- CSV , JSON

1.) Parquet
2.) ORC
3.) AVRO

CSV , JSON

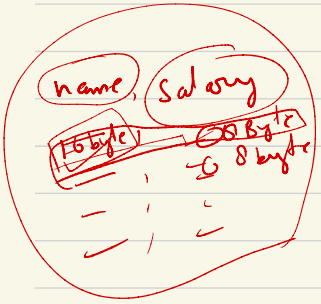
{ "name": Val
"Col": Val2
}

"name" → String
1 → int
76.5 → Float, double

machine can understand 0 & 1

'A' , 'B' , 'C'  1 Byte = 8 bit

Char = 1 Byte for Storage
int = 4 Byte for Storage
float = 4, 8, 16 byte for storage




1000 Row → 10 chars

Storage for one single Row
= 10 + 8 = 18 byte

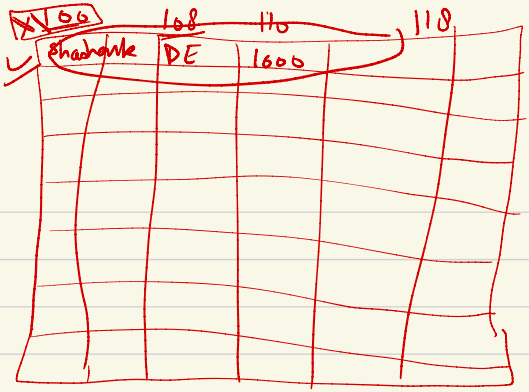
1000 × 18 = 18000 bytes

T1

| name | title | salary |
|---|---|---|
| Shan | DE | 1000 |
| Rahul | SDE | 2000 |
| Ankul | QA | 5000 |

table in MySQL DB
or any RDBMS
Row-Column format

Shashank = 8 char




int a = 4;

100 + 8 = 108

108 + 2 = 110

110 + 8 = 118

Shashank, DE, 1600

S 100 ---------- E 117

Rahul, SDE, 2000

S = 118 133

118 + 15 = 123

123 + 3 = 126

126 + 8 = 134

name, title, salary

Shashank, DE, 1000

Rahul, SDE, 2000

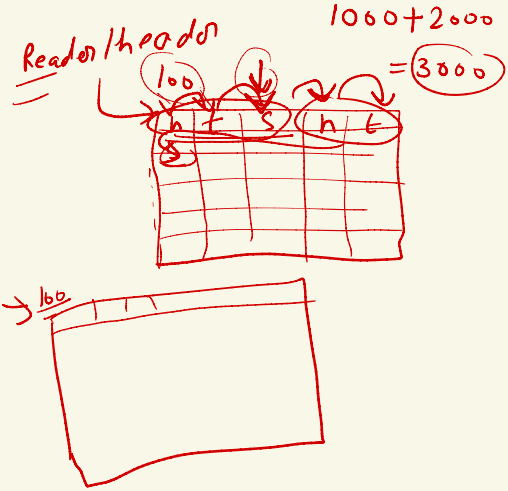
select sum(salary) from tbl;

① Row-Columnar

② Columnar Format

int a = 4;

Print(@a)




Disk)

Read header

write

Columnar format

| name, | title, | Salary |
|---|---|---|
| shashank, | DE) | 1000 |
| Rahul, | SDE) | 2000 |

Clounar




| name → | Shashank | Rahul | Ankur |
|---|---|---|---|
| title → | DE | SDE | QA |
| salary → | 1000 | 2000 | 5000 |

100 168 116

Disk Read

Select sum(Salary) from tbl;

NO SQL → Key-value, document, Columar

For frequent or heavy Analytical queries prefer Columnar formats or Columnar NoSQL DBS

1) Parquet (Columnar)

2) ORC (Columnar) [optimized Row-Columnar]

3) AVRO (Row-Columnar)

→ Compression

Spark → Parquet

Hive → ORC

AVRO → for frequent write & update

.csv
.JSON
.csv
.JSON

→ HDFS ← 100 GB storing the data

CSV = 1 GB
Parquet = 280 MB

1.) Fast Analytical queries
2.) Compression

KAFKA


Compression

ORC > Parquet > AVRO

Serilization & Deserilization

# objects

→ class, objects

→ physical Instance

```
class Car {
    String name;
    String model;
    float Price;
}
```

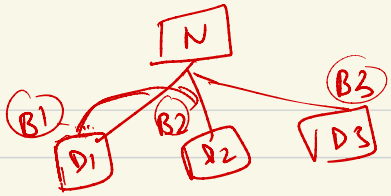
```
class Car Car1 = new CAR();
```

```
Car1.name = "Hyndai"
Car1.model = "SE1"
Car1.Price = 100K
```

Car1 → memory
Car2 →
Car3 →

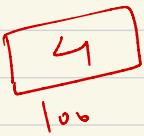
Car2
Car3

Car1.name




Data flow over Network ⇒ Serialized data

int a = 4;




