) read less data.

iii) optimizer do work.

i) write less code: ip & o/p

→ unified interface to reading/writing data in variety formats

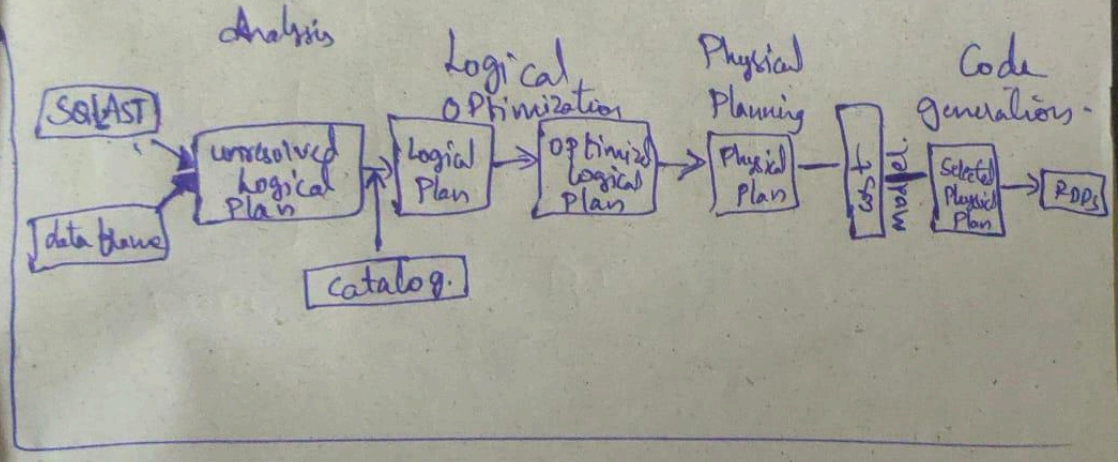
```
df = sql(Content.read)
  .format("json")
  .option("samplingRatio", 0.1)
  .load("Path")
df.write
  .format("parquet")
  .mode("append")
  .partitionBy("year")
  .saveAsTable("fasterTable")
```

Builder methods

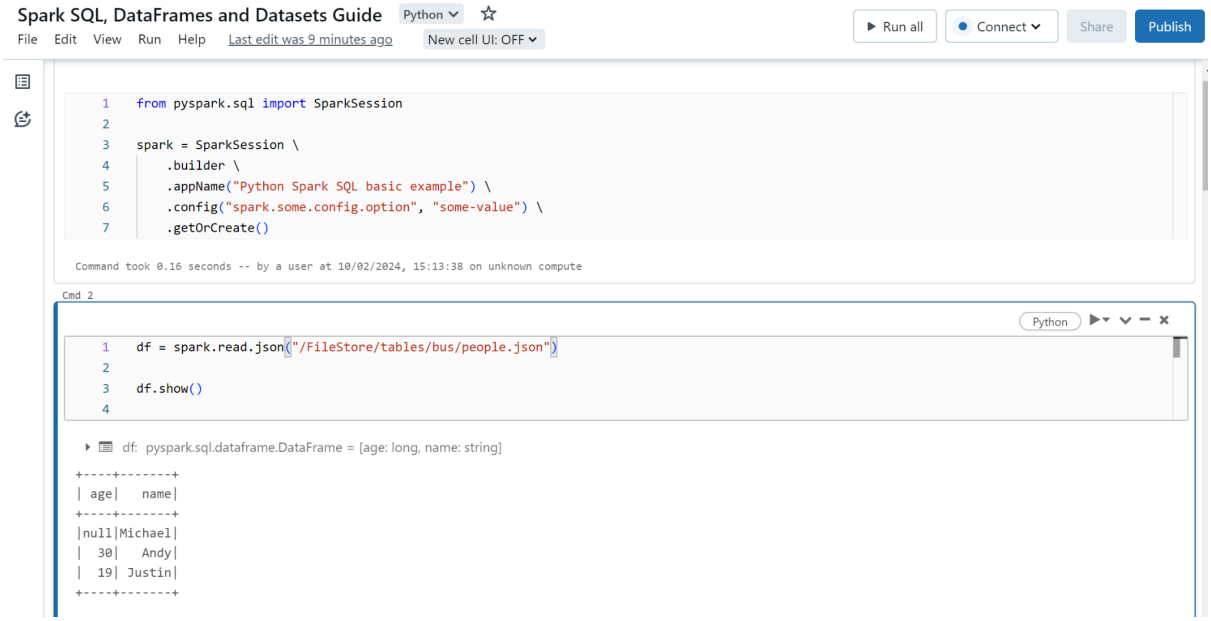
load(...), save(...) & saveAsTable(...)

Create new builders for doing I/O.

11) Plan Optimization & Execution :-



HANDS ON :


Spark SQL, DataFrames and Datasets Guide
Python
File Edit View Run Help Last edit was 9 minutes ago
New cell UI: OFF
Run all
Connect
Share
Publish
1 from pyspark.sql import SparkSession
2
3 spark = SparkSession \
4 .builder \
5 .appName("Python Spark SQL basic example") \
6 .config("spark.some.config.option", "some-value") \
7 .getOrCreate()
Command took 0.16 seconds -- by a user at 10/02/2024, 15:13:38 on unknown compute
Cmd 2
Python
1 df = spark.read.json("/FileStore/tables/bus/people.json")
2
3 df.show()
4
df: pyspark.sql.dataframe.DataFrame = [age: long, name: string]
+----+-------+
| age| name|
+----+-------+
null	Michael
30	Andy
19	Justin
+----+-------+


Spark SQL, DataFrames and Datasets Guide
Python
File Edit View Run Help Last edit was 9 minutes ago
New cell UI: OFF
Run all
Connect
Share
Publish
Python
1 # Print the schema in a tree format
2 df.printSchema()
3
4 df.select("name").show()
5
6 # Select everybody, but increment the age by 1
7 df.select(df['name'], df['age'] + 1).show()
8
9
10 # Select people older than 21
11 df.filter(df['age'] > 21).show()
12
13 # Count people by age
14 df.groupBy("age").count().show()
root
|-- age: long (nullable = true)
|-- name: string (nullable = true)
+-------+
| name|
+-------+
|Michael|
| Andy|
| Justin|
+-------+

```
+-------+---------+
|   name|(age + 1)|
+-------+---------+
|Michael|     null|
|   Andy|       31|
| Justin|       20|
+-------+---------+

+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+

+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    1|
|  30|    1|
+----+-----+
```

Command took 3.08 seconds -- by a user at 10/02/2024, 15:29:50 on unknown compute


Spark SQL, DataFrames and Datasets Guide
Python
File Edit View Run Help
Last edit was 11 minutes ago
New cell UI: OFF
Run all
Connect
Share
Publish
Cmd 5
Python
# Register the DataFrame as a global temporary view
df.createGlobalTempView("people")
# Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.people").show()
# Global temporary view is cross-session
spark.newSession().sql("SELECT * FROM global_temp.people").show()
+----+-------+
| age| name|
+----+-------+
null	Michael
30	Andy
19	Justin
+----+-------+
+----+-------+
| age| name|
+----+-------+
null	Michael
30	Andy
19	Justin
+----+-------+