



SPARK ZERO TO HERO

100+ PAGES

All about Data Engineering! →

Spark Tutorial

What is Spark ?

Spark is an Open Source Unified Computing Engine with set of libraries for parallel data processing on Computer Cluster.

It supports widely used Programming languages such as:

1. Scala
2. Python
3. Java
4. R

It processes data in memory(RAM) which makes it 100 times faster than traditional Hadoop Map Reduce.

Spark Components

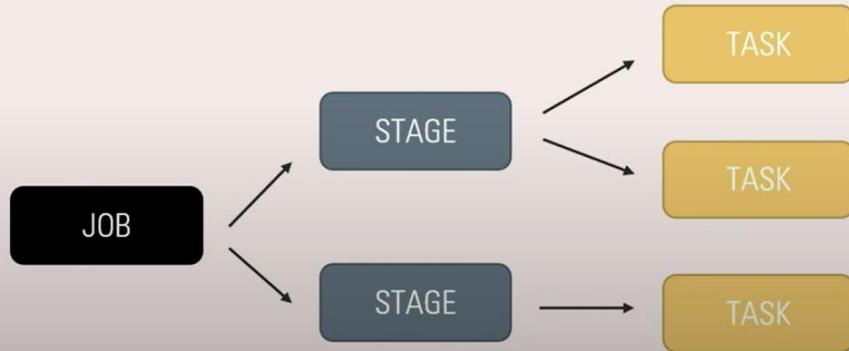
Following represents Spark Components on High level

1. Low Level API – RDD & Distributed Variables
2. Structured API – DataFrames, Datasets and SQL
3. Libraries and Ecosystem, Structured Streaming and Advanced Analytics



How Spark works ?

1. What are Driver and Executors ?
2. What is JOBS, Stages & Tasks ?



Driver:-

1. Heart of the Spark Application
2. Manages the information and state of executors
3. Analyses, distributes and Schedules the work on executors

Executor:-

1. Execute the code
2. Report the status of execution to driver

A User assigns a job to driver and driver in turn analyses distribution and breaks down the job into Stages and Tasks. And it assigns it to the executors. Here executors are basically a JVM process which runs in the cluster of machines and it consists of cores.

NOTE:-

1. Each task can only work on 1 partition of data at a time.
2. Tasks can execute in parallel.

- 3. Executor are JVM processes running on cluster machines.**
- 4. Executors hosts cores and each core can run 1 task at a time.**

What is Partition ?

To allow every executors to work in parallel, Spark breaks down the data into chunks called partitions.

So, from our last example – the bag of marble contained pouches of marble, which can be considered as Partition.

What is Transformation ?

The instruction or code to modify and transform data is known as Transformation.

Eg. select, where, groupBy etc.

Transformation help is building the logical plan.

Two types:

1. Narrow Transformation
2. Wide Transformation

Narrow Transformation:-

After applying transformation each partition contribute to at-most one partition.

Wide Transformation:-

After applying transformation if one partition contribute to more than one partition. This type of transformations lead to data shuffle.

What are Actions ?

To trigger the execution we need to call an Action.

This basically executes the plan created by Transformation

Actions are of three types:

1. View data in console
2. Collect data to native language
3. Write data to output data sources

Spark prefers Lazy Evaluation

Spark will wait till last moment to execute the graph of computation.

This allows Spark to optimize, plan and use the resources properly for execution.

What is Spark Session?

1. The Driver Process is known as Spark Session
2. It is the entry point for a Spark execution
3. The Spark Session instance executes the code in the cluster.
4. And the relation is one-to-one, i.e. For 1 Spark Application we will have 1 Spark Session instance.

Structured API - DataFrames

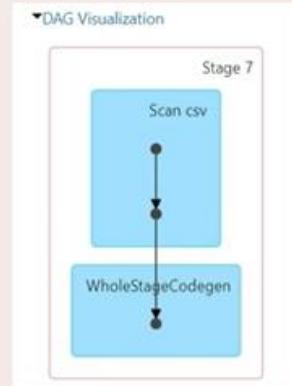
1. DataFrame is the most common Structured API, represented like a table
2. The table is represented in form on Rows and Columns
3. DataFrame has schema, which is the metadata for the columns
4. Data in DataFrames are in Partitions.
5. DataFrames are immutable

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000.0	2015-01-01
002	101	Jane Smith	25	Female	45000.0	2016-02-15
003	102	Bob Brown	35	Male	55000.0	2014-05-01
004	102	Alice Lee	28	Female	48000.0	2017-09-30
005	103	Jack Chan	40	Male	60000.0	2013-04-01
006	103	Jill Wong	32	Female	52000.0	2018-07-01
007	101	James Johnson	42	Male	70000.0	2012-03-15
008	102	Kate Kim	29	Female	51000.0	2019-10-01
009	103	Tom Tan	33	Male	58000.0	2016-06-01
010	104	Lisa Lee	27	Female	47000.0	2018-08-01

How Spark works on Data Partitions?

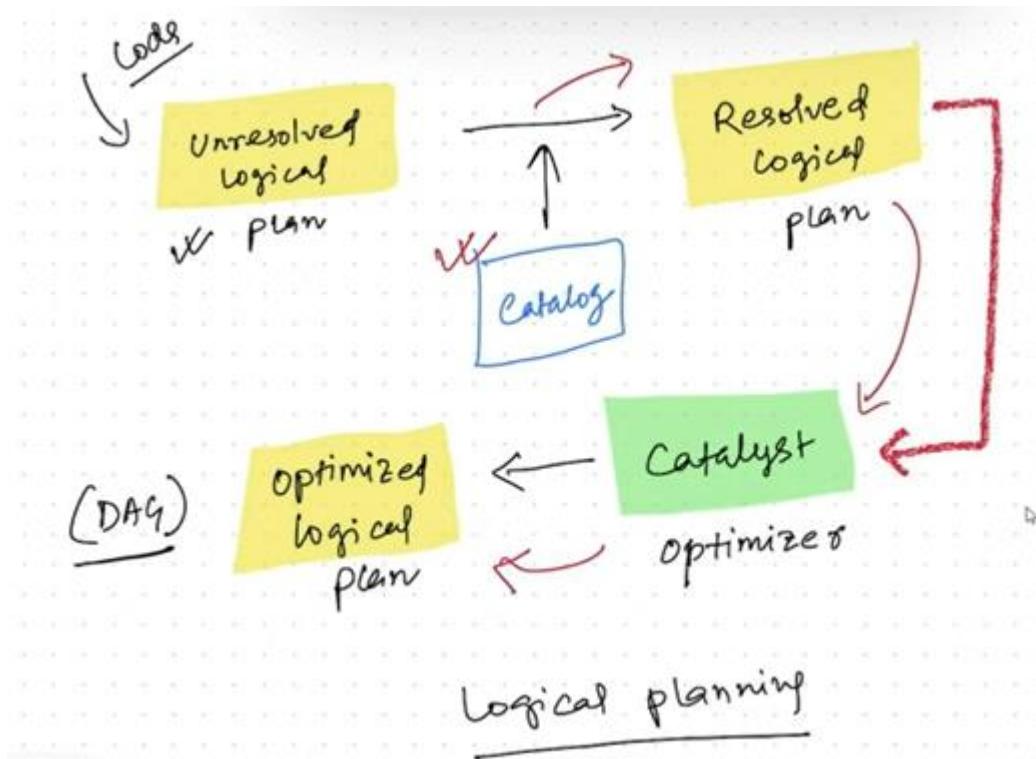
- Spark distributes the data in form of partitions to the Cluster.

Structured API Execution Plan

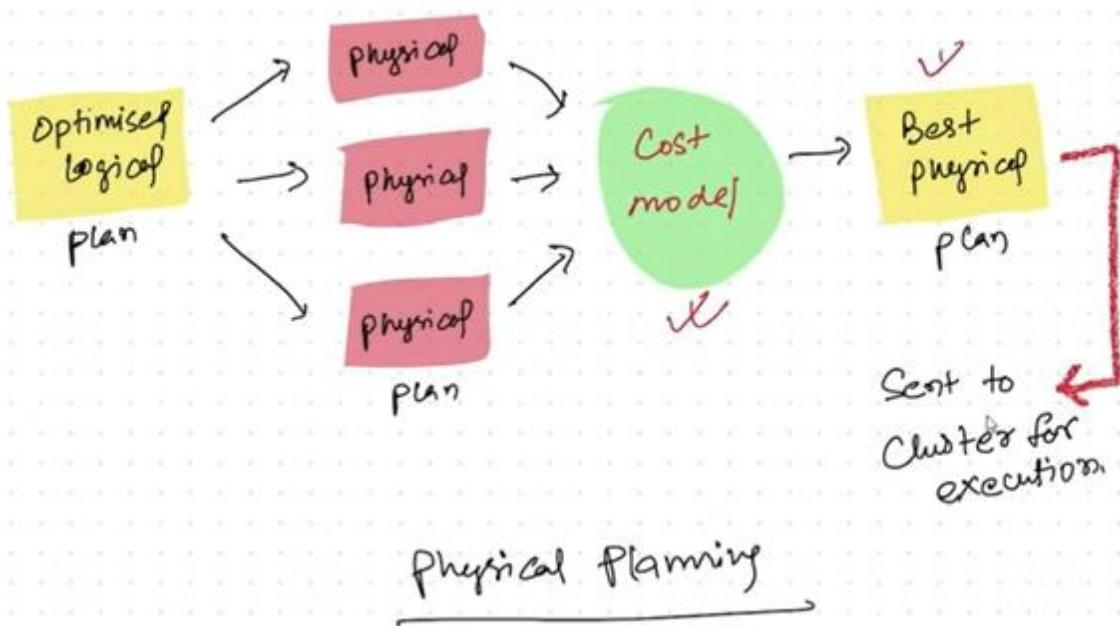


1. Logical Planning
2. Physical Planning

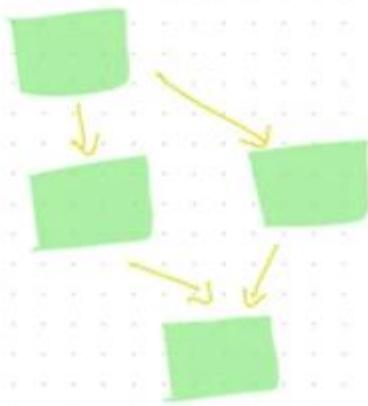
Logical Planning is the 1st Phase of Execution Planning



Physical Planning is the 2nd and Final Phase of Execution Planning



DAG
↓
Directed Acyclic Graph



Directed Acyclic Graph(DAG)- A DAG represents the logical execution plan of a Spark Job.

