**1.Read Modes In Spark**

**Core structure :**

Data frame Reader – .format(--)\ ‘\’ for new line

.Option( “Key”, “Value”)\

.Schema(---)\

.load(--)

Format() -> Data file format. Ex -CSV, JSON, JDBC/ODBC file format.

Option() -> It is optional ->infer schema, mode, header.

Schema() -> Manual schema that you can pass optional.

Load() -> Contains a path where your data location or file location.

**DataFrame Reader API:**

Spark.Read() – Used to access DataFrame API.

Ex –

spark.read.format("csv")\

.option("header","false")\

.option("inferschema","false")\

.option("mode","FAILFAST")\

.load("/FileStore/tables/Flight\_data-1.csv")

.option("inferschema","false") –If inferschema is false then it loads data without any datatype consideration and it is true then it will infer column datatype as per column data.

Ex –

A close up of a document

Description automatically generated

If we print **flight\_df.printSchema() –**If inferschema = ‘False’ then it considers all data as a string. If Inferschema = ‘True’ then it considers ‘Count ’ columns as a Int due to whole values in that columns.

**Mode**

**Types of Mode**

1. **Fail Fast:**

Fail execution if malformed record in dataset.

1. **Drop Malformed:**

Drop the corrupted records.

1. **Permissive:**

* It is default that means it allows all the records.
* It is set NULL values to corrupted records.

**2.Schema In Spark**

**Creating Manual Schema**

**Possible Interview Questions**

1.How to create a Schema is Pyspark?

* There are two methods of schema creation in Spark.
* StructType and StructField
* DDL

from pyspark.sql.types import StructType,StructField,StringType,IntegerType

myschema = StructType([

    StructField("DEST\_COUNTRY\_NAME", StringType(), True),

    StructField("ORIGIN\_COUNTRY\_NAME", StringType(), True),

    StructField("count", IntegerType(), True)

])

flight\_df = spark.read.format("csv") \

    .option("header", "true") \

    .option("inferSchema", "false") \

    .schema(myschema)\

    .option("mode", "FAILFAST") \

    .load("/FileStore/tables/Flight\_data-1.csv")

flight\_df.printSchema()

%fs – for checking our file

ls/FileStore/tables

https://spark.apache.org/docs/latest/sql-data-sources-csv.html

**DDL :**

**Ex -** ddl\_myschema = “id integer, name string, age integer”

2.What are other ways of creating a Schema?

* Creating a DataFrame without creating schema just set .option(“inferschema”,”True”)

3.What is ‘Struct’ and ‘StructType’ in Schema?

* **StructType:** It defines the structure of our DataFrame. Also, we called as list of StructField.
* **StructField:** It is nothing but a single field within the DataFrame.

4.What if I have header in my Data?

* If header is already present in data, then we can just skip 1’st row and set Header is True.

Ex – .option(“SkipRow”,1,True)

**Handling Corrupted Records In Spark**

**Potential Interview Questions**

1. Have you worked with corrupted records?

* Yes.

1. When do you say that it is a corrupted record?

* ****

Above file is a comma separated in that for id3 we have in address there are 2 addresses Bangalore, India in that case we can say it is a corrupted record due to it consider that India as separate column record.

1. What happens when we encounter with corrupted records in different read modes?

* While mode is ‘**FAILFAST’**  it does not enter a single record in table.
* While mode is ‘**PERMISSIVE’** it allows to insert records in table and place NULL for malformed records.
* While mode is **‘DROPMALFORMED’** it drops the malformed records from the data.

1. How can we print bad records?

from pyspark.sql.types import StructType,StructField,StringType,IntegerType

employee\_schema = StructType([

    StructField("id",IntegerType(),True),

    StructField("name",StringType(),True),

    StructField("age",IntegerType(),True),

    StructField("salary",IntegerType(),True),

    StructField("address",StringType(),True),

    StructField("nominee",StringType(),True),

    StructField("\_corrupt\_record",StringType(),True)

])

employee\_df = spark.read.format("csv")\

    .option("header","true")\

    .option("inferschema","true")\

    .option("mode","PERMISSIVE")\

    .schema(employee\_schema)\

    .load("/FileStore/tables/employee.csv")

employee\_df.show(truncate= False)

1. Where do you store corrupted records and how we can access it later?

employee\_df = spark.read.format("csv")\

.option("header","true")\

.option("inferschema","true")\

.schema(employee\_schema)\

.option("badRecordsPath","/FileStore/tables/bad\_records")\

 .load("/FileStore/tables/employee.csv")

**How to read JSON file in Pyspark**

**Possible Interview Questions**

1.What is JSON data and how to read it in spark?