Deserialize

↓

Converting something to its original form or int original datatype

variable
object
data

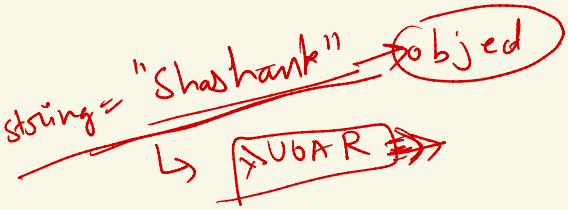
Serialized form (Converting original data or object or

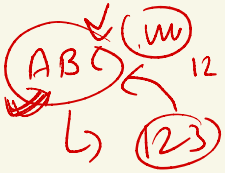
data types int Bytes)

## Advantage of Serialization

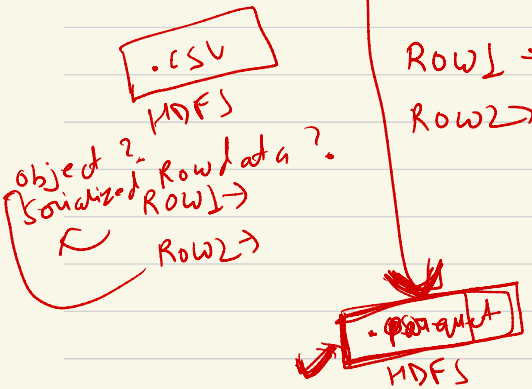
- Data in bytes
- Fast transmission over network
- Less data size

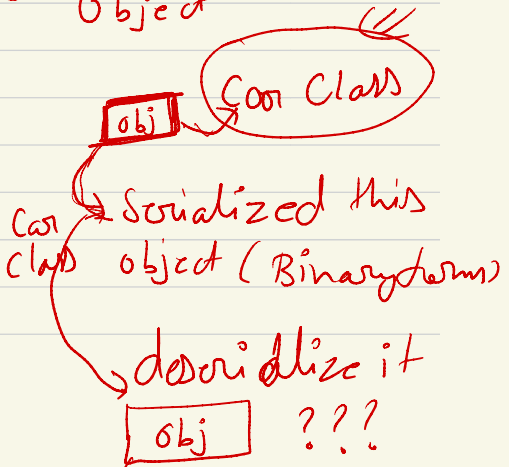
Select * from employee;




Select * from employee;



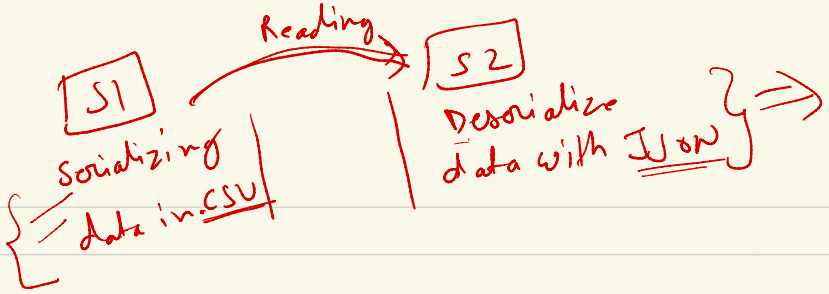
Class Animals
{
string type;
}

car class ??!

Class Animal = ??!

query { JSON }


Java should have a mechanism or libraris which can deserialize it



Source

↓

sales.csv

Table for Parquet

stored as parquet

Load data

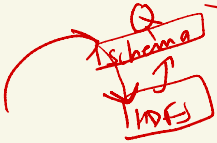
when I run

select * from pq_table;

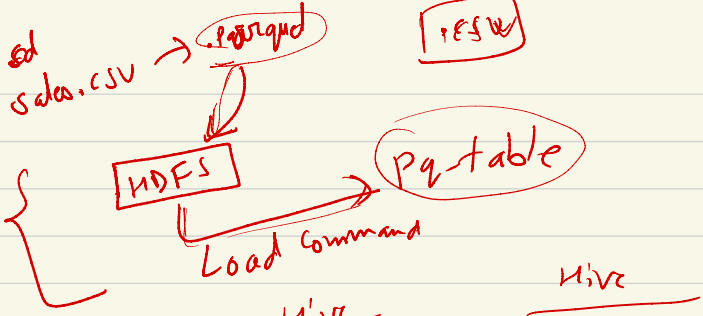
Schema on Read

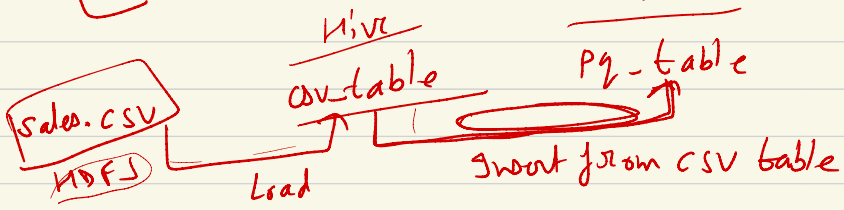
?