Spark Hands On

1. SparkSession
2. Create DataFrame & Columns
3. Spark UI
4. Introduction to Transformation & Actions
5. PySpark Shell

Example- Generate the EMP dataframe and filter EMP salary>50000. Write the output in CSV format.

Spark Session Object is the entry point for our Spark Application

Spark Session object name -We are using generic name “spark” for better understanding.

Local([*]) is to run Spark locally with as many worker threads as logical cores on your machine.

In [1]:

```
# Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Spark Introduction")
    .master("local[*]")
    .getOrCreate()
)
```

In [2]:

```
spark
```

Out[2]:

SparkSession - in-memory

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Spark Introduction

In [3]:

```
# Emp Data & Schema

emp_data = [
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"],
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],
    ["018", "104", "Nancy Liu", "29", "Female", "50000", "2017-06-01"],
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]
]

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date str
```

```
In [4]: # Create emp DataFrame
emp = spark.createDataFrame(data=emp_data, schema=emp_schema)

In [5]: # Check number of partitions
emp.rdd.getNumPartitions()

Out[5]: 8

In [6]: # Show data (ACTION)
emp.show()

+-----+-----+-----+-----+-----+
|employee_id|department_id|      name|age|gender|salary| hire_date|
+-----+-----+-----+-----+-----+
|     001|       101| John Doe| 30|  Male| 50000|2015-01-01|
|     002|       101| Jane Smith| 25|Female| 45000|2016-02-15|
|     003|       102| Bob Brown| 35|  Male| 55000|2014-05-01|
|     004|       102| Alice Lee| 28|Female| 48000|2017-09-30|
|     005|       103| Jack Chan| 40|  Male| 60000|2013-04-01|
|     006|       103| Jill Wong| 32|Female| 52000|2018-07-01|
|     007| 101|James Johnson| 42|  Male| 70000|2012-03-15|
|     008|       102| Kate Kim| 29|Female| 51000|2019-10-01|
|     009|       103| Tom Tan| 33|  Male| 58000|2016-06-01|
|     010|       104| Lisa Lee| 27|Female| 47000|2018-08-01|
|     011|       104| David Park| 38|  Male| 65000|2015-11-01|
|     012|       105| Susan Chen| 31|Female| 54000|2017-02-15|
|     013|       106| Brian Kim| 45|  Male| 75000|2011-07-01|
|     014|       107| Emily Lee| 26|Female| 46000|2019-01-01|
|     015|       106| Michael Lee| 37|  Male| 63000|2014-09-30|
|     016|       107| Kelly Zhang| 30|Female| 49000|2018-04-01|
|     017|       105| George Wang| 34|  Male| 57000|2016-03-15|
|     018|       104| Nancy Liu| 29|Female| 50000|2017-06-01|
|     019|       103| Steven Chen| 36|  Male| 62000|2015-08-01|
|     020|       102| Grace Kim| 32|Female| 53000|2018-11-01|
+-----+-----+-----+-----+-----+
```

```
In [7]: # Write our first Transformation (EMP salary > 50000)
emp_final = emp.where("salary > 50000")

In [8]: # Validate number of Partitions
emp_final.rdd.getNumPartitions()

Out[8]: 8

In [9]: # Write data as CSV output (ACTION)
emp_final.write.format("csv").save("data/output/1/emp.csv")
```

If you want to change the spark session object name then this code you can use: (eg, new_object_name = spark.getActiveSession())

Spark Hands On

1. DataFrame Columns
2. DataFrame Rows
3. DataFrame Schema
4. Structured Transformations – select, expr, selectExpr, cast

Schema for DataFrame- Schema is the metadata that defines the name and the type of the column.

```
In [6]: # Schema for emp  
emp.schema  
  
Out[6]: StructType([StructField('employee_id', StringType(), True), StructField('department_id', StringType(), True), StructField('name', StringType(), True), StructField('age', StringType(), True), StructField('gender', StringType(), True), StructField('salary', StringType(), True), StructField('hire_date', StringType(), True)])  
  
In [8]: # Small Example for Schema  
from pyspark.sql.types import StructType, StructField, StringType, IntegerType  
schema_string = "name string, age int"  
  
schema_spark = StructType([  
    StructField("name", StringType(), True),  
    StructField("age", IntegerType(), True)  
])
```

DataFrame is divided into two parts: Rows & Columns

You need DataFrame to manipulate the columns

```
In [18]: # Columns and expression  
from pyspark.sql.functions import col, expr  
  
emp["salary"]  
  
Out[18]: Column<'salary'>  
  
In [20]: # SELECT columns  
# select employee_id, name, age, salary from emp  
  
emp_filtered = emp.select(col("employee_id"), expr("name"), emp.age, emp.salary)
```

```
In [21]: # SHOW Dataframe (ACTION)
```

```
emp_filtered.show()
```

employee_id	name	age	salary
001	John Doe	30	50000
002	Jane Smith	25	45000
003	Bob Brown	35	55000
004	Alice Lee	28	48000
005	Jack Chan	40	60000
006	Jill Wong	32	52000
007	James Johnson	42	70000
008	Kate Kim	29	51000
009	Tom Tan	33	58000
010	Lisa Lee	27	47000
011	David Park	38	65000
012	Susan Chen	31	54000
013	Brian Kim	45	75000
014	Emily Lee	26	46000
015	Michael Lee	37	63000
016	Kelly Zhang	30	49000
017	George Wang	34	57000
018	Nancy Liu	29	50000
019	Steven Chen	36	62000
020	Grace Kim	32	53000

```
In [22]:
```

```
# Using expr for select
# select employee_id as emp_id, name, cast(age as int) as age, salary from emp_filtered
emp_casted = emp_filtered.select(expr("employee_id as emp_id"), emp.name, expr("cast(age as int) as age"), emp.salary)
```

```
In [23]:
```

```
# SHOW Dataframe (ACTION)
```

```
emp_casted.show()
```

emp_id	name	age	salary
001	John Doe	30	50000
002	Jane Smith	25	45000
003	Bob Brown	35	55000
004	Alice Lee	28	48000
005	Jack Chan	40	60000
006	Jill Wong	32	52000
007	James Johnson	42	70000
008	Kate Kim	29	51000
009	Tom Tan	33	58000
010	Lisa Lee	27	47000
011	David Park	38	65000
012	Susan Chen	31	54000
013	Brian Kim	45	75000
014	Emily Lee	26	46000
015	Michael Lee	37	63000
016	Kelly Zhang	30	49000
017	George Wang	34	57000
018	Nancy Liu	29	50000
019	Steven Chen	36	62000
020	Grace Kim	32	53000

```
In [25]: emp_casted_1 = emp_filtered.selectExpr("employee_id as emp_id", "name", "cast(age as int) as age", "salary")
```

```
In [26]: emp_casted_1.show()
```

```
+-----+-----+-----+
|emp_id|      name|age|salary|
+-----+-----+-----+
| 001| John Doe| 30| 50000|
| 002| Jane Smith| 25| 45000|
| 003| Bob Brown| 35| 55000|
| 004| Alice Lee| 28| 48000|
| 005| Jack Chan| 40| 60000|
| 006| Jill Wong| 32| 52000|
| 007|James Johnson| 42| 70000|
| 008| Kate Kim| 29| 51000|
| 009| Tom Tan| 33| 58000|
| 010| Lisa Lee| 27| 47000|
| 011| David Park| 38| 65000|
| 012| Susan Chen| 31| 54000|
| 013| Brian Kim| 45| 75000|
| 014| Emily Lee| 26| 46000|
| 015| Michael Lee| 37| 63000|
| 016| Kelly Zhang| 30| 49000|
| 017| George Wang| 34| 57000|
| 018| Nancy Liu| 29| 50000|
| 019| Steven Chen| 36| 62000|
| 020| Grace Kim| 32| 53000|
+-----+-----+-----+
```

```
In [24]: emp_casted.printSchema()
```

```
root
 |-- emp_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- age: integer (nullable = true)
 |-- salary: string (nullable = true)
```

```
In [27]: # Filter emp based on Age > 30
# select emp_id, name, age, salary from emp_casted where age > 30
```

```
emp_final = emp_casted.select("emp_id", "name", "age", "salary").where("age > 30")
```

```
In [27]: # Filter emp based on Age > 30
# select emp_id, name, age, salary from emp_casted where age > 30
```

```
emp_final = emp_casted.select("emp_id", "name", "age", "salary").where("age > 30")
```

```
In [28]: # SHOW Dataframe (ACTION)
```

```
emp_final.show()
```

```
+-----+-----+-----+
|emp_id|      name|age|salary|
+-----+-----+-----+
| 003| Bob Brown| 35| 55000|
| 005| Jack Chan| 40| 60000|
| 006| Jill Wong| 32| 52000|
| 007|James Johnson| 42| 70000|
| 009| Tom Tan| 33| 58000|
| 011| David Park| 38| 65000|
| 012| Susan Chen| 31| 54000|
| 013| Brian Kim| 45| 75000|
| 015| Michael Lee| 37| 63000|
| 017| George Wang| 34| 57000|
| 019| Steven Chen| 36| 62000|
| 020| Grace Kim| 32| 53000|
+-----+-----+-----+
```

```
In [29]: # Write the data back as CSV (ACTION)
emp_final.write.format("csv").save("data/output/2/emp.csv")

In [30]: # Bonus TIP
schema_str = "name string, age int"
from pyspark.sql.types import _parse_datatype_string
schema_spark = _parse_datatype_string(schema_str)
schema_spark
```

Out[30]: StructType([StructField('name', StringType(), True), StructField('age', IntegerType(), True)])

The function `_parse_datatype_string` in PySpark is a private method used internally to parse a string representation of a data type into a PySpark DataType object. It is primarily utilized when working with schema definitions or manipulating data types programmatically.