* JSON is a JavaScript Object Notation which means the data representation in ‘KEY’ and ‘Value’ pair. Also it contains semi structured type of data .

Ex – {“ID”:1,

”Name”:”Shivaji”}

Key Value

spark.read.format("json")\

    .option("inferSchema","true")\

    .option("mode","PERMISSIVE")\

    .load("/FileStore/tables/line\_delimited\_json.json").show()

Line-Delimited JSON:

* Line-delimited JSON is a format where each line of the file contains a single JSON object. This format is useful for streaming applications or for processing large datasets that are too big to fit into memory all at once.

Ex- {"name": "Alice", "age": 34}

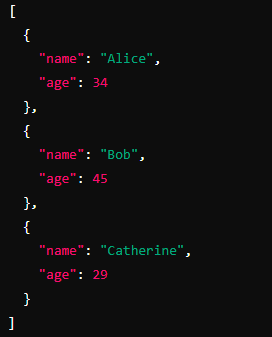
{"name": "Bob", "age": 45}

{"name": "Catherine", "age": 29}

Multi-Line JSON:

* Multi-line JSON (sometimes just called JSON) is a format where a single JSON object or array is written over multiple lines. This format is commonly used for smaller datasets or configuration files where the JSON data is nested and complex, making it more readable.

Ex –



2.What if I have 3 keys in all line and 4 keys in one line?

* It returns the value in 4 key column and remaining column of 4th key values should be Null.

EX-

{"name":"Manish","age":20,"salary":20000},

{"name":"Nikita","age":25,"salary":21000},

{"name":"Pritam","age":16,"salary":22000},

{"name":"Prantosh","age":35,"salary":25000},

{"name":"Vikash","age":67,"salary":40000,"gender":"M"}

A screenshot of a computer code

Description automatically generated

3.What will happen if I have corrupted json file?

* If we load corrupted json file, it will give us a extra column that store the corrupted record in that column.

spark.read.format("json")\

    .option("inferSchema","true")\

    .option("mode","PERMISSIVE")\

    .load("/FileStore/tables/corrupted\_json.json").show(truncate=False)

A screenshot of a computer

Description automatically generated

Nested JSON:

spark.read.format("json")\

    .option("inferSchema","true")\

    .option("mode","PERMISSIVE")\

    .option("multiline","true")\

    .load("/FileStore/tables/File5.json").printSchema()