Schema on Write

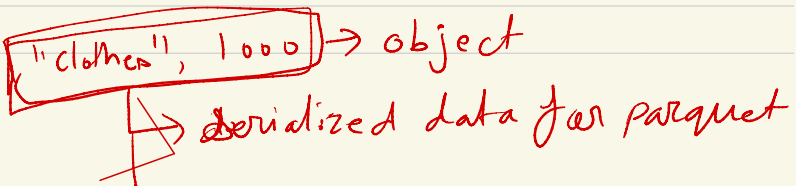


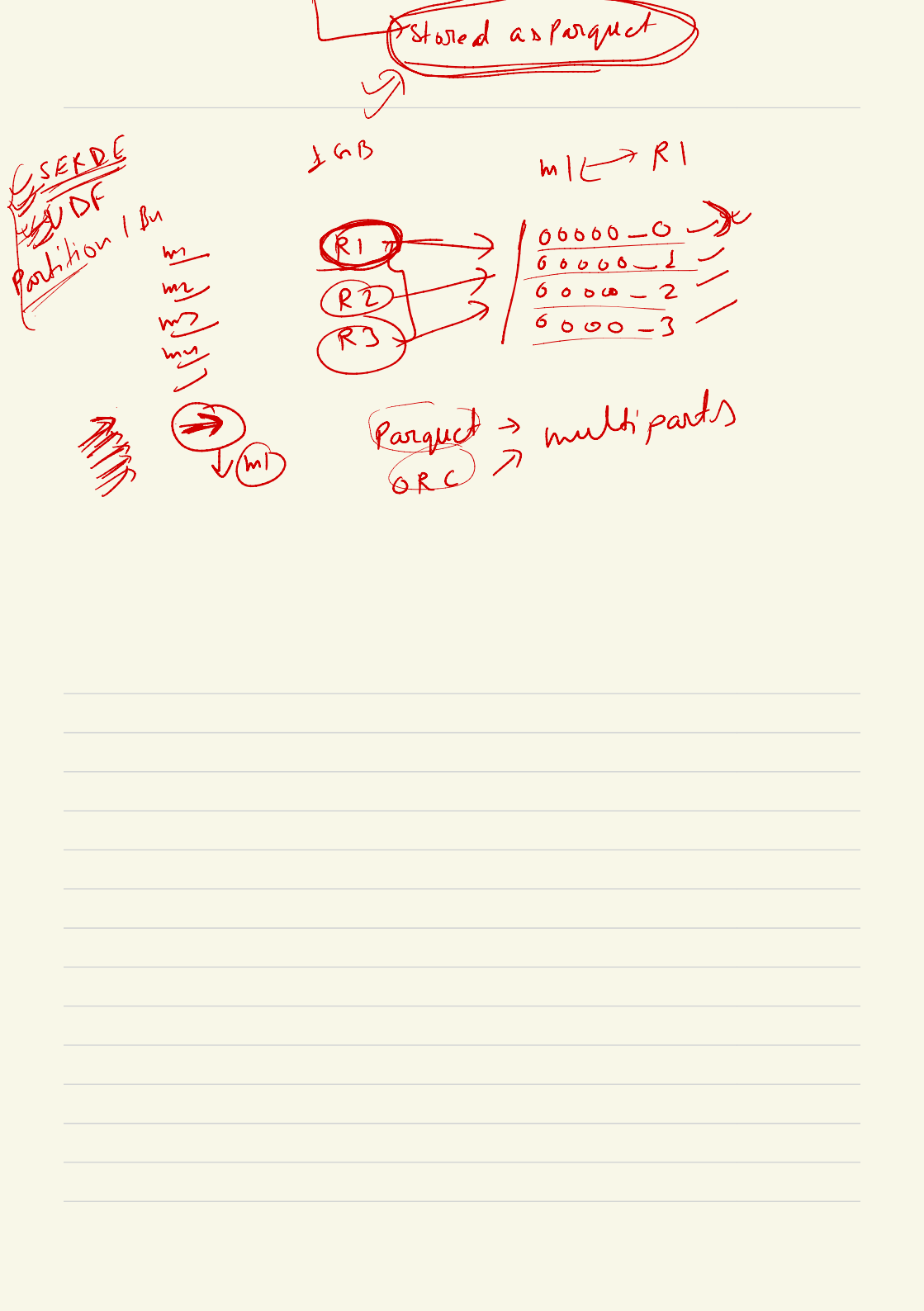

Hive was longing for Internal serialization and deserialization




from csv_table Insert Overwrite pq_table Select *;




stored as parquet
SERDE
UDF
Partition / Bu
1 GB
m1 → R1
R1
R2
R3
00000_0
00000_1
00000_2
00000_3
m1
Parquet
ORC
multi parts