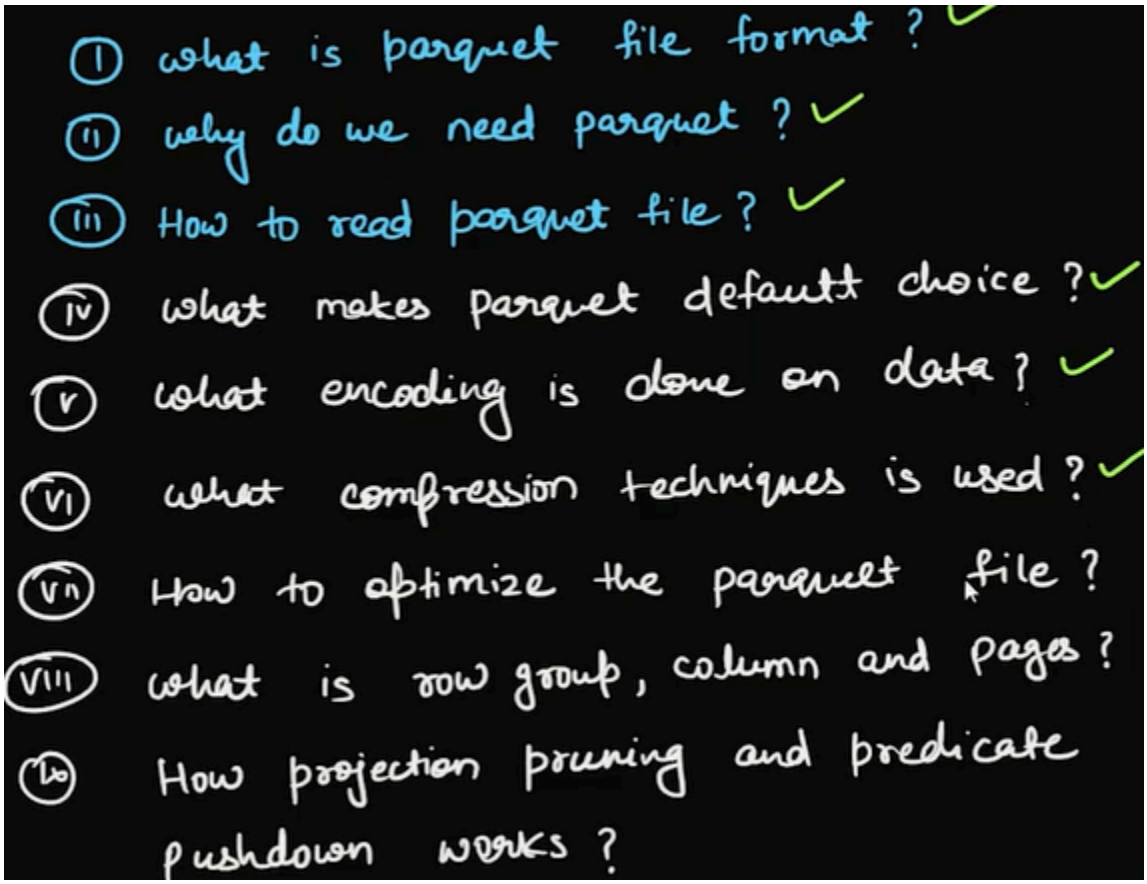
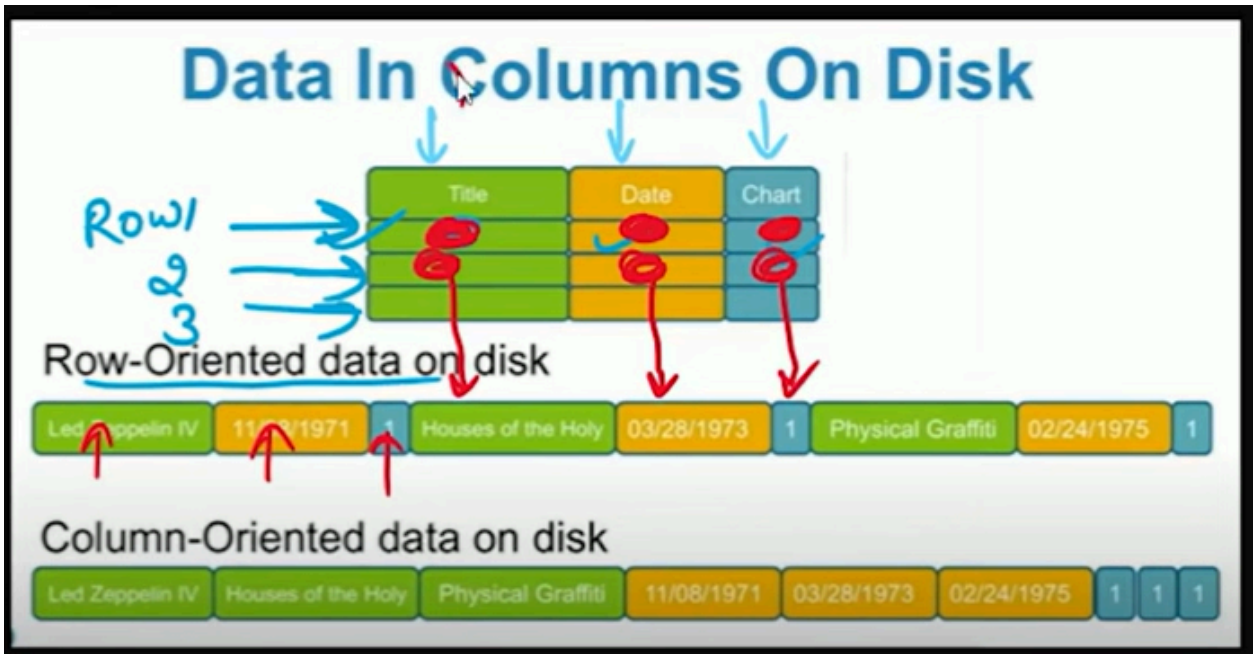
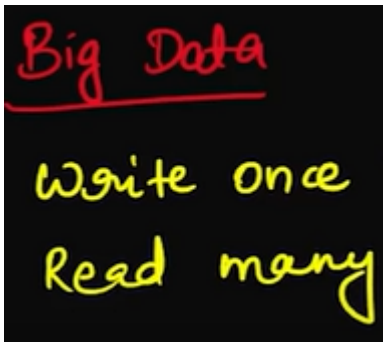
Parquet File Format

1. 