Spark Hands On

1. Adding Columns
2. Using Literals/Static values
3. Renaming Columns
4. Removing Columns
5. Filtering and LIMIT for DataFrame
6. Structured Transformations – `withColumn`, `withColumnRenamed`, `lit`

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Basic Transformation - II")
    .master("local[*]")
    .getOrCreate()
)
spark
```

Out[1]: SparkSession - in-memory

SparkContext	v3.3.0
Spark UI	local[*]
Version	v3.3.0
Master	local[*]
AppName	Basic Transformation - II

```
In [2]: # Emp Data & Schema

emp_data = [
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"],
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],
    ["018", "104", "Nancy Liu", "29", "Female", "50000", "2017-06-01"],
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]
]

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date str"

In [3]: # Create emp DataFrame

emp = spark.createDataFrame(data=emp_data, schema=emp_schema)
```

```
In [4]: # Show emp dataframe (ACTION)

emp.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29	Female	50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [5]: # Print Schema
```

```
emp.printSchema()
```

```
root
|-- employee_id: string (nullable = true)
|-- department_id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: string (nullable = true)
|-- gender: string (nullable = true)
|-- salary: string (nullable = true)
|-- hire_date: string (nullable = true)
```

```
In [8]: # Casting Column
```

```
# select employee_id, name, age, cast(salary as double) as salary from emp
from pyspark.sql.functions import col, cast

emp_casted = emp.select("employee_id", "name", "age", col("salary").cast("double"))
```

```
In [9]: emp_casted.printSchema()
```

```
root
|-- employee_id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: string (nullable = true)
|-- salary: double (nullable = true)
```

```
In [10]: # Adding Columns
```

```
# select employee_id, name, age, salary, (salary * 0.2) as tax from emp_casted

emp_taxed = emp_casted.withColumn("tax", col("salary") * 0.2)
```

```
In [11]: emp_taxed.show()
```

employee_id	name	age	salary	tax
001	John Doe	30	50000.0	10000.0
002	Jane Smith	25	45000.0	9000.0
003	Bob Brown	35	55000.0	11000.0
004	Alice Lee	28	48000.0	9600.0
005	Jack Chan	40	60000.0	12000.0
006	Jill Wong	32	52000.0	10400.0
007	James Johnson	42	70000.0	14000.0
008	Kate Kim	29	51000.0	10200.0
009	Tom Tan	33	58000.0	11600.0
010	Lisa Lee	27	47000.0	9400.0
011	David Park	38	65000.0	13000.0
012	Susan Chen	31	54000.0	10800.0
013	Brian Kim	45	75000.0	15000.0
014	Emily Lee	26	46000.0	9200.0
015	Michael Lee	37	63000.0	12600.0
016	Kelly Zhang	30	49000.0	9800.0
017	George Wang	34	57000.0	11400.0
018	Nancy Liu	29	50000.0	10000.0
019	Steven Chen	36	62000.0	12400.0
020	Grace Kim	32	53000.0	10600.0

```
In [12]: # Literals
# select employee_id, name, age, salary, tax, 1 as columnOne, 'two' as columnTwo from emp_taxed
from pyspark.sql.functions import lit

emp_new_cols = emp_taxed.withColumn("columnOne", lit(1)).withColumn("columnTwo", lit('two'))
```

```
In [13]: emp_new_cols.show()
```

employee_id	name	age	salary	tax	columnOne	columnTwo
001	John Doe	30	50000.0	10000.0	1	two
002	Jane Smith	25	45000.0	9000.0	1	two
003	Bob Brown	35	55000.0	11000.0	1	two
004	Alice Lee	28	48000.0	9600.0	1	two
005	Jack Chan	40	60000.0	12000.0	1	two
006	Jill Wong	32	52000.0	10400.0	1	two
007	James Johnson	42	70000.0	14000.0	1	two
008	Kate Kim	29	51000.0	10200.0	1	two
009	Tom Tan	33	58000.0	11600.0	1	two
010	Lisa Lee	27	47000.0	9400.0	1	two
011	David Park	38	65000.0	13000.0	1	two
012	Susan Chen	31	54000.0	10800.0	1	two
013	Brian Kim	45	75000.0	15000.0	1	two
014	Emily Lee	26	46000.0	9200.0	1	two
015	Michael Lee	37	63000.0	12600.0	1	two
016	Kelly Zhang	30	49000.0	9800.0	1	two
017	George Wang	34	57000.0	11400.0	1	two
018	Nancy Liu	29	50000.0	10000.0	1	two
019	Steven Chen	36	62000.0	12400.0	1	two
020	Grace Kim	32	53000.0	10600.0	1	two

```
In [14]: # Renaming Columns
# select employee_id as emp_id, name, age, salary, tax, columnOne, columnTwo from emp_new_cols

emp_1 = emp_new_cols.withColumnRenamed("employee_id", "emp_id")
```

```
In [15]: emp_1.show()
```

emp_id	name	age	salary	tax	columnOne	columnTwo
001	John Doe	30	50000.0	10000.0	1	two
002	Jane Smith	25	45000.0	9000.0	1	two
003	Bob Brown	35	55000.0	11000.0	1	two
004	Alice Lee	28	48000.0	9600.0	1	two
005	Jack Chan	40	60000.0	12000.0	1	two
006	Jill Wong	32	52000.0	10400.0	1	two
007	James Johnson	42	70000.0	14000.0	1	two
008	Kate Kim	29	51000.0	10200.0	1	two
009	Tom Tan	33	58000.0	11600.0	1	two
010	Lisa Lee	27	47000.0	9400.0	1	two
011	David Park	38	65000.0	13000.0	1	two
012	Susan Chen	31	54000.0	10800.0	1	two
013	Brian Kim	45	75000.0	15000.0	1	two
014	Emily Lee	26	46000.0	9200.0	1	two
015	Michael Lee	37	63000.0	12600.0	1	two
016	Kelly Zhang	30	49000.0	9800.0	1	two
017	George Wang	34	57000.0	11400.0	1	two
018	Nancy Liu	29	50000.0	10000.0	1	two
019	Steven Chen	36	62000.0	12400.0	1	two
020	Grace Kim	32	53000.0	10600.0	1	two

```
In [18]: # Column names with Spaces
# select employee_id as emp_id, name, age, salary, tax, columnOne, columnTwo as `Column Two` from emp_new_cols

emp_2 = emp_new_cols.withColumnRenamed("columnTwo", "Column Two")
```

```
In [19]: emp_2.show()
```

employee_id	name	age	salary	tax	columnOne	Column Two
001	John Doe	30	50000.0	10000.0	1	two
002	Jane Smith	25	45000.0	9000.0	1	two
003	Bob Brown	35	55000.0	11000.0	1	two
004	Alice Lee	28	48000.0	9600.0	1	two
005	Jack Chan	40	60000.0	12000.0	1	two
006	Jill Wong	32	52000.0	10400.0	1	two
007	James Johnson	42	70000.0	14000.0	1	two
008	Kate Kim	29	51000.0	10200.0	1	two
009	Tom Tan	33	58000.0	11600.0	1	two
010	Lisa Lee	27	47000.0	9400.0	1	two
011	David Park	38	65000.0	13000.0	1	two
012	Susan Chen	31	54000.0	10800.0	1	two
013	Brian Kim	45	75000.0	15000.0	1	two
014	Emily Lee	26	46000.0	9200.0	1	two
015	Michael Lee	37	63000.0	12600.0	1	two
016	Kelly Zhang	30	49000.0	9800.0	1	two
017	George Wang	34	57000.0	11400.0	1	two
018	Nancy Liu	29	50000.0	10000.0	1	two
019	Steven Chen	36	62000.0	12400.0	1	two
020	Grace Kim	32	53000.0	10600.0	1	two

```
In [22]: # Remove Column

emp_dropped = emp_new_cols.drop("columnTwo", "columnOne")
```

```
In [23]: emp_dropped.show()
```

employee_id	name	age	salary	tax
001	John Doe	30	50000.0	10000.0
002	Jane Smith	25	45000.0	9000.0
003	Bob Brown	35	55000.0	11000.0
004	Alice Lee	28	48000.0	9600.0
005	Jack Chan	40	60000.0	12000.0
006	Jill Wong	32	52000.0	10400.0
007	James Johnson	42	70000.0	14000.0
008	Kate Kim	29	51000.0	10200.0
009	Tom Tan	33	58000.0	11600.0
010	Lisa Lee	27	47000.0	9400.0
011	David Park	38	65000.0	13000.0
012	Susan Chen	31	54000.0	10800.0
013	Brian Kim	45	75000.0	15000.0
014	Emily Lee	26	46000.0	9200.0
015	Michael Lee	37	63000.0	12600.0
016	Kelly Zhang	30	49000.0	9800.0
017	George Wang	34	57000.0	11400.0
018	Nancy Liu	29	50000.0	10000.0
019	Steven Chen	36	62000.0	12400.0
020	Grace Kim	32	53000.0	10600.0

```
In [24]: # Filter data
# select employee_id as emp_id, name, age, salary, tax, columnOne from emp_col_dropped where tax > 1000
emp_filtered = emp_dropped.where("tax > 10000")
```

```
In [25]: emp_filtered.show()
```

employee_id	name	age	salary	tax
003	Bob Brown	35	55000.0	11000.0
005	Jack Chan	40	60000.0	12000.0
006	Jill Wong	32	52000.0	10400.0
007	James Johnson	42	70000.0	14000.0
008	Kate Kim	29	51000.0	10200.0
009	Tom Tan	33	58000.0	11600.0
011	David Park	38	65000.0	13000.0
012	Susan Chen	31	54000.0	10800.0
013	Brian Kim	45	75000.0	15000.0
015	Michael Lee	37	63000.0	12600.0
017	George Wang	34	57000.0	11400.0
019	Steven Chen	36	62000.0	12400.0
020	Grace Kim	32	53000.0	10600.0

```
In [26]: # LIMIT data
# select employee_id as emp_id, name, age, salary, tax, columnOne from emp_filtered Limit 5
emp_limit = emp_filtered.limit(5)
```

```
In [28]: # Show data
emp_limit.show(2)
```

employee_id	name	age	salary	tax
003	Bob Brown	35	55000.0	11000.0
005	Jack Chan	40	60000.0	12000.0

only showing top 2 rows

```
In [ ]: # Bonus TIP
# Add multiple columns

columns = {
    "tax" : col("salary") * 0.2 ,
    "oneNumber" : lit(1),
    "columnTwo" : lit("two")
}

emp_final = emp.withColumns(columns)
```

```
In [ ]: emp_final.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	tax	oneNumber	columnTwo
001	101	John Doe	30	Male	50000	2015-01-01	10000.0	1	two
002	101	Jane Smith	25	Female	45000	2016-02-15	9000.0	1	two
003	102	Bob Brown	35	Male	55000	2014-05-01	11000.0	1	two
004	102	Alice Lee	28	Female	48000	2017-09-30	9600.0	1	two
005	103	Jack Chan	40	Male	60000	2013-04-01	12000.0	1	two
006	103	Jill Wong	32	Female	52000	2018-07-01	10400.0	1	two
007	101	James Johnson	42	Male	70000	2012-03-15	14000.0	1	two
008	102	Kate Kim	29	Female	51000	2019-10-01	10200.0	1	two
009	103	Tom Tan	33	Male	58000	2016-06-01	11600.0	1	two
010	104	Lisa Lee	27	Female	47000	2018-08-01	9400.0	1	two
011	104	David Park	38	Male	65000	2015-11-01	13000.0	1	two
012	105	Susan Chen	31	Female	54000	2017-02-15	10800.0	1	two
013	106	Brian Kim	45	Male	75000	2011-07-01	15000.0	1	two
014	107	Emily Lee	26	Female	46000	2019-01-01	9200.0	1	two
015	106	Michael Lee	37	Male	63000	2014-09-30	12600.0	1	two
016	107	Kelly Zhang	30	Female	49000	2018-04-01	9800.0	1	two
017	105	George Wang	34	Male	57000	2016-03-15	11400.0	1	two
018	104	Nancy Liu	29	Female	50000	2017-06-01	10000.0	1	two
019	103	Steven Chen	36	Male	62000	2015-08-01	12400.0	1	two
020	102	Grace Kim	32	Female	53000	2018-11-01	10600.0	1	two

Adding or Overriding columns :- Spark provides `withColumn()` to create new column or override existing column.