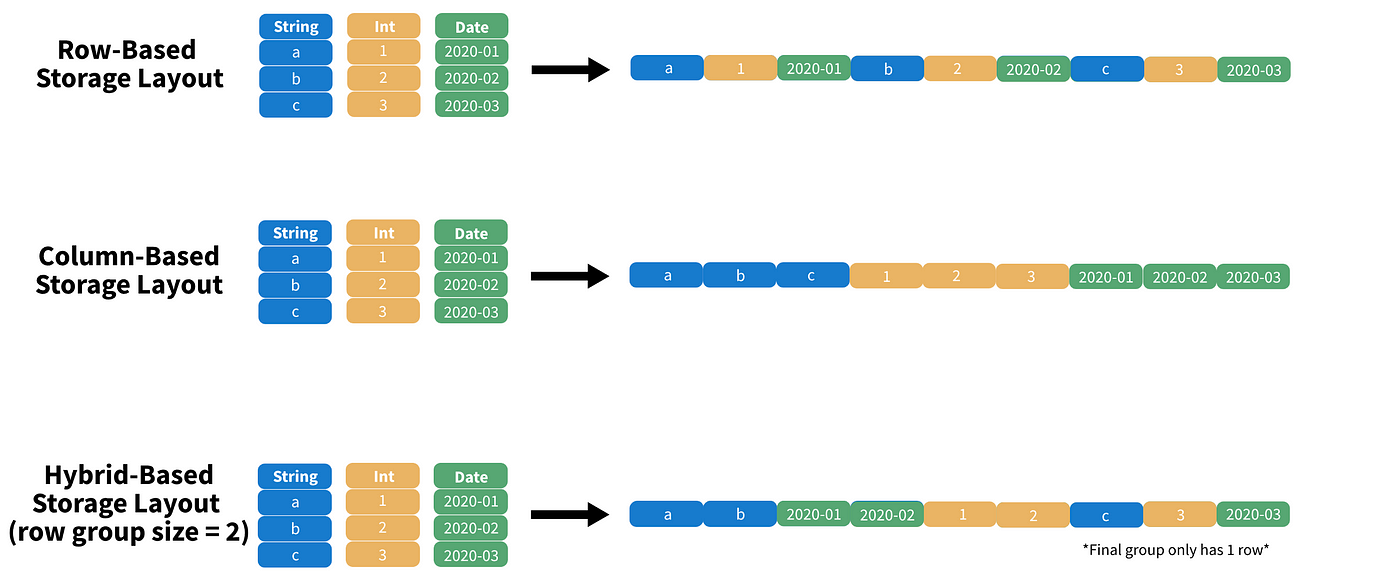
**Parquet File Format**

**Potential Interview Questions**

1.What is Parquet file format?

* Parquet file format is a columnar storage file format optimized for use with big data processing frameworks like Apache Hadoop, Apache Spark, and others. It is particularly well-suited for analytical querying on large datasets.

2.What is Columnar and Row based file format?



Row Based Storage:

* In row-oriented storage, data is stored row by row. Each row contains all the fields for a single record, and these records are stored sequentially.
* Efficient for row-wise access and modification.

Ex –

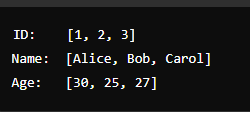
A group of pink text on a black background

Description automatically generated

Column Based Storage:

* In column-oriented storage, data is stored column by column. All values for a particular column are stored together. This format is efficient for analytical workloads where operations typically involve accessing a subset of columns over many rows.
* Efficient for read-heavy analytical queries that access only a few columns.

Ex-

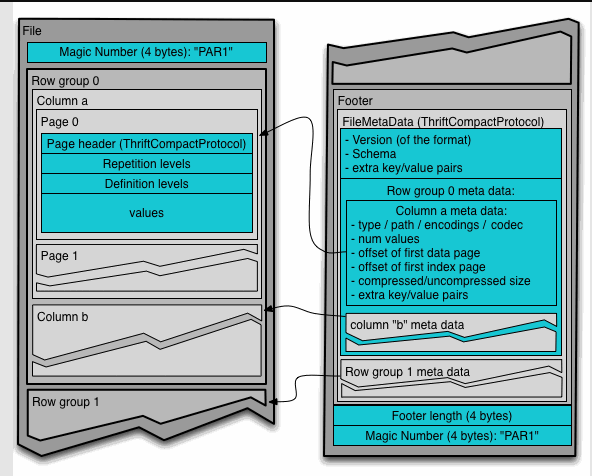


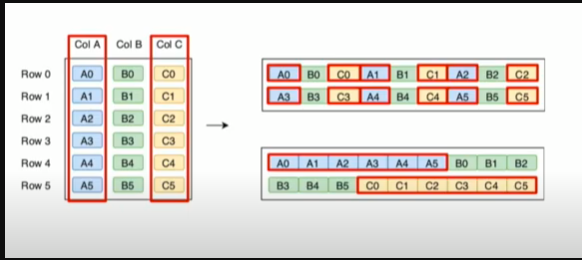
Hybrid Storage:

* Parquet employs a hybrid approach to balance the benefits of both row-oriented and column-oriented storage. It uses a columnar format but also groups data into row groups.