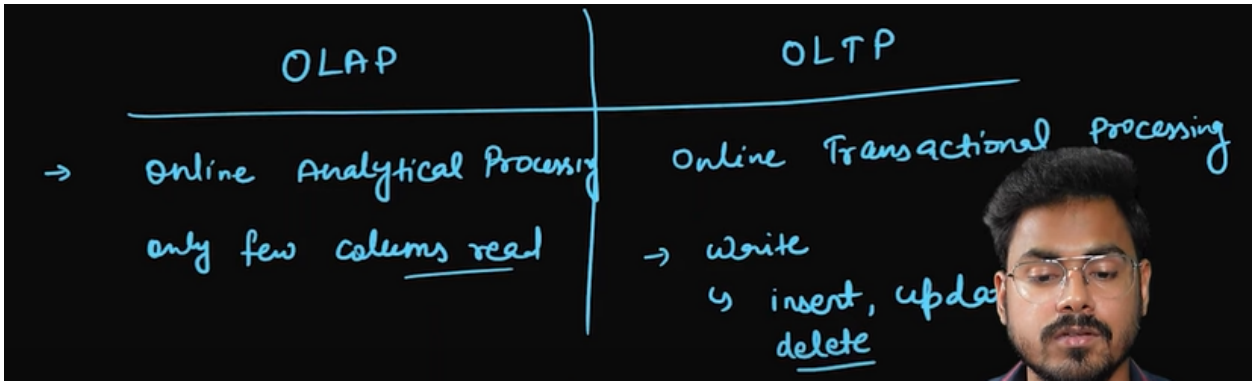
2. Parquet is a columnar based file format

3. So how does it store in the physical layer? Here in this picture ..we can see how row-oriented stores the data and how columnar stores the data

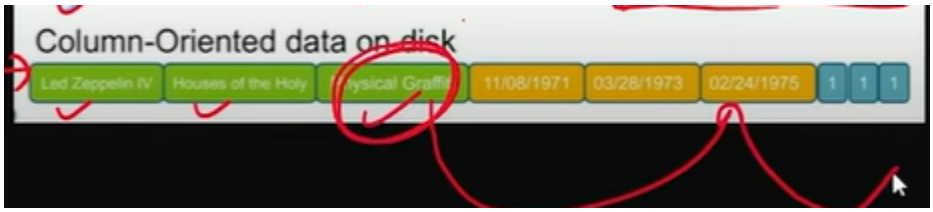


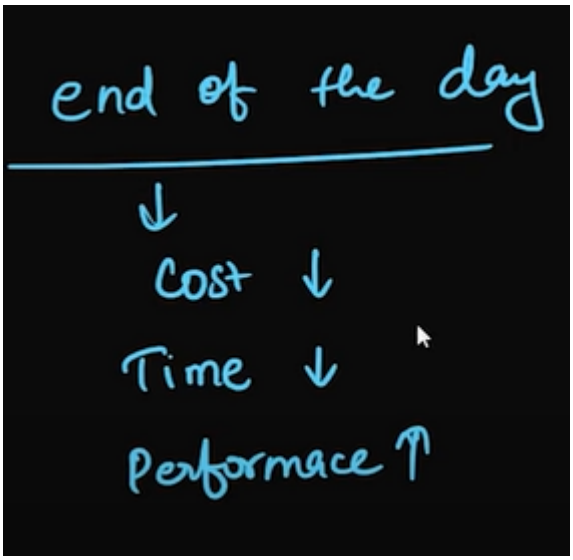
4. In BigData we follow 

5. Lets suppose we want to access the data of specific column
6. If we use row oriented format…then it needs scan the each row and jump to next row(which needs more processing)
7. But the same in columnar format..we can directly retrieve the column which we need

8. Here comes the concept of OLAP and OLTP
9. 

10. OLTP is used in banking sector(for example to update the customer transactions, withdrawals etc) and OLAP is for analytical purposes
11. Here if we want to update the customer data(each row is a customer)..then we can just go the customer's id data and update his data
12. But if we want to do the same in columnar..then we need to jump between columns



13. End of the day we need 

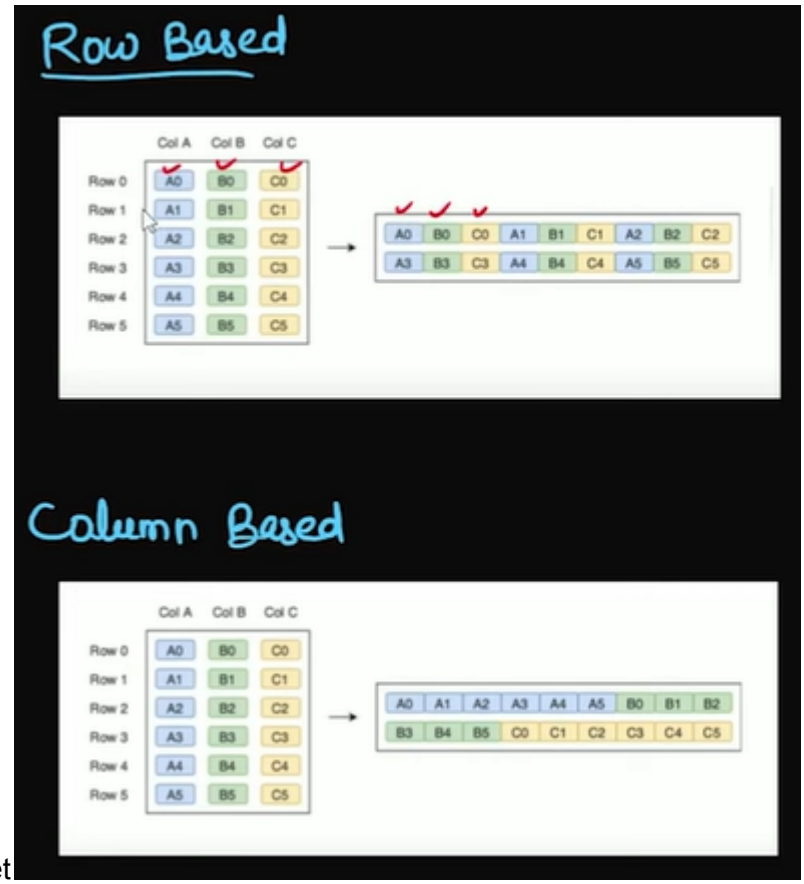
14. Practical:
15. To read a parquet file in dataframe we use:

```
df= spark.read.parquet("/FileStore/tables/part_r_00000_1a9822ba_b8fb_4d8e_844a_ea30d0801b9e_gz.parquet")
df.show()
```