Adding static values columns :- Spark provides `lit()` to create a distributed static column.

Renaming existing columns :- Spark provides `withColumnRenamed()` to the same.

Multiple ways for Rename :- We can use `expr()` or `selectExpr()` as well for renaming columns (eg, `selectExpr(employee_id as emp_id)`)

Downstream system :- Any system which is dependent on your output.

Note- It is never recommended to use spaces in the column names.

Remove columns from DataFrame :- use `drop()`.

Limit : In case we need to get only few records and not full dataset.

Spark Hands On

1. Working with String Data
 1. Case When, Regex_Replace etc
2. Working with Dates
 1. `to_date`, `current_date`, `current_timestamp`
3. Working with NULL values
 1. `nvl`, `na.drop`, `na.fill`

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Working with Strings & Dates")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Working with Strings & Dates

In [2]: # Emp Data & Schema

```
emp_data = [
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"],
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],
    ["018", "104", "Nancy Liu", "29", "", "50000", "2017-06-01"],
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]
]

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date str
```

In [3]: # Create emp DataFrame

```
emp = spark.createDataFrame(data=emp_data, schema=emp_schema)
```

```
In [4]: # Show emp dataframe (ACTION)
```

```
emp.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29		50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [5]: # Print Schema
```

```
emp.printSchema()
```

```
root
|-- employee_id: string (nullable = true)
|-- department_id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: string (nullable = true)
|-- gender: string (nullable = true)
|-- salary: string (nullable = true)
|-- hire_date: string (nullable = true)
```

```
In [50]:
```

```
# Case When
# select employee_id, name, age, salary, gender,
# case when gender = 'Male' then 'M' when gender = 'Female' then 'F' else null end as new_gender, hire_date from emp
from pyspark.sql.functions import when, col, expr

emp_gender_fixed = emp.withColumn("new_gender", when(col("gender") == 'Male', 'M')
                                    .when(col("gender") == 'Female', 'F')
                                    .otherwise(None)
                                    )
emp_gender_fixed_1 = emp.withColumn("new_gender", expr("case when gender = 'Male' then 'M' when gender = 'Female' then 'F' else null end"))
```

```
ed_1 = emp.withColumn("new_gender", expr("case when gender = 'Male' then 'M' when gender = 'Female' then 'F' else null end"))
```

```
In [51]: emp_gender_fixed_1.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	new_gender
001	101	John Doe	30	Male	50000	2015-01-01	M
002	101	Jane Smith	25	Female	45000	2016-02-15	F
003	102	Bob Brown	35	Male	55000	2014-05-01	M
004	102	Alice Lee	28	Female	48000	2017-09-30	F
005	103	Jack Chan	40	Male	60000	2013-04-01	M
006	103	Jill Wong	32	Female	52000	2018-07-01	F
007	101	James Johnson	42	Male	70000	2012-03-15	M
008	102	Kate Kim	29	Female	51000	2019-10-01	F
009	103	Tom Tan	33	Male	58000	2016-06-01	M
010	104	Lisa Lee	27	Female	47000	2018-08-01	F
011	104	David Park	38	Male	65000	2015-11-01	M
012	105	Susan Chen	31	Female	54000	2017-02-15	F
013	106	Brian Kim	45	Male	75000	2011-07-01	M
014	107	Emily Lee	26	Female	46000	2019-01-01	F
015	106	Michael Lee	37	Male	63000	2014-09-30	M
016	107	Kelly Zhang	30	Female	49000	2018-04-01	F
017	105	George Wang	34	Male	57000	2016-03-15	M
018	104	Nancy Liu	29		50000	2017-06-01	null
019	103	Steven Chen	36	Male	62000	2015-08-01	M
020	102	Grace Kim	32	Female	53000	2018-11-01	F

```
In [12]: # Replace in Strings  
# select employee_id, name, replace(name, 'J', 'Z') as new_name, age, salary, gender, new_gender, hire_date from emp_gender_j  
from pyspark.sql.functions import regexp_replace
```

```
emp_name_fixed = emp_gender_fixed.withColumn("new_name", regexp_replace(col("name"), "J", "Z"))
```

```
In [13]: emp_name_fixed.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	new_gender	new_name
001	101	John Doe	30	Male	50000	2015-01-01	M	Zohn Doe
002	101	Jane Smith	25	Female	45000	2016-02-15	F	Zane Smith
003	102	Bob Brown	35	Male	55000	2014-05-01	M	Bob Brown
004	102	Alice Lee	28	Female	48000	2017-09-30	F	Alice Lee
005	103	Jack Chan	40	Male	60000	2013-04-01	M	Zack Chan
006	103	Jill Wong	32	Female	52000	2018-07-01	F	Zill Wong
007	101	James Johnson	42	Male	70000	2012-03-15	M	Zames Johnson
008	102	Kate Kim	29	Female	51000	2019-10-01	F	Kate Kim
009	103	Tom Tan	33	Male	58000	2016-06-01	M	Tom Tan
010	104	Lisa Lee	27	Female	47000	2018-08-01	F	Lisa Lee
011	104	David Park	38	Male	65000	2015-11-01	M	David Park
012	105	Susan Chen	31	Female	54000	2017-02-15	F	Susan Chen
013	106	Brian Kim	45	Male	75000	2011-07-01	M	Brian Kim
014	107	Emily Lee	26	Female	46000	2019-01-01	F	Emily Lee
015	106	Michael Lee	37	Male	63000	2014-09-30	M	Michael Lee
016	107	Kelly Zhang	30	Female	49000	2018-04-01	F	Kelly Zhang
017	105	George Wang	34	Male	57000	2016-03-15	M	George Wang
018	104	Nancy Liu	29		50000	2017-06-01	null	Nancy Liu
019	103	Steven Chen	36	Male	62000	2015-08-01	M	Steven Chen
020	102	Grace Kim	32	Female	53000	2018-11-01	F	Grace Kim

```
In [14]: # Convert Date
# select *, to_date(hire_date, 'YYYY-MM-DD') as hire_date from emp_name_fixed
from pyspark.sql.functions import to_date

emp_date_fix = emp_name_fixed.withColumn("hire_date", to_date(col("hire_date"), 'yyyy-MM-dd'))
```

```
In [16]: emp_date_fix.printSchema()
```

```
root
 |-- employee_id: string (nullable = true)
 |-- department_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: string (nullable = true)
 |-- hire_date: date (nullable = true)
 |-- new_gender: string (nullable = true)
 |-- new_name: string (nullable = true)
```

```
In [17]: # Add Date Columns
# Add current_date, current_timestamp, extract year from hire_date
from pyspark.sql.functions import current_date, current_timestamp

emp_dated = emp_date_fix.withColumn("date_now", current_date()).withColumn("timestamp_now", current_timestamp())
```

```
In [19]: emp_dated.show(truncate=False)
```