 **Row-wise access**: Within a row group, accessing all columns for a specific row is efficient.

 **Columnar benefits**: For analytical queries, only the necessary column chunks are read, reducing I/O.



* Suppose we have 1 million records then it will group the records as partition on group level.it follows a hybrid-based storage for increasing speed of query.
* 
* In Row 0 we have column A,B,C then we have divided that 1 million records into partition of 1 lack records into one row group that is Row 0.
* That means in Row 0 stored 1st 1 lack records as A0,A1,A2,…..A99999 as columnar manner.
* Again, each column have separate page that contains the data information.

parquet\_file = pq.ParquetFile(rr'C:\Users\waghnshi\Downloads\part-r-00000-1a9822ba-b8fb-4d8e-844a-ea30d0801b9e.gz.parquet')

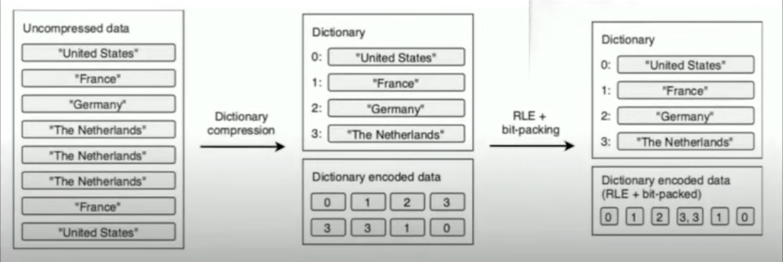
parquet\_file.metadata

parquet\_file.metadata.row\_group(0)

parquet\_file.metadata.row\_group(0).column(0)

parquet\_file.metadata.row\_group(0).column(0).statistics

By using above commands in CMD we can see the metadata information of parquet file.



In above diagram we learn what is **Compression**, **RLE(Run length Encoding)** and **bit packing**.

For example, we have column Destination Country that contains the 1 lack records but we know that we have only 200 countries are present then we consider that repetitive data is present in that column.

**1.Compression:**

* In above diagram the repetitive data is encoding in dictionary that gives the number to each unique country that means compression of unique data with encoding and it’s metadata have that complete information of encoded data.
* For Example follow above diagram.

**2.RLE(Run Length Encoding):**

* In Run length encoding it will compress the repetitive data in continuous memory.
* Ex – If we have encoded data like 0,1,2,3,3,3,1,0 then in run length encoding it will compress as like 0,1,2,3,3,1,0 and it will automatically reduce the storage of that repetitive data that means bit packing.

**Data Organization in Parquet:**

**File**

**Row Group(We have metadata at group level also)**

**Column**

**Pages**

**Metadata**

**Min**

**Max**

**Count**

* We have metadata in Row Group level so we can only scan records that we want so that’s why it did not waste time and I/O.

What is predicate pushdown and projection pruning?

**Predicate Pushdown:**

Reduces the number of rows read by applying filters at the storage level.

**How It Works:**

**Initial Query**: A query with a filter condition (e.g., SELECT \* FROM table WHERE age > 30).

**Pushdown to Storage Layer**: The query engine passes the filter condition to the storage engine.

**Storage Engine Filters Data**: The storage engine reads only the data that satisfies the filter condition, reducing I/O and processing overhead.

**Query Engine Processes Filtered Data**: The query engine processes only the filtered data, which is a subset of the original dataset.

#### Example:

Consider a Parquet file with a column age and a query SELECT \* FROM users WHERE age > 30. Without predicate pushdown, the query engine would read all rows and then filter out those where age <= 30. With predicate pushdown, the storage layer reads only the rows where age > 30.

**Projection Pruning:**

Reduces the number of columns read by selecting only the necessary columns for the query.

#### How It Works:

1. **Initial Query**: A query specifying certain columns (e.g., SELECT name, age FROM table).
2. **Identify Required Columns**: The query engine identifies the columns needed for the query.
3. **Read Only Required Columns**: The storage engine reads only the specified columns from the storage.
4. **Query Engine Processes Selected Columns**: The query engine processes only the data from the selected columns.

#### Example:

Consider a Parquet file with columns id, name, age, and address and a query SELECT name, age FROM users. Without projection pruning, the query engine would read all columns. With projection pruning, only name and age columns are read.