16. output

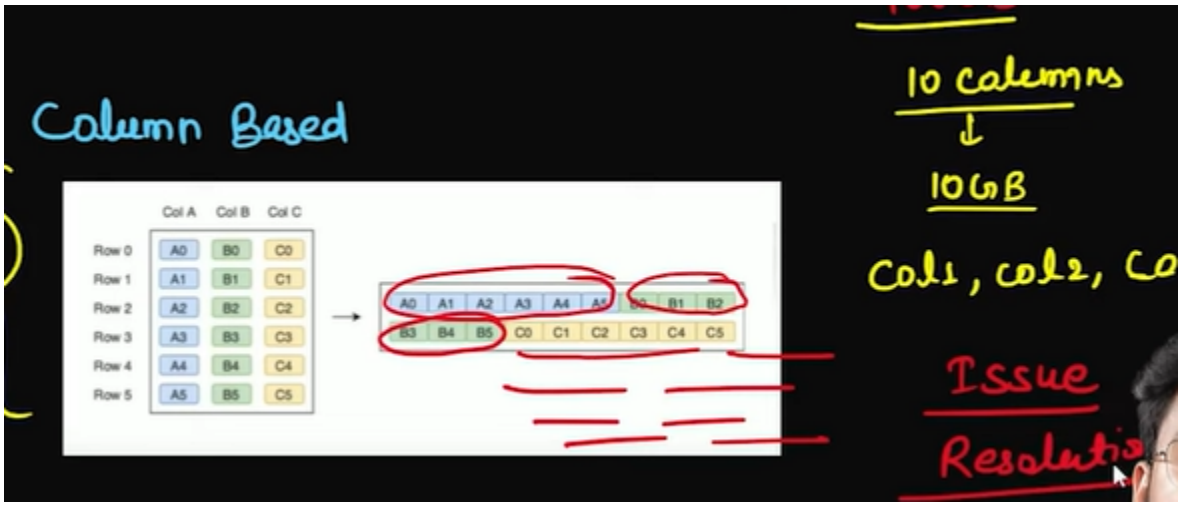
DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	1
United States	Ireland	264
United States	India	69
Egypt	United States	24
Equatorial Guinea	United States	1
United States	Singapore	25

17. Deepdive of parquet

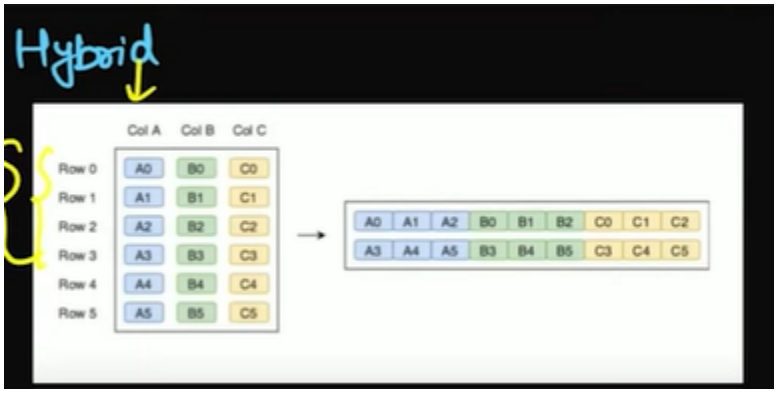


18. Lets suppose we have file of 100GB and 10 columns wer each column's store's 10GB of data

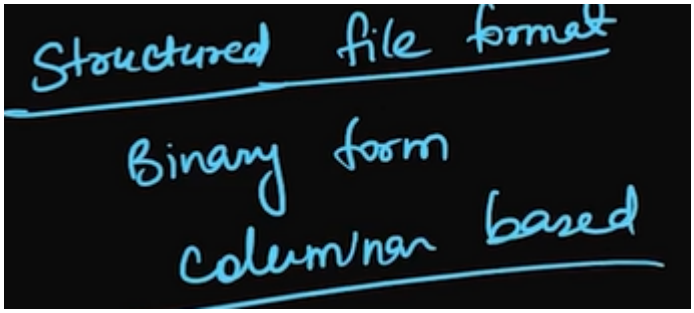
19. Now if we want to read all the columns ..then we need to scan all the columns ..which does not serve our purpose of cost,time and performance