employee_id	department_id	name	age	gender	salary	hire_date	new_gender	new_name	date_now	timestamp_now
001 0.22104	101	John Doe	30	Male	50000	2015-01-01 M		Zohn Doe	2023-04-12 2023-04-12 07:57:3	
002 0.22104	101	Jane Smith	25	Female	45000	2016-02-15 F		Zane Smith	2023-04-12 2023-04-12 07:57:3	
003 0.22104	102	Bob Brown	35	Male	55000	2014-05-01 M		Bob Brown	2023-04-12 2023-04-12 07:57:3	
004 0.22104	102	Alice Lee	28	Female	48000	2017-09-30 F		Alice Lee	2023-04-12 2023-04-12 07:57:3	
005 0.22104	103	Jack Chan	40	Male	60000	2013-04-01 M		Zack Chan	2023-04-12 2023-04-12 07:57:3	
006 0.22104	103	Jill Wong	32	Female	52000	2018-07-01 F		Zill Wong	2023-04-12 2023-04-12 07:57:3	
007 0.22104	101	James Johnson	42	Male	70000	2012-03-15 M		James Johnson	2023-04-12 2023-04-12 07:57:3	
008 0.22104	102	Kate Kim	29	Female	51000	2019-10-01 F		Kate Kim	2023-04-12 2023-04-12 07:57:3	
009 0.22104	103	Tom Tan	33	Male	58000	2016-06-01 M		Tom Tan	2023-04-12 2023-04-12 07:57:3	
010 0.22104	104	Lisa Lee	27	Female	47000	2018-08-01 F		Lisa Lee	2023-04-12 2023-04-12 07:57:3	
011 0.22104	104	David Park	38	Male	65000	2015-11-01 M		David Park	2023-04-12 2023-04-12 07:57:3	
012 0.22104	105	Susan Chen	31	Female	54000	2017-02-15 F		Susan Chen	2023-04-12 2023-04-12 07:57:3	
013 0.22104	106	Brian Kim	45	Male	75000	2011-07-01 M		Brian Kim	2023-04-12 2023-04-12 07:57:3	
014 0.22104	107	Emily Lee	26	Female	46000	2019-01-01 F		Emily Lee	2023-04-12 2023-04-12 07:57:3	
015 0.22104	106	Michael Lee	37	Male	63000	2014-09-30 M		Michael Lee	2023-04-12 2023-04-12 07:57:3	
016 0.22104	107	Kelly Zhang	30	Female	49000	2018-04-01 F		Kelly Zhang	2023-04-12 2023-04-12 07:57:3	
017 0.22104	105	George Wang	34	Male	57000	2016-03-15 M		George Wang	2023-04-12 2023-04-12 07:57:3	
018 0.22104	104	Nancy Liu	29		50000	2017-06-01 null		Nancy Liu	2023-04-12 2023-04-12 07:57:3	
019 0.22104	103	Steven Chen	36	Male	62000	2015-08-01 M		Steven Chen	2023-04-12 2023-04-12 07:57:3	
020 0.22104	102	Grace Kim	32	Female	53000	2018-11-01 F		Grace Kim	2023-04-12 2023-04-12 07:57:3	

```
In [27]: # Drop Null gender records
emp_1 = emp_dated.na.drop()
```

```
In [23]: emp_1.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	new_gender	new_name	date_now	timestamp_no
0:... 001	101	John Doe	30	Male	50000	2015-01-01	M	Zohn Doe	2023-04-12	2023-04-12 08:0
0:... 002	101	Jane Smith	25	Female	45000	2016-02-15	F	Zane Smith	2023-04-12	2023-04-12 08:0
0:... 003	102	Bob Brown	35	Male	55000	2014-05-01	M	Bob Brown	2023-04-12	2023-04-12 08:0
0:... 004	102	Alice Lee	28	Female	48000	2017-09-30	F	Alice Lee	2023-04-12	2023-04-12 08:0
0:... 005	103	Jack Chan	40	Male	60000	2013-04-01	M	Zack Chan	2023-04-12	2023-04-12 08:0
0:... 006	103	Jill Wong	32	Female	52000	2018-07-01	F	Zill Wong	2023-04-12	2023-04-12 08:0
0:... 007	101	James Johnson	42	Male	70000	2012-03-15	M	Zames Zohnson	2023-04-12	2023-04-12 08:0
0:... 008	102	Kate Kim	29	Female	51000	2019-10-01	F	Kate Kim	2023-04-12	2023-04-12 08:0
0:... 009	103	Tom Tan	33	Male	58000	2016-06-01	M	Tom Tan	2023-04-12	2023-04-12 08:0
0:... 010	104	Lisa Lee	27	Female	47000	2018-08-01	F	Lisa Lee	2023-04-12	2023-04-12 08:0
0:... 011	104	David Park	38	Male	65000	2015-11-01	M	David Park	2023-04-12	2023-04-12 08:0
0:... 012	105	Susan Chen	31	Female	54000	2017-02-15	F	Susan Chen	2023-04-12	2023-04-12 08:0
0:... 013	106	Brian Kim	45	Male	75000	2011-07-01	M	Brian Kim	2023-04-12	2023-04-12 08:0
0:... 014	107	Emily Lee	26	Female	46000	2019-01-01	F	Emily Lee	2023-04-12	2023-04-12 08:0
0:... 015	106	Michael Lee	37	Male	63000	2014-09-30	M	Michael Lee	2023-04-12	2023-04-12 08:0
0:... 016	107	Kelly Zhang	30	Female	49000	2018-04-01	F	Kelly Zhang	2023-04-12	2023-04-12 08:0
0:... 017	105	George Wang	34	Male	57000	2016-03-15	M	George Wang	2023-04-12	2023-04-12 08:0
0:... 019	103	Steven Chen	36	Male	62000	2015-08-01	M	Steven Chen	2023-04-12	2023-04-12 08:0
0:... 020	102	Grace Kim	32	Female	53000	2018-11-01	F	Grace Kim	2023-04-12	2023-04-12 08:0

```
In [33]: # Fix Null values
# select *, nvl('new_gender', '0') as new_gender from emp_dated
from pyspark.sql.functions import coalesce, lit

emp_null_df = emp_dated.withColumn("new_gender", coalesce(col("new_gender"), lit("0")))
```

```
In [34]: emp_null_df.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	new_gender	new_name	date_now	timestamp_no
001	101	John Doe	30	Male	50000	2015-01-01	M	Zohn Doe	2023-04-12	2023-04-12 08:00
002	101	Jane Smith	25	Female	45000	2016-02-15	F	Zane Smith	2023-04-12	2023-04-12 08:00
003	102	Bob Brown	35	Male	55000	2014-05-01	M	Bob Brown	2023-04-12	2023-04-12 08:00
004	102	Alice Lee	28	Female	48000	2017-09-30	F	Alice Lee	2023-04-12	2023-04-12 08:00
005	103	Jack Chan	40	Male	60000	2013-04-01	M	Zack Chan	2023-04-12	2023-04-12 08:00
006	103	Jill Wong	32	Female	52000	2018-07-01	F	Zill Wong	2023-04-12	2023-04-12 08:00
007	101	James Johnson	42	Male	70000	2012-03-15	M	Zames Zohnson	2023-04-12	2023-04-12 08:00
008	102	Kate Kim	29	Female	51000	2019-10-01	F	Kate Kim	2023-04-12	2023-04-12 08:00
009	103	Tom Tan	33	Male	58000	2016-06-01	M	Tom Tan	2023-04-12	2023-04-12 08:00
010	104	Lisa Lee	27	Female	47000	2018-08-01	F	Lisa Lee	2023-04-12	2023-04-12 08:00
011	104	David Park	38	Male	65000	2015-11-01	M	David Park	2023-04-12	2023-04-12 08:00
012	105	Susan Chen	31	Female	54000	2017-02-15	F	Susan Chen	2023-04-12	2023-04-12 08:00
013	106	Brian Kim	45	Male	75000	2011-07-01	M	Brian Kim	2023-04-12	2023-04-12 08:00
014	107	Emily Lee	26	Female	46000	2019-01-01	F	Emily Lee	2023-04-12	2023-04-12 08:00
015	106	Michael Lee	37	Male	63000	2014-09-30	M	Michael Lee	2023-04-12	2023-04-12 08:00
016	107	Kelly Zhang	30	Female	49000	2018-04-01	F	Kelly Zhang	2023-04-12	2023-04-12 08:00
017	105	George Wang	34	Male	57000	2016-03-15	M	George Wang	2023-04-12	2023-04-12 08:00
018	104	Nancy Liu	29		50000	2017-06-01	O	Nancy Liu	2023-04-12	2023-04-12 08:00
019	103	Steven Chen	36	Male	62000	2015-08-01	M	Steven Chen	2023-04-12	2023-04-12 08:00
020	102	Grace Kim	32	Female	53000	2018-11-01	F	Grace Kim	2023-04-12	2023-04-12 08:00

```
In [36]: # Drop old columns and Fix new column names
emp_final = emp_null_df.drop("name", "gender").withColumnRenamed("new_name", "name").withColumnRenamed("new_gender", "gender")
```

```
In [37]: emp_final.show()
```

employee_id	department_id	age	salary	hire_date	gender	name	date_now	timestamp_now
001	101	30	50000	2015-01-01	M	Zohn Doe	2023-04-12	2023-04-12 08:08:...
002	101	25	45000	2016-02-15	F	Zane Smith	2023-04-12	2023-04-12 08:08:...
003	102	35	55000	2014-05-01	M	Bob Brown	2023-04-12	2023-04-12 08:08:...
004	102	28	48000	2017-09-30	F	Alice Lee	2023-04-12	2023-04-12 08:08:...
005	103	40	60000	2013-04-01	M	Zack Chan	2023-04-12	2023-04-12 08:08:...
006	103	32	52000	2018-07-01	F	Zill Wong	2023-04-12	2023-04-12 08:08:...
007	101	42	70000	2012-03-15	M	Zames Zohnson	2023-04-12	2023-04-12 08:08:...
008	102	29	51000	2019-10-01	F	Kate Kim	2023-04-12	2023-04-12 08:08:...
009	103	33	58000	2016-06-01	M	Tom Tan	2023-04-12	2023-04-12 08:08:...
010	104	27	47000	2018-08-01	F	Lisa Lee	2023-04-12	2023-04-12 08:08:...
011	104	38	65000	2015-11-01	M	David Park	2023-04-12	2023-04-12 08:08:...
012	105	31	54000	2017-02-15	F	Susan Chen	2023-04-12	2023-04-12 08:08:...
013	106	45	75000	2011-07-01	M	Brian Kim	2023-04-12	2023-04-12 08:08:...
014	107	26	46000	2019-01-01	F	Emily Lee	2023-04-12	2023-04-12 08:08:...
015	106	37	63000	2014-09-30	M	Michael Lee	2023-04-12	2023-04-12 08:08:...
016	107	30	49000	2018-04-01	F	Kelly Zhang	2023-04-12	2023-04-12 08:08:...
017	105	34	57000	2016-03-15	M	George Wang	2023-04-12	2023-04-12 08:08:...
018	104	29	50000	2017-06-01	O	Nancy Liu	2023-04-12	2023-04-12 08:08:...
019	103	36	62000	2015-08-01	M	Steven Chen	2023-04-12	2023-04-12 08:08:...
020	102	32	53000	2018-11-01	F	Grace Kim	2023-04-12	2023-04-12 08:08:...

```
In [38]: # Write data as CSV
emp_final.write.format("csv").save("data/output/4/emp.csv")
```

```
In [47]: # Bonus TIP
# Convert date into String and extract date information
from pyspark.sql.functions import date_format

emp_fixed = emp_final.withColumn("date_year", date_format(col("timestamp_now"), "z"))
```

```
In [48]: emp_fixed.show()
```

employee_id	department_id	age	salary	hire_date	gender	name	date_now	timestamp_now	date_year
001	101	30	50000	2015-01-01	M	Zohn Doe	2023-04-12	2023-04-12 08:12:...	UTC
002	101	25	45000	2016-02-15	F	Zane Smith	2023-04-12	2023-04-12 08:12:...	UTC
003	102	35	55000	2014-05-01	M	Bob Brown	2023-04-12	2023-04-12 08:12:...	UTC
004	102	28	48000	2017-09-30	F	Alice Lee	2023-04-12	2023-04-12 08:12:...	UTC
005	103	40	60000	2013-04-01	M	Zack Chan	2023-04-12	2023-04-12 08:12:...	UTC
006	103	32	52000	2018-07-01	F	Zill Wong	2023-04-12	2023-04-12 08:12:...	UTC
007	101	42	70000	2012-03-15	M	James Johnson	2023-04-12	2023-04-12 08:12:...	UTC
008	102	29	51000	2019-10-01	F	Kate Kim	2023-04-12	2023-04-12 08:12:...	UTC
009	103	33	58000	2016-06-01	M	Tom Tan	2023-04-12	2023-04-12 08:12:...	UTC
010	104	27	47000	2018-08-01	F	Lisa Lee	2023-04-12	2023-04-12 08:12:...	UTC
011	104	38	65000	2015-11-01	M	David Park	2023-04-12	2023-04-12 08:12:...	UTC
012	105	31	54000	2017-02-15	F	Susan Chen	2023-04-12	2023-04-12 08:12:...	UTC
013	106	45	75000	2011-07-01	M	Brian Kim	2023-04-12	2023-04-12 08:12:...	UTC
014	107	26	46000	2019-01-01	F	Emily Lee	2023-04-12	2023-04-12 08:12:...	UTC
015	106	37	63000	2014-09-30	M	Michael Lee	2023-04-12	2023-04-12 08:12:...	UTC
016	107	30	49000	2018-04-01	F	Kelly Zhang	2023-04-12	2023-04-12 08:12:...	UTC
017	105	34	57000	2016-03-15	M	George Wang	2023-04-12	2023-04-12 08:12:...	UTC
018	104	29	50000	2017-06-01	O	Nancy Liu	2023-04-12	2023-04-12 08:12:...	UTC
019	103	36	62000	2015-08-01	M	Steven Chen	2023-04-12	2023-04-12 08:12:...	UTC
020	102	32	53000	2018-11-01	F	Grace Kim	2023-04-12	2023-04-12 08:12:...	UTC