#### Example:

Consider a Parquet file with columns id, name, age, address, and a query SELECT name, age FROM users WHERE age > 30.

* **Predicate Pushdown**: The storage engine reads only rows where age > 30.
* **Projection Pruning**: Only the name and age columns are read from the filtered rows.

Df=spark.read.parquet("/FileStore/tables/part\_r\_00000\_1a9822ba\_b8fb\_4d8e\_844a\_ea30d0801b9e\_gz.parquet")

**Write Data in spark(On Disk)**

**Potential Interview Questions**

**1.What are the modes available in DataFrame writer?**

**2.What is Partition By and Bucket By?**

**3.How to write data into multiple positions?**

**DataFrame Writer General Structure:**

DataFrame writer. Format()\

.option()\

.PartitionBy()\

.bucketBy()\

.save()

**Modes In DataFrame writer API**

**Append:** This mode appends the data to the existing data. If the data already exists at the target location, the new data is added to the existing data.

**Overwrite:** This mode overwrites the existing data. If the data already exists at the target location, it is deleted and replaced with the new data.

**Ignore:** This mode ignores the save operation if the data already exists. If the data already exists at the target location, no changes are made.

**ErrorIfExists (or Fail):** This mode raises an error if the data already exists at the target location. This is the default behavior.

Partitioning and Bucketing

**1.What is Partitioning in Spark?**

* Partitioning in spark is nothing but when you partition a DataFrame or Dataset it creates a directory for each partition and saves data files under directory.
* from pyspark.sql import SparkSession

df= spark.read.format("csv")\

    .option("header","true")\

    .option("inferSchema","true")\

    .load("/FileStore/tables/Partition.csv")

**Write file with partition on disk.**

df.write.format("csv")\

    .option("header","true")\

    .option("mode","overwrite")\

    .option("path","/FileStore/tables/PartitionBy\_address.csv")\

    .partitionBy("address")\

    .save()

dbutils.fs.ls("/FileStore/tables/PartitionBy\_address.csv")

**2.What is Bucketing in Spark?**

* Bucket is nothing but it creates a specified number of files(buckets) and distributes data evenly across files.
* It does not support .save(). It can stores that data in Hive metaStore.
* df.write.format("csv")\

    .option("header","true")\

    .option("mode","overwrite")\

    .option("path","/FileStore/tables/BucketBy\_id.csv")\

    .bucketBy(3,"id")\

    .saveAsTable("BucketBy\_Id\_table")

dbutils.fs.ls("/FileStore/tables/BucketBy\_id.csv")

**3.Why do we need these two?**

**Partitioning**:

* Organizes data into separate directories based on the values of one or more columns. This is useful for improving query performance by enabling partition pruning, where queries can skip entire partitions that do not match the filter criteria.

**Bucketing**:

* Distributes data into a fixed number of buckets based on the hash of a specified column. This is especially useful for optimizing join operations by ensuring an even distribution of data across the buckets.

**4.When to use Partitioning?**

 **Filtering**:

* When your queries frequently filter data based on specific column values.

 **Data Management**:

* When you need to manage data in logical segments, such as deleting or archiving old data without affecting other data.

 **Query Optimization**:

* When you need to improve query performance by enabling partition pruning.

**5.When to use Bucketing?**

 **Join Optimization**:

* When you frequently perform join operations on a specific column and want to reduce the shuffle cost.

 **Balanced Data Distribution**:

* When you need to ensure that data is evenly distributed across buckets to avoid data skews. that means you have same columns in both tables with same column name.

 **Large Datasets**:

* When working with large datasets where pre-shuffling the data into buckets can significantly improve query performance.

**How to create DataFrame in Spark**

**table**

my\_data =[(1,   1),

(2,   1),

(3,   1),

(4,  2),

(5,   1),

(6,  2),

(7,  2)]

Schema

my\_schema = ['id','num']

**DataFrame**

my\_df = spark.createDataFrame(data=my\_data,schema=my\_schema).show()

DataFrame Transformation in Spark

**1.What is schema?**

**2.What is DataFrame?**

**3.How to select columns ?**

**4.How many ways to select columns?**

**5.What is expression?**