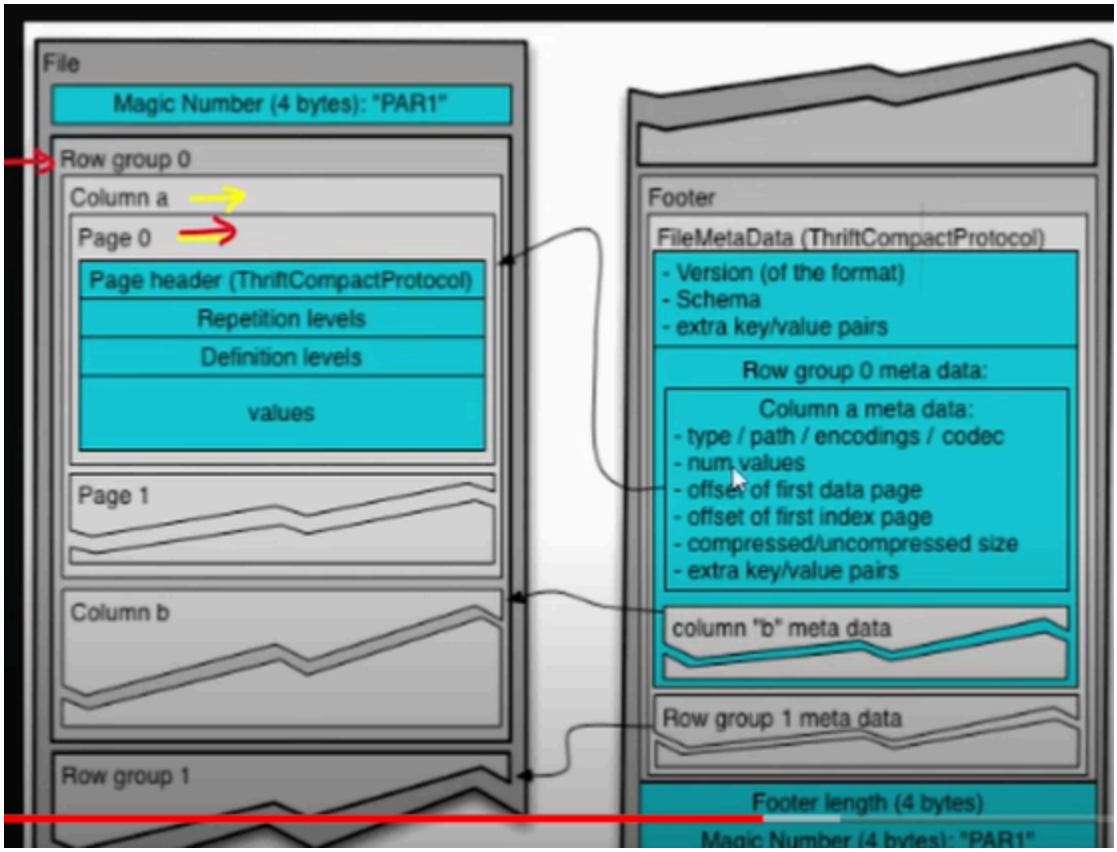


20. So here we use HYBRID of row and column

21. 

Parquet is referred as columnar format in many books but internally it is like an hybrid format (Combination of Row and Columnar). Parquet defines a small number of primitive types. The data stored in a Parquet file is described by a schema, which has at its root a message containing a group of fields. Dec 9, 2020

22. So here parquet is 

23. 

Row Groups and Column Groups:

- The image highlights two sections: "Row group 0" and "Column a". These represent row groups and column groups, respectively.
- **Row groups:** Parquet files are divided into row groups, which are essentially data chunks containing a subset of rows from the table. This improves efficiency when reading specific data ranges.
- **Column groups:** Data within each row group is further organized into column groups. Each column group stores the data for a single column from all the rows within that row group. This columnar storage is the core benefit of Parquet, allowing for efficient compression and retrieval of specific columns.

24.

Parquet File Metadata:

The bottom section of the image shows the file metadata. This information helps data processing tools understand the structure and content of the Parquet file. It includes details like:

- **Magic Number:** An identifier that validates the file format (e.g., "PAR1" for Parquet).
- **Footer Length:** Indicates the size of the file footer, containing important metadata about the data.
- **Schema:** Defines the data types and structure of the columns stored within the file.
- **Row group and column metadata:** Provides information specific to each row group and column group, such as the number of values, data encodings used for compression, and the offset (location) of data within the file.

Overall, the image depicts the internal structure of a Parquet file, highlighting the separation of data into row groups and column groups, along with the metadata that describes the file's contents.

Here are some additional points to consider:

- Parquet offers efficient storage and faster reads for queries that target specific columns compared to traditional row-based storage formats like CSV.
- While Parquet is a columnar format, some implementations might combine it with row-based storage within a hybrid model for specific use cases.

25.

26. To read parquet file in cmd we use

```
C:\Users\nikita>parquet-tools show  C:\Users\nikita\Downloads\Spark-The-Definitive-Guide-master\data\f
light-data\parquet\2010-summary.parquet\part-r-00000-1a9822ba-b8fb-4d8e-844a-ea30d0801b9e.gz.parquet
```

27. To inspect our parquet we use

```
C:\Users\nikita>parquet-tools inspect  C:\Users\nikita\Downloads\Spark-The-Definitive-Guide-master\dat
a\flight-data\parquet\2010-summary.parquet\part-r-00000-1a9822ba-b8fb-4d8e-844a-ea30d0801b9e.gz.parque
```

28. This inspect will give the metadata of our parquet file

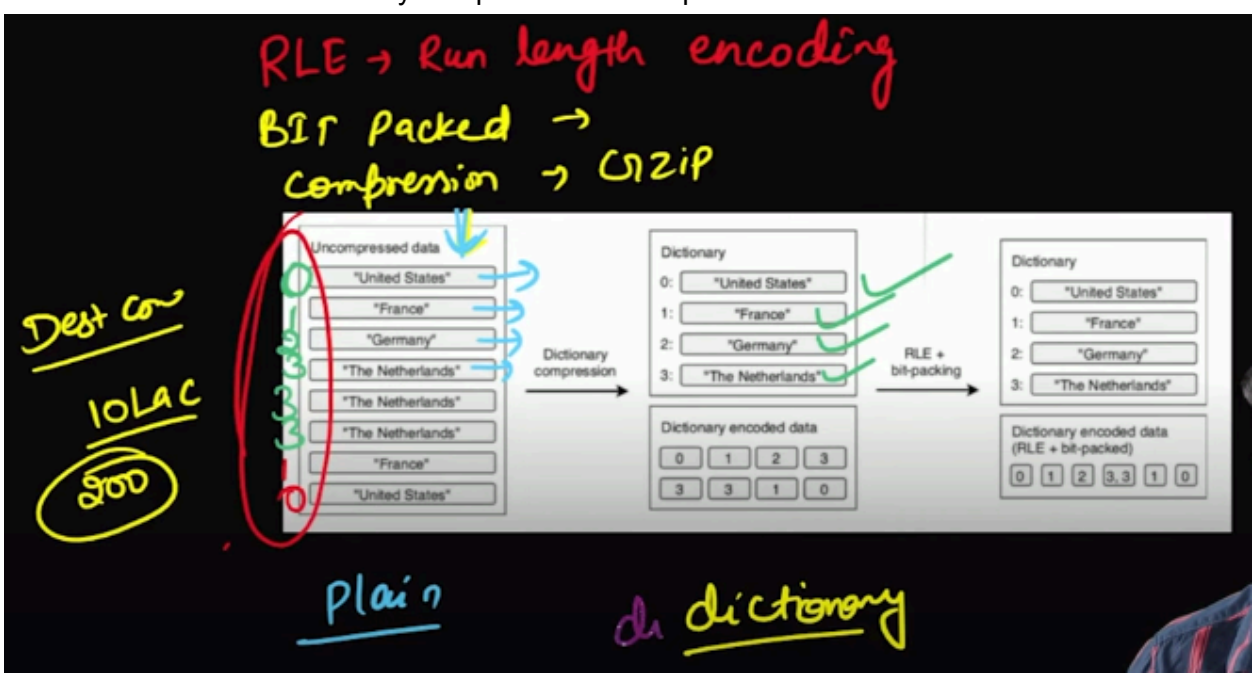
```
############ file meta data ############
created_by: parquet-mr (build 32c46643845ea8a705c35d4ec8fc654cc8ff816d)
num_columns: 3
num_rows: 255
num_row_groups: 1
format_version: 1.0
serialized_size: 658
```