Spark Hands On

1. Union of DataFrames
2. Sorting data
3. Aggregations

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Sort Union & Aggregation")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Sort Union & Aggregation

```
In [2]: # Emp Data & Schema

emp_data_1 = [
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"]
]

emp_data_2 = [
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],
    ["018", "104", "Nancy Liu", "29", "", "50000", "2017-06-01"],
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]
]

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date string
```

```
In [3]: # Create emp DataFrame

emp_data_1 = spark.createDataFrame(data=emp_data_1, schema=emp_schema)
emp_data_2 = spark.createDataFrame(data=emp_data_2, schema=emp_schema)
```

```
In [4]: # Show emp dataframe (ACTION)

emp_data_1.show()
emp_data_2.show()

+-----+-----+-----+-----+-----+
|employee_id|department_id|      name|age|gender|salary| hire_date|
+-----+-----+-----+-----+-----+
|     001|       101| John Doe| 30|   Male| 50000|2015-01-01|
|     002|       101| Jane Smith| 25|Female| 45000|2016-02-15|
|     003|       102| Bob Brown| 35|   Male| 55000|2014-05-01|
|     004|       102| Alice Lee| 28|Female| 48000|2017-09-30|
|     005|       103| Jack Chan| 40|   Male| 60000|2013-04-01|
|     006|       103| Jill Wong| 32|Female| 52000|2018-07-01|
|     007|       101|James Johnson| 42|   Male| 70000|2012-03-15|
|     008|       102| Kate Kim| 29|Female| 51000|2019-10-01|
|     009|       103| Tom Tan| 33|   Male| 58000|2016-06-01|
|     010|       104| Lisa Lee| 27|Female| 47000|2018-08-01|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
|employee_id|department_id|      name|age|gender|salary| hire_date|
+-----+-----+-----+-----+-----+
|     011|       104| David Park| 38|   Male| 65000|2015-11-01|
|     012|       105| Susan Chen| 31|Female| 54000|2017-02-15|
|     013|       106| Brian Kim| 45|   Male| 75000|2011-07-01|
|     014|       107| Emily Lee| 26|Female| 46000|2019-01-01|
|     015|       106| Michael Lee| 37|   Male| 63000|2014-09-30|
|     016|       107| Kelly Zhang| 30|Female| 49000|2018-04-01|
|     017|       105| George Wang| 34|   Male| 57000|2016-03-15|
|     018|       104| Nancy Liu| 29|      | 50000|2017-06-01|
|     019|       103| Steven Chen| 36|   Male| 62000|2015-08-01|
|     020|       102| Grace Kim| 32|Female| 53000|2018-11-01|
+-----+-----+-----+-----+-----+
```

```
In [5]: # Print Schema
```

```
emp_data_1.printSchema()  
emp_data_2.printSchema()
```

```
root  
|-- employee_id: string (nullable = true)  
|-- department_id: string (nullable = true)  
|-- name: string (nullable = true)  
|-- age: string (nullable = true)  
|-- gender: string (nullable = true)  
|-- salary: string (nullable = true)  
|-- hire_date: string (nullable = true)
```

```
root  
|-- employee_id: string (nullable = true)  
|-- department_id: string (nullable = true)  
|-- name: string (nullable = true)  
|-- age: string (nullable = true)  
|-- gender: string (nullable = true)  
|-- salary: string (nullable = true)  
|-- hire_date: string (nullable = true)
```

```
In [8]:
```

```
# UNION and UNION ALL  
# select * from emp_data_1 UNION select * from emp_data_2  
emp = emp_data_1.unionAll(emp_data_2)
```

```
In [9]:
```

```
emp.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29		50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [17]: # Sort the emp data based on desc Salary  
# select * from emp order by salary desc  
from pyspark.sql.functions import desc, asc, col  
  
emp_sorted = emp.orderBy(col("salary").asc())
```

```
In [18]: emp_sorted.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
002	101	Jane Smith	25	Female	45000	2016-02-15
014	107	Emily Lee	26	Female	46000	2019-01-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
004	102	Alice Lee	28	Female	48000	2017-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
001	101	John Doe	30	Male	50000	2015-01-01
018	104	Nancy Liu	29		50000	2017-06-01
008	102	Kate Kim	29	Female	51000	2019-10-01
006	103	Jill Wong	32	Female	52000	2018-07-01
020	102	Grace Kim	32	Female	53000	2018-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
017	105	George Wang	34	Male	57000	2016-03-15
009	103	Tom Tan	33	Male	58000	2016-06-01
005	103	Jack Chan	40	Male	60000	2013-04-01
019	103	Steven Chen	36	Male	62000	2015-08-01
015	106	Michael Lee	37	Male	63000	2014-09-30
011	104	David Park	38	Male	65000	2015-11-01
007	101	James Johnson	42	Male	70000	2012-03-15
013	106	Brian Kim	45	Male	75000	2011-07-01

```
In [30]: # Aggregation  
# select dept_id, count(employee_id) as total_dept_count from emp_sorted group by dept_id  
from pyspark.sql.functions import count  
  
emp_count = emp_sorted.groupBy("department_id").agg(count("employee_id").alias("total_dept_count"))
```

```
In [31]: emp_count.show()
```

department_id	total_dept_count
101	3
102	4
103	4
104	3
105	2
106	2
107	2

```
In [34]: # Aggregation  
# select dept_id, sum(salary) as total_dept_salary from emp_sorted group by dept_id  
from pyspark.sql.functions import sum  
  
emp_sum = emp_sorted.groupBy("department_id").agg(sum("salary").alias("total_dept_salary"))
```

```
In [35]: emp_sum.show()
```

```
+-----+-----+
|department_id|total_dept_salary|
+-----+-----+
|      101|     165000.0|
|      107|     95000.0|
|      104|    162000.0|
|      102|    207000.0|
|      103|    232000.0|
|      106|    138000.0|
|      105|    111000.0|
+-----+-----+
```

```
In [36]:
```

```
# Aggregation with having clause
# select dept_id, avg(salary) as avg_dept_salary from emp_sorted group by dept_id having avg(salary) > 50000
from pyspark.sql.functions import avg

emp_avg = emp_sorted.groupBy("department_id").agg(avg("salary").alias("avg_dept_salary")).where("avg_dept_salary > 50000")
```

```
In [37]:
```

```
emp_avg.show()
```

```
+-----+-----+
|department_id|avg_dept_salary|
+-----+-----+
|      101|     55000.0|
|      104|     54000.0|
|      102|     51750.0|
|      103|     58000.0|
|      106|     69000.0|
|      105|     55500.0|
+-----+-----+
```

```
In [42]:
```

```
# Bonus TIP - unionByName
# In case the column sequence is different
emp_data_2_other = emp_data_2.select("employee_id", "salary", "department_id", "name", "hire_date", "gender", "age")

emp_data_1.printSchema()
emp_data_2_other.printSchema()
```

```
root
 |-- employee_id: string (nullable = true)
 |-- department_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: string (nullable = true)
 |-- hire_date: string (nullable = true)

root
 |-- employee_id: string (nullable = true)
 |-- salary: string (nullable = true)
 |-- department_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- hire_date: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: string (nullable = true)
```

```
In [43]:
```

```
emp_fixed = emp_data_1.unionByName(emp_data_2_other)
```

```
In [ ]: emp_fixed.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29		50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [ ]: emp.count()
```

```
Out[ ]: 20
```

Union vs UnionAll :- Union removed the duplicated from the combined dataset whereas UnionAll doesn't.

Using select or selectExpr- We can write the column aggregation expressions in select or selectExpr as well.

Spark Hands On

1. Unique/Distinct data
2. Window Functions
3. Databricks Community Cloud

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Unique data & Window Functions")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Unique data & Window Functions


```
In [2]: # Emp Data & Schema

emp_data = [
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"],
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],
    ["018", "104", "Nancy Liu", "29", "", "50000", "2017-06-01"],
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]
]

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date string
```



```
In [3]: # Create emp DataFrame

emp = spark.createDataFrame(data=emp_data, schema=emp_schema)
```



```
In [4]: # Show emp dataframe (ACTION)

emp.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29		50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [5]: # Print Schema
```

```
emp.printSchema()
```

```
root
|-- employee_id: string (nullable = true)
|-- department_id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: string (nullable = true)
|-- gender: string (nullable = true)
|-- salary: string (nullable = true)
|-- hire_date: string (nullable = true)
```

```
In [6]: # Get unique data
```

```
# select distinct emp.* from emp
emp_unique = emp.distinct()
```

```
In [7]: emp_unique.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
010	104	Lisa Lee	27	Female	47000	2018-08-01
009	103	Tom Tan	33	Male	58000	2016-06-01
008	102	Kate Kim	29	Female	51000	2019-10-01
012	105	Susan Chen	31	Female	54000	2017-02-15
011	104	David Park	38	Male	65000	2015-11-01
014	107	Emily Lee	26	Female	46000	2019-01-01
013	106	Brian Kim	45	Male	75000	2011-07-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
019	103	Steven Chen	36	Male	62000	2015-08-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29		50000	2017-06-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [ ]: # Unique of department_ids
```