29.

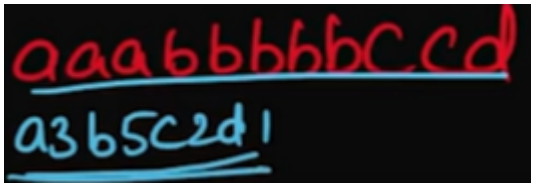
```
parquet_file = pq.ParquetFile(r'C:\Users\nikita\Downloads\Spark-The-Definitive-Guide-master\data\
parquet_file.metadata
parquet_file.metadata.row_group(0)
parquet_file.metadata.row_group(0).column(0)
parquet_file.metadata.row_group(0).column(0).statistics
```

This gives the statistics in granular level..see below

30.
```
>>> parquet_file.metadata.row_group(0)
<pyarrow._parquet.RowGroupMetaData object at 0x00000236D18828B8>
  num_columns: 3
  num_rows: 255
  total_byte_size: 5642
>>> parquet_file.metadata.row_group(0).column(0).statistics
<pyarrow._parquet.Statistics object at 0x00000236D193CD18>
  has_min_max: True
  min: Afghanistan
  max: Vietnam
  null_count: 0
  distinct_count: 0
  num_values: 255
  physical_type: BYTE_ARRAY
  logical_type: String
  converted_type (legacy): UTF8
```
31. Here row_group(0).colums(0) ..gives us the details of column A
32. We have discussed initially ..that we write once,read many
33. So how does write works in spark?
34. Imagine we have 100k of rows in country column..but in reality we have only 200 countries ..so will use dictionary compression to compress the data..and then RLE+bit



35. Here we have sample of uncompressed data..then it undergoes dictionary compression and compresses to 

36. What is run length encoding? 

37. BitPacking

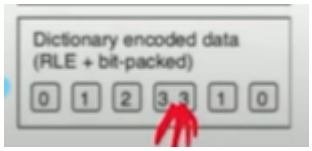
With Bit Packing:

1. **Identifying Candidate:** Parquet identifies "color_id" as a candidate for bit packing because it has a small range of unique values (1 to 3).
2. **Minimum Bits:** It calculates that only 2 bits are required to represent all the values (1 needs 1 bit, 2 and 3 need 2 bits).
3. **Packing the Data:**

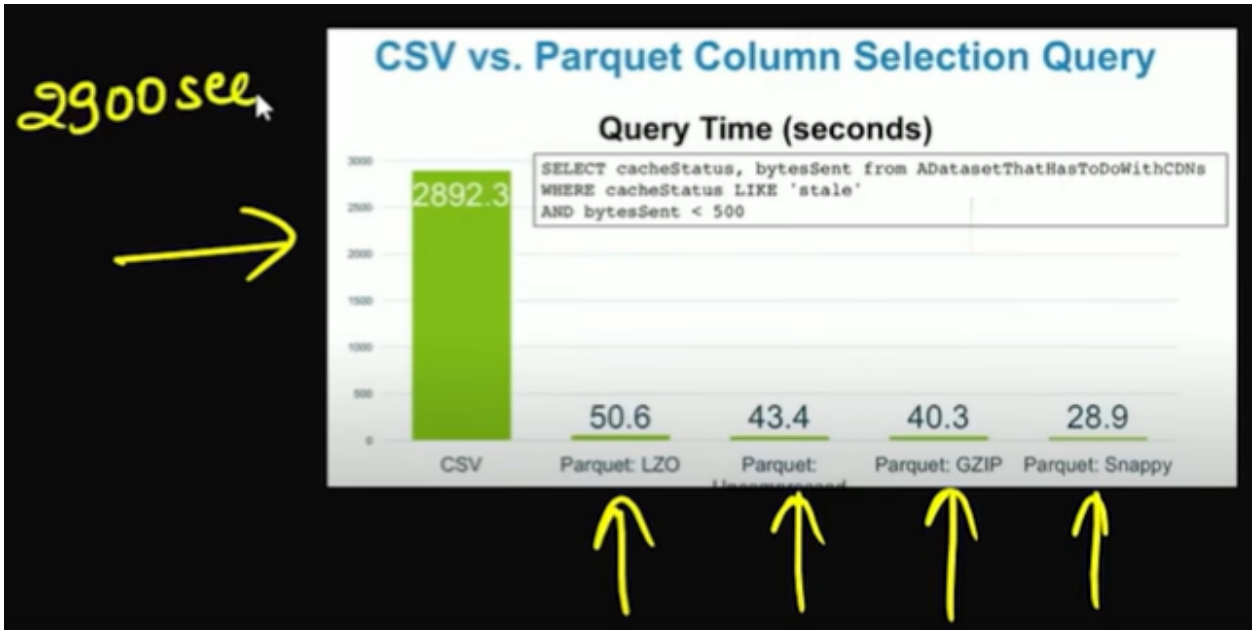
- Color ID | Binary Representation (2 bits)
- -------- | --------
- 1 | 01
- 2 | 10
- 3 | 11

These binary representations are then packed sequentially within a byte. For example, if the order of colors in the data is red, blue, green, the packed byte might look like:

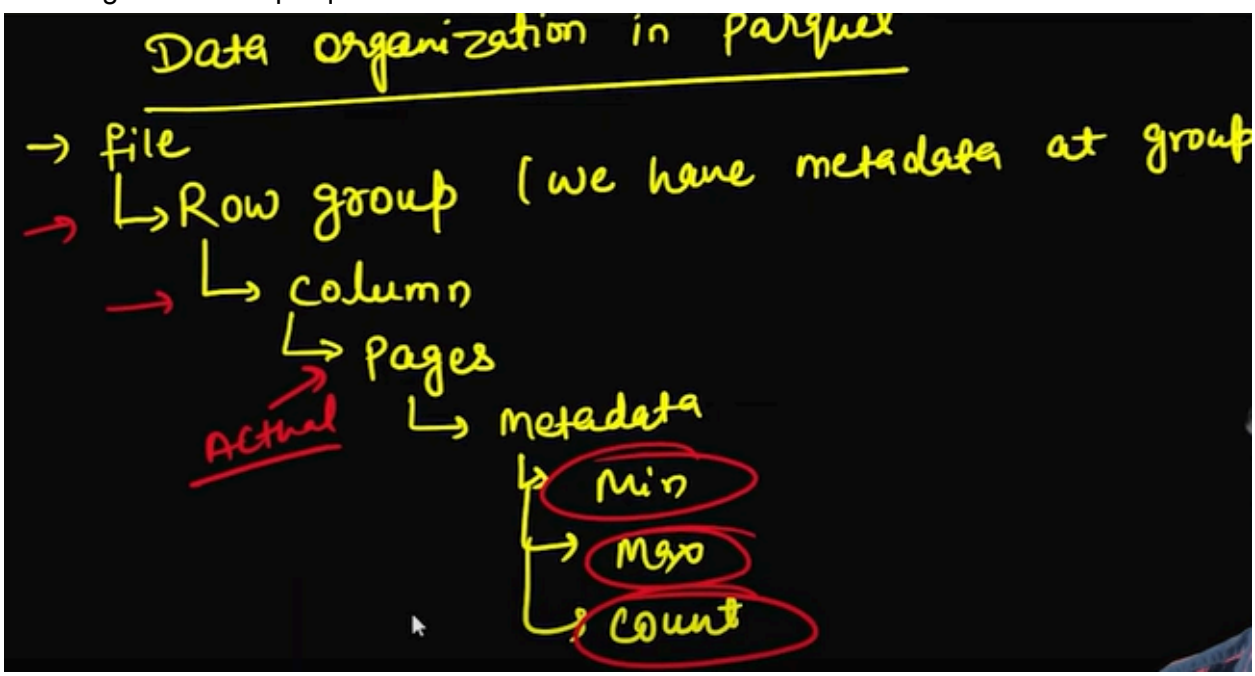
`01100000` (red's 2 bits followed by blue and green's 2 bits each, with padding zeros to fill the byte)

38. So with the help of all this..our data is compressed to 

(as it has metadata..it can read the data easily)

39. 

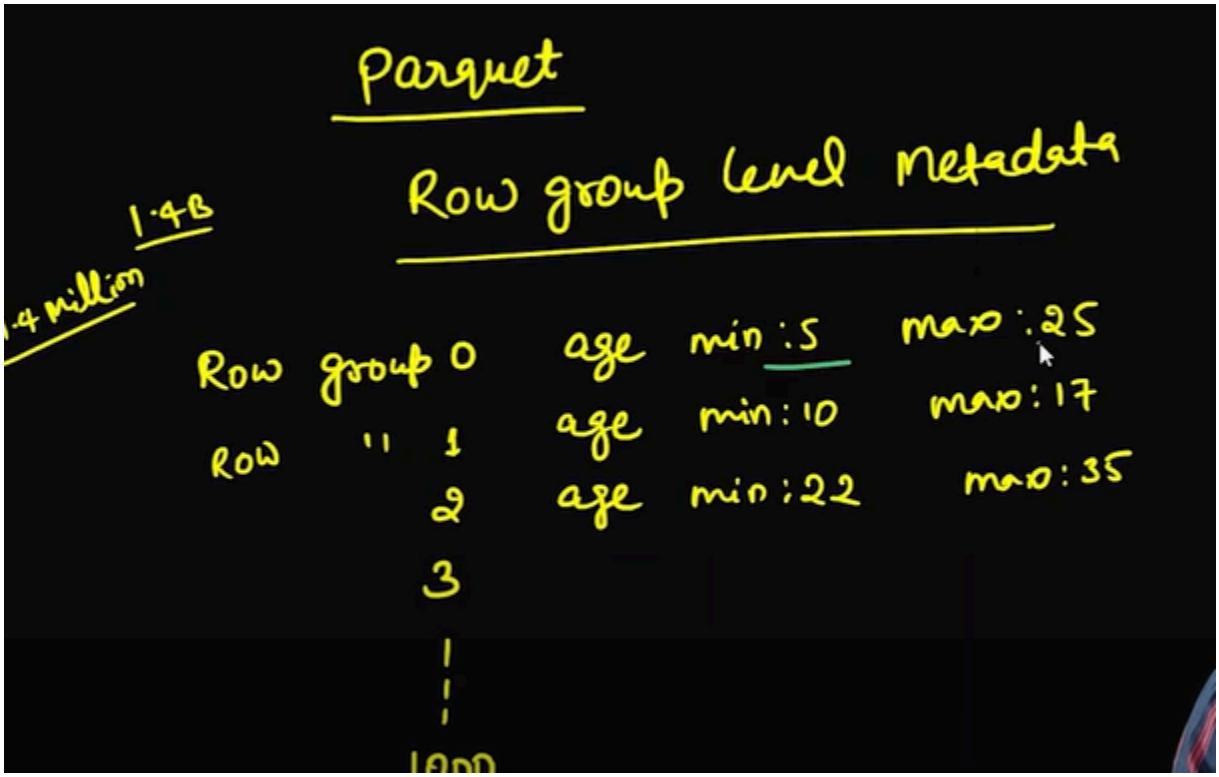
40. If we compress our data usign Snappy technique..then it gives the above query results in 29secons..which is 100x faster than csv
41. Data Organisation in parquet