```
# select distinct department_id from emp
emp_dept_id = emp.select("department_id").distinct()
```

```
In [ ]: emp_dept_id.show()
```

department_id
101
102
103
104
105
107
106

```
In [14]:
```

```
# Window Functions
# select *, max(salary) over(partition by department_id order by salary desc) as max_salary from emp_unique
from pyspark.sql.window import Window
from pyspark.sql.functions import max, col, desc

window_spec = Window.partitionBy(col("department_id")).orderBy(col("salary").desc())
max_func = max(col("salary")).over(window_spec)

emp_1 = emp.withColumn("max_salary", max_func)
```

```
In [18]: emp_1.show()

# Window Functions - 2nd highest salary of each department
# select *, row_number() over(partition by department_id order by salary desc) as rn from emp_unique where rn = 2
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number, desc, col

window_spec = Window.partitionBy(col("department_id")).orderBy(col("salary").desc())
rn = row_number().over(window_spec)

emp_2 = emp.withColumn("rn", rn).where("rn = 2")
```

employee_id	department_id	name	age	gender	salary	hire_date	max_salary
007	101	James Johnson	42	Male	70000	2012-03-15	70000
001	101	John Doe	30	Male	50000	2015-01-01	70000
002	101	Jane Smith	25	Female	45000	2016-02-15	70000
003	102	Bob Brown	35	Male	55000	2014-05-01	55000
020	102	Grace Kim	32	Female	53000	2018-11-01	55000
008	102	Kate Kim	29	Female	51000	2019-10-01	55000
004	102	Alice Lee	28	Female	48000	2017-09-30	55000
019	103	Steven Chen	36	Male	62000	2015-08-01	62000
005	103	Jack Chan	40	Male	60000	2013-04-01	62000
009	103	Tom Tan	33	Male	58000	2016-06-01	62000
006	103	Jill Wong	32	Female	52000	2018-07-01	62000
011	104	David Park	38	Male	65000	2015-11-01	65000
018	104	Nancy Liu	29		50000	2017-06-01	65000
010	104	Lisa Lee	27	Female	47000	2018-08-01	65000
017	105	George Wang	34	Male	57000	2016-03-15	57000
012	105	Susan Chen	31	Female	54000	2017-02-15	57000
013	106	Brian Kim	45	Male	75000	2011-07-01	75000
015	106	Michael Lee	37	Male	63000	2014-09-30	75000
016	107	Kelly Zhang	30	Female	49000	2018-04-01	49000
014	107	Emily Lee	26	Female	46000	2019-01-01	49000

```
In [19]: emp_2.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	rn
001	101	John Doe	30	Male	50000	2015-01-01	2
020	102	Grace Kim	32	Female	53000	2018-11-01	2
005	103	Jack Chan	40	Male	60000	2013-04-01	2
018	104	Nancy Liu	29		50000	2017-06-01	2
012	105	Susan Chen	31	Female	54000	2017-02-15	2
015	106	Michael Lee	37	Male	63000	2014-09-30	2
014	107	Emily Lee	26	Female	46000	2019-01-01	2

```
In [22]: # Window function using expr
# select *, row_number() over(partition by department_id order by salary desc) as rn from emp_unique where rn = 2
from pyspark.sql.functions import expr

emp_3 = emp.withColumn("rn", expr("row_number() over(partition by department_id order by salary desc)").where("rn = 2"))
```

```
In [22]: # Window function using expr
# select *, row_number() over(partition by department_id order by salary desc) as rn from emp_unique where rn = 2
from pyspark.sql.functions import expr

emp_3 = emp.withColumn("rn", expr("row_number() over(partition by department_id order by salary desc)").where("rn = 2"))
```

```
In [23]: emp_3.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	rn
001	101	John Doe	30	Male	50000	2015-01-01	2
020	102	Grace Kim	32	Female	53000	2018-11-01	2
005	103	Jack Chan	40	Male	60000	2013-04-01	2
018	104	Nancy Liu	29		50000	2017-06-01	2
012	105	Susan Chen	31	Female	54000	2017-02-15	2
015	106	Michael Lee	37	Male	63000	2014-09-30	2
014	107	Emily Lee	26	Female	46000	2019-01-01	2

```
In [ ]: # Bonus TIP
# Databricks community cloud
```

Unique and Distinct data from selected column.

Window or Analytical Functions :- Applies aggregate and ranking functions over a particular window(set of rows).

Expr() function for the rescue:- In case anytime PySpark API looks difficult, we can use SQL in expr functions easily.

Spark Hands On

1. Data Repartitioning using Repartition/Coalesce
2. Joins in PySpark

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Joins and Data Partitions")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Joins and Data Partitions

```
In [2]: # Emp Data & Schema

emp_data = [
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"],
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],
    ["018", "104", "Nancy Liu", "29", "", "50000", "2017-06-01"],
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]
]

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date str"

dept_data = [
    ["101", "Sales", "NYC", "US", "1000000"],
    ["102", "Marketing", "LA", "US", "900000"],
    ["103", "Finance", "London", "UK", "1200000"],
    ["104", "Engineering", "Beijing", "China", "1500000"],
    ["105", "Human Resources", "Tokyo", "Japan", "800000"],
    ["106", "Research and Development", "Perth", "Australia", "1100000"],
    ["107", "Customer Service", "Sydney", "Australia", "950000"]
]

dept_schema = "department_id string, department_name string, city string, country string, budget string"
```

```
In [3]: # Create emp & dept DataFrame

emp = spark.createDataFrame(data=emp_data, schema=emp_schema)
dept = spark.createDataFrame(data=dept_data, schema=dept_schema)
```

```
In [4]: # Show emp dataframe (ACTION)

emp.show()
dept.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29		50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
dept.show()

+-----+-----+-----+-----+
|employee_id|department_id|      name|age|gender|salary| hire_date|
+-----+-----+-----+-----+
|    001|       101| John Doe| 30| Male| 50000|2015-01-01|
|    002|       101| Jane Smith| 25|Female| 45000|2016-02-15|
|    003|       102| Bob Brown| 35| Male| 55000|2014-05-01|
|    004|       102| Alice Lee| 28|Female| 48000|2017-09-30|
|    005|       103| Jack Chan| 40| Male| 60000|2013-04-01|
|    006|       103| Jill Wong| 32|Female| 52000|2018-07-01|
|    007|       101|James Johnson| 42| Male| 70000|2012-03-15|
|    008|       102| Kate Kim| 29|Female| 51000|2019-10-01|
|    009|       103| Tom Tan| 33| Male| 58000|2016-06-01|
|    010|       104| Lisa Lee| 27|Female| 47000|2018-08-01|
|    011|       104| David Park| 38| Male| 65000|2015-11-01|
|    012|       105| Susan Chen| 31|Female| 54000|2017-02-15|
|    013|       106| Brian Kim| 45| Male| 75000|2011-07-01|
|    014|       107| Emily Lee| 26|Female| 46000|2019-01-01|
|    015|       106| Michael Lee| 37| Male| 63000|2014-09-30|
|    016|       107| Kelly Zhang| 30|Female| 49000|2018-04-01|
|    017|       105| George Wang| 34| Male| 57000|2016-03-15|
|    018|       104| Nancy Liu| 29|     | 50000|2017-06-01|
|    019|       103| Steven Chen| 36| Male| 62000|2015-08-01|
|    020|       102| Grace Kim| 32|Female| 53000|2018-11-01|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|department_id| department_name|   city| country| budget|
+-----+-----+-----+-----+
|      101|        Sales| NYC| US|1000000|
|      102| Marketing| LA| US| 900000|
|      103|   Finance| London| UK|1200000|
|      104| Engineering|Beijing| China|1500000|
|      105| Human Resources| Tokyo| Japan| 800000|
|      106|Research and Deve...| Perth|Australia|1100000|
|      107| Customer Service| Sydney|Australia| 950000|
+-----+-----+-----+-----+
```

```
In [5]: # Print Schema
emp.printSchema()
dept.printSchema()
```

```
root
 |-- employee_id: string (nullable = true)
 |-- department_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: string (nullable = true)
 |-- hire_date: string (nullable = true)

root
 |-- department_id: string (nullable = true)
 |-- department_name: string (nullable = true)
 |-- city: string (nullable = true)
 |-- country: string (nullable = true)
 |-- budget: string (nullable = true)
```

```
In [6]: # Get number of partitions for emp
emp.rdd.getNumPartitions()
```

Out[6]: 8

```
In [7]: # Get number of partitions for dept
dept.rdd.getNumPartitions()
```

Out[7]: 8

```
In [22]: # Repartition of data using repartition & coalesce  
emp_partitioned = emp.repartition(4, "department_id")
```

```
In [23]: emp_partitioned.rdd.getNumPartitions()
```

```
Out[23]: 4
```

```
In [26]: # Find the partition info for partitions and repartition  
from pyspark.sql.functions import spark_partition_id  
  
emp_1 = emp.repartition(4, "department_id").withColumn("partition_num", spark_partition_id())
```