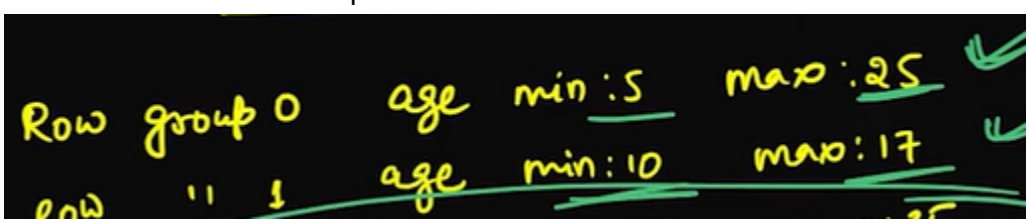
42. Optimization in parquet
43. Lets take an example query

44. Lets think we are dealing with aadhar data and we have 1.4B records
45. So in parquet format each row represent 1.4millions of data

46. So here we need age less than 18 right…
47. In parquet it also contains row level meta data



48. Now as we need details of person less than 18



49. We'll just scan row0 and row1 instead of scanning everything
50. Predicate Pushdown and Projection Pruning

Predicate Pushdown:

- **Concept:** It leverages the metadata and statistics stored in the Parquet file footer to filter data that doesn't meet the WHERE clause conditions (predicates) in the query.
- **Benefits:**
  - Reduces the amount of data transferred from storage to the processing engine. This is especially beneficial for large Parquet files.
  - Improves query processing speed as only relevant data needs to be processed.
- **Example:**

Imagine a Parquet table storing customer data with columns like "customer_id", "name", "city", and "purchase_date". You want to find customers who live in "Seattle" and made a purchase after "2023-01-01".

- **Without Predicate Pushdown:** The entire table data would be transferred to the processing engine. Then, the engine would filter the data based on the city and purchase date conditions.
- **With Predicate Pushdown:** The Parquet file footer likely contains statistics about the minimum and maximum values for each column ("city" and "purchase_date"). The processing engine can use these statistics to determine which data pages within the Parquet file might contain relevant data. Only those data pages are then transferred and further filtered based on the exact conditions.

51.

Projection Pushdown:

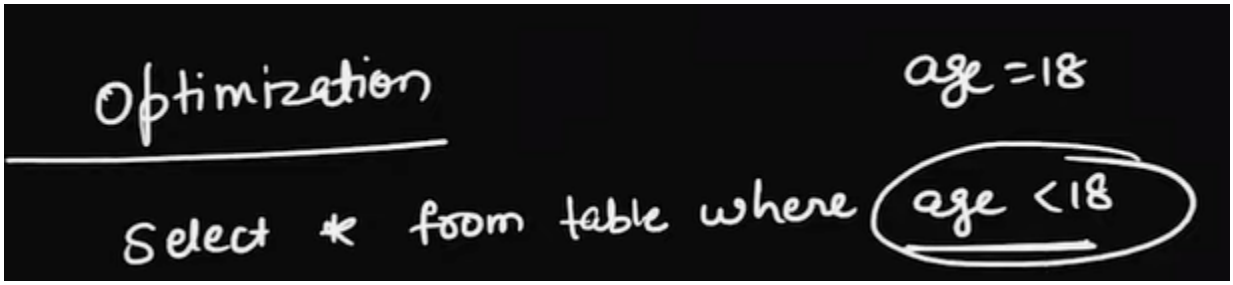
- **Concept:** It optimizes queries that only require specific columns (projections) by selectively reading only those columns from the Parquet file.
- **Benefits:**
  - Reduces data transfer by retrieving only the required columns, leading to faster query execution.
  - Minimizes processing overhead at the engine as it doesn't need to handle unnecessary data.
- **Example:**

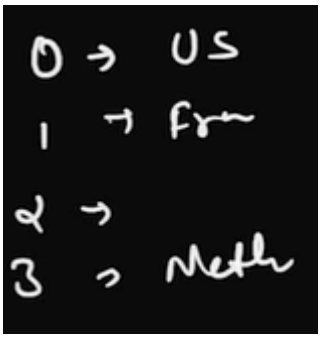
Continuing the previous customer data example, suppose you only need the "customer_id" and "city" columns for further analysis.

- **Without Projection Pushdown:** The entire table data (including all columns) would be transferred from the Parquet file, even though you only need two specific columns.
- **With Projection Pushdown:** The processing engine identifies the requested columns ("customer_id" and "city") and instructs the Parquet reader to only retrieve those specific columns from the file.

52.

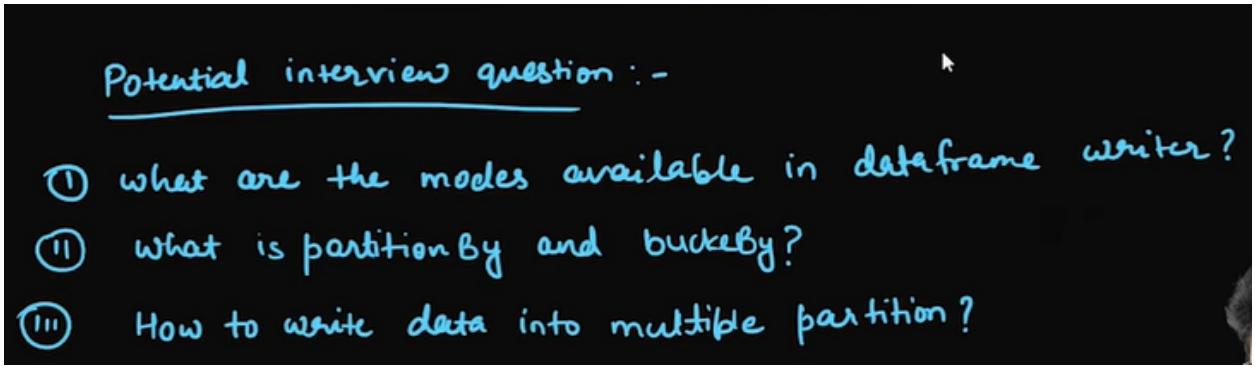
53. Now what if we need age = 18



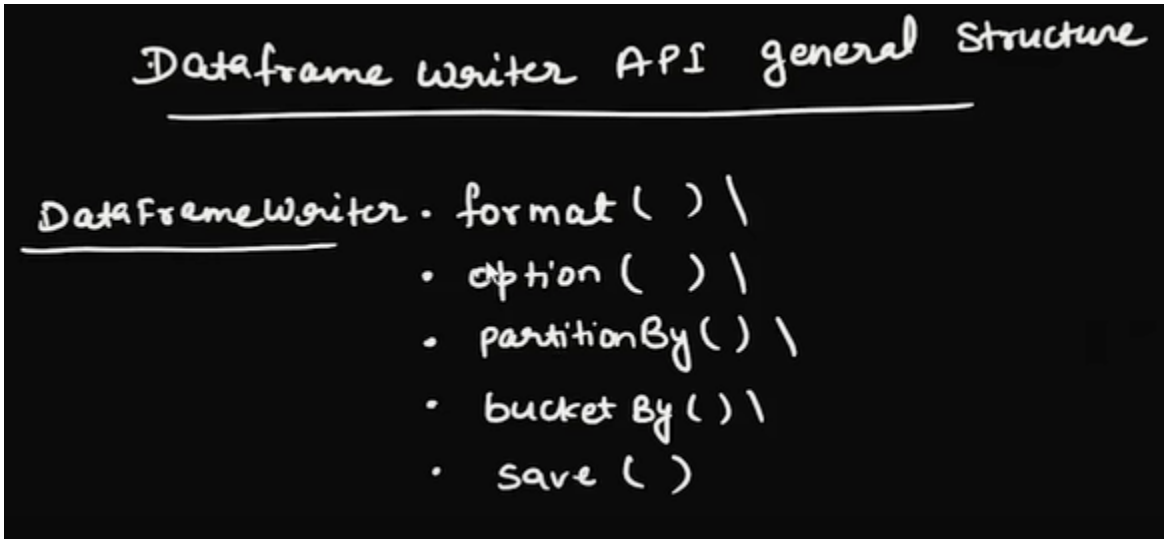


54. While compressing to dictionary format..we get result like this ..similart for age too
55. So if in dictionary meta data..if there's no age=18..then it does not even need to scan the original data..which make our query fast

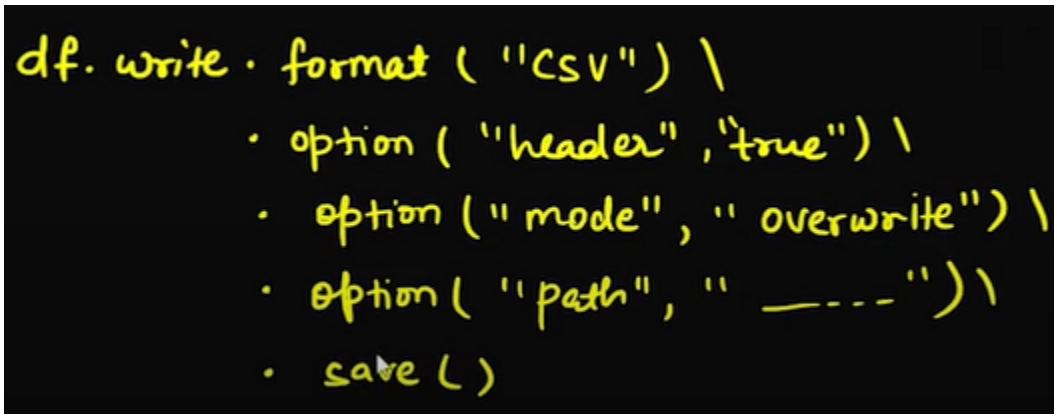
How to write data in spark

1. 

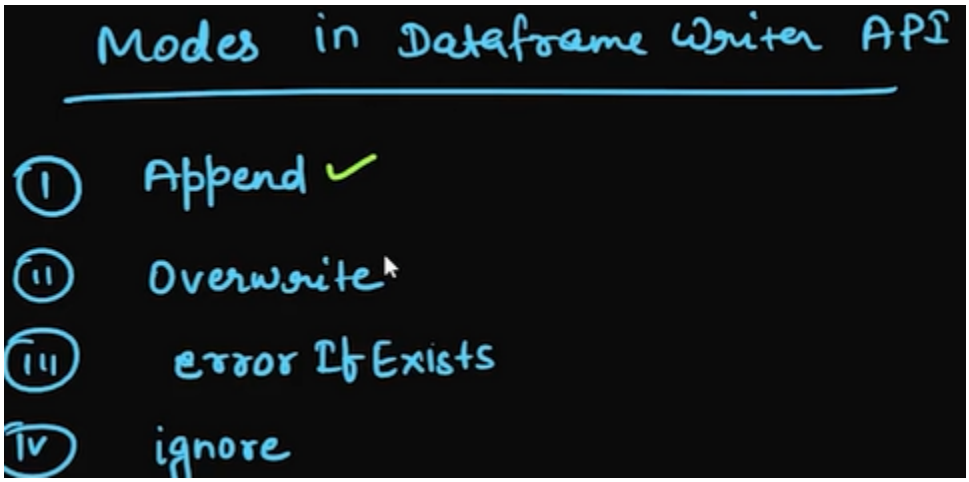
2. Dataframe write general structure



3. To write a CSV file ..we use



here path is wer to write our csv file

4. 

5. Modes Explained : https://g.co/gemini/share/9a9eadca9e8d

6. Writing a sample CSV file

```
df.write.format("csv")\
        .option("header","true")\
        .option("mode","overwrite")\
        .option("path","/FileStore/tables/csv_write/")\
        .save()
```

7. To see the files in our path we use

```
dbutils.fs.ls("/FileStore/tables/csv_write/")
```

8.