```
In [27]: emp_1.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	partition_num
003	102	Bob Brown	35	Male	55000	2014-05-01	0
004	102	Alice Lee	28	Female	48000	2017-09-30	0
008	102	Kate Kim	29	Female	51000	2019-10-01	0
014	107	Emily Lee	26	Female	46000	2019-01-01	0
016	107	Kelly Zhang	30	Female	49000	2018-04-01	0
020	102	Grace Kim	32	Female	53000	2018-11-01	0
012	105	Susan Chen	31	Female	54000	2017-02-15	1
017	105	George Wang	34	Male	57000	2016-03-15	1
010	104	Lisa Lee	27	Female	47000	2018-08-01	2
011	104	David Park	38	Male	65000	2015-11-01	2
013	106	Brian Kim	45	Male	75000	2011-07-01	2
015	106	Michael Lee	37	Male	63000	2014-09-30	2
018	104	Nancy Liu	29		50000	2017-06-01	2
001	101	John Doe	30	Male	50000	2015-01-01	3
002	101	Jane Smith	25	Female	45000	2016-02-15	3
005	103	Jack Chan	40	Male	60000	2013-04-01	3
006	103	Jill Wong	32	Female	52000	2018-07-01	3
007	101	James Johnson	42	Male	70000	2012-03-15	3
009	103	Tom Tan	33	Male	58000	2016-06-01	3
019	103	Steven Chen	36	Male	62000	2015-08-01	3

```
In [33]: # INNER JOIN datasets  
# select e.emp_name, d.department_name, d.department_id, e.salary  
# from emp e inner join dept d on emp.department_id = dept.department_id  
  
df_joined = emp.alias("e").join(dept.alias("d"), how="inner", on=emp.department_id==dept.department_id)
```

```
In [35]: df_joined.select("e.name", "d.department_id", "d.department_name", "e.salary").show()
```

	name	department_id	department_name	salary
John Doe	101	Sales	50000	
Jane Smith	101	Sales	45000	
James Johnson	101	Sales	70000	
Bob Brown	102	Marketing	55000	
Alice Lee	102	Marketing	48000	
Kate Kim	102	Marketing	51000	
Grace Kim	102	Marketing	53000	
Jack Chan	103	Finance	60000	
Jill Wong	103	Finance	52000	
Tom Tan	103	Finance	58000	
Steven Chen	103	Finance	62000	
Lisa Lee	104	Engineering	47000	
David Park	104	Engineering	65000	
Nancy Liu	104	Engineering	50000	
Susan Chen	105	Human Resources	54000	
George Wang	105	Human Resources	57000	
Brian Kim	106	Research and Dev...	75000	
Michael Lee	106	Research and Dev...	63000	
Emily Lee	107	Customer Service	46000	
Kelly Zhang	107	Customer Service	49000	

```
In [36]: # LEFT OUTER JOIN datasets
# select e.emp_name, d.department_name, d.department_id, e.salary
# from emp e left outer join dept d on emp.department_id = dept.department_id

df_joined = emp.alias("e").join(dept.alias("d"), how="left_outer", on=emp.department_id==dept.department_id)
```

```
In [38]: df_joined.select("e.name", "d.department_name", "d.department_id", "e.salary").show()
```

John Doe	Sales	101	50000	
Jane Smith	Sales	101	45000	
Bob Brown	Marketing	102	55000	
Alice Lee	Marketing	102	48000	
Jack Chan	Finance	103	60000	
Jill Wong	Finance	103	52000	
James Johnson	Sales	101	70000	
Lisa Lee	Engineering	104	47000	
Kate Kim	Marketing	102	51000	
Tom Tan	Finance	103	58000	
David Park	Engineering	104	65000	
Susan Chen	Human Resources	105	54000	
Emily Lee	Customer Service	107	46000	
Brian Kim	Research and Dev...	106	75000	
Kelly Zhang	Customer Service	107	49000	
Michael Lee	Research and Dev...	106	63000	
Nancy Liu	Engineering	104	50000	
Grace Kim	Marketing	102	53000	
Steven Chen	Finance	103	62000	
George Wang	Human Resources	105	57000	

```
In [41]: # Write the final dataset
df_joined.select("e.name", "d.department_name", "d.department_id", "e.salary").write.format("csv").save("data/output/7/emp_jo")
```

```
In [52]: # Bonus TIP
# Joins with cascading conditions
# Join with Department_id and only for departments 101 or 102
# Join with not null/null conditions

df_final = emp.join(dept, how="left_outer",
                     on=(emp.department_id==dept.department_id) & ((emp.department_id == "101") | (emp.department_id == "102"))
                     & (emp.salary.isNull()))

```

```
In [53]: df_final.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	department_id	department_name	city	country	budget
001	101	John Doe	30	Male	50000	2015-01-01	null	null null	null	null	null
002	101	Jane Smith	25	Female	45000	2016-02-15	null	null null	null	null	null
003	102	Bob Brown	35	Male	55000	2014-05-01	null	null null	null	null	null
004	102	Alice Lee	28	Female	48000	2017-09-30	null	null null	null	null	null
005	103	Jack Chan	40	Male	60000	2013-04-01	null	null null	null	null	null
006	103	Jill Wong	32	Female	52000	2018-07-01	null	null null	null	null	null
007	101	James Johnson	42	Male	70000	2012-03-15	null	null null	null	null	null
010	104	Lisa Lee	27	Female	47000	2018-08-01	null	null null	null	null	null
008	102	Kate Kim	29	Female	51000	2019-10-01	null	null null	null	null	null
009	103	Tom Tan	33	Male	58000	2016-06-01	null	null null	null	null	null
011	104	David Park	38	Male	65000	2015-11-01	null	null null	null	null	null
012	105	Susan Chen	31	Female	54000	2017-02-15	null	null null	null	null	null
014	107	Emily Lee	26	Female	46000	2019-01-01	null	null null	null	null	null
013	106	Brian Kim	45	Male	75000	2011-07-01	null	null null	null	null	null
016	107	Kelly Zhang	30	Female	49000	2018-04-01	null	null null	null	null	null
015	106	Michael Lee	37	Male	63000	2014-09-30	null	null null	null	null	null
018	104	Nancy Liu	29	Male	50000	2017-06-01	null	null null	null	null	null
020	102	Grace Kim	32	Female	53000	2018-11-01	null	null null	null	null	null
019	103	Steven Chen	36	Male	62000	2015-08-01	null	null null	null	null	null
017	105	George Wang	34	Male	57000	2016-03-15	null	null null	null	null	null

Coalesce vs Repartitions:-

Repartition involves data shuffling, whereas Coalesce doesn't.

Repartition can increase or decrease partition numbers but Coalesce can only decrease not increase.

Repartition allows uniform data distribution, but Coalesce can't guarantee it.

Spark Hands On

1. Reading Data from CSV Files
2. Understand Background Working using Spark UI
3. Handing BAD records
4. Spark Read Modes (PERMISSIVE, DROPMALFORMED, FAILFAST)

```
In [10]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Reading from CSV Files")
    .master("local[*]")
    .getOrCreate()
)

spark
```

```
Out[10]: SparkSession - in-memory
```

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Reading from CSV Files

```
In [13]: # Read a csv file into dataframe
```

```
df = spark.read.format("csv").option("header", True).option("inferSchema", True).load("data/input/emp.csv")
```

```
In [14]: df.printSchema()
```

```
root
|-- employee_id: integer (nullable = true)
|-- department_id: integer (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
|-- gender: string (nullable = true)
|-- salary: integer (nullable = true)
|-- hire_date: timestamp (nullable = true)
```

```
In [15]: df.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
1	101	John Doe	30	Male	50000	2015-01-01 00:00:00
2	101	Jane Smith	25	Female	45000	2016-02-15 00:00:00
3	102	Bob Brown	35	Male	55000	2014-05-01 00:00:00
4	102	Alice Lee	28	Female	48000	2017-09-30 00:00:00
5	103	Jack Chan	40	Male	60000	2013-04-01 00:00:00
6	103	Jill Wong	32	Female	52000	2018-07-01 00:00:00
7	101	James Johnson	42	Male	70000	2012-03-15 00:00:00
8	102	Kate Kim	29	Female	51000	2019-10-01 00:00:00
9	103	Tom Tan	33	Male	58000	2016-06-01 00:00:00
10	104	Lisa Lee	27	Female	47000	2018-08-01 00:00:00
11	104	David Park	38	Male	65000	2015-11-01 00:00:00
12	105	Susan Chen	31	Female	54000	2017-02-15 00:00:00
13	106	Brian Kim	45	Male	75000	2011-07-01 00:00:00
14	107	Emily Lee	26	Female	46000	2019-01-01 00:00:00
15	106	Michael Lee	37	Male	63000	2014-09-30 00:00:00
16	107	Kelly Zhang	30	Female	49000	2018-04-01 00:00:00
17	105	George Wang	34	Male	57000	2016-03-15 00:00:00
18	104	Nancy Liu	29	Female	50000	2017-06-01 00:00:00
19	103	Steven Chen	36	Male	62000	2015-08-01 00:00:00
20	102	Grace Kim	32	Female	53000	2018-11-01 00:00:00

```
In [18]: # Reading with Schema  
_schema = "employee_id int, department_id int, name string, age int, gender string, salary double, hire_date date"  
  
df_schema = spark.read.format("csv").option("header",True).schema(_schema).load("data/input/emp.csv")
```

```
In [19]: df_schema.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
1	101	John Doe	30	Male	50000.0	2015-01-01
2	101	Jane Smith	25	Female	45000.0	2016-02-15
3	102	Bob Brown	35	Male	55000.0	2014-05-01
4	102	Alice Lee	28	Female	48000.0	2017-09-30
5	103	Jack Chan	40	Male	60000.0	2013-04-01
6	103	Jill Wong	32	Female	52000.0	2018-07-01
7	101	James Johnson	42	Male	70000.0	2012-03-15
8	102	Kate Kim	29	Female	51000.0	2019-10-01
9	103	Tom Tan	33	Male	58000.0	2016-06-01
10	104	Lisa Lee	27	Female	47000.0	2018-08-01
11	104	David Park	38	Male	65000.0	2015-11-01
12	105	Susan Chen	31	Female	54000.0	2017-02-15
13	106	Brian Kim	45	Male	75000.0	2011-07-01
14	107	Emily Lee	26	Female	46000.0	2019-01-01
15	106	Michael Lee	37	Male	63000.0	2014-09-30
16	107	Kelly Zhang	30	Female	49000.0	2018-04-01
17	105	George Wang	34	Male	57000.0	2016-03-15
18	104	Nancy Liu	29	Female	50000.0	2017-06-01
19	103	Steven Chen	36	Male	62000.0	2015-08-01
20	102	Grace Kim	32	Female	53000.0	2018-11-01

```
In [36]: # Handle BAD records - PERMISSIVE (Default mode)  
  
_schema = "employee_id int, department_id int, name string, age int, gender string, salary double, hire_date date, bad_record string"  
  
df_p = spark.read.format("csv").schema(_schema).option("columnNameOfCorruptRecord", "bad_record").option("header", True).load("data/input/emp.csv")
```

```
In [37]: df_p.printSchema()
```

```
root  
|-- employee_id: integer (nullable = true)  
|-- department_id: integer (nullable = true)  
|-- name: string (nullable = true)  
|-- age: integer (nullable = true)  
|-- gender: string (nullable = true)  
|-- salary: double (nullable = true)  
|-- hire_date: date (nullable = true)  
|-- bad_record: string (nullable = true)
```

```
In [38]: df_p.show()
```

employee_id	department_id	name	age	gender	salary	hire_date	bad_record
1	101	John Doe	30	Male	50000.0	2015-01-01	null
2	101	Jane Smith	25	Female	45000.0	2016-02-15	null
3	102	Bob Brown	35	Male	55000.0	2014-05-01	null
4	102	Alice Lee	28	Female	48000.0	2017-09-30	null
5	103	Jack Chan	40	Male	60000.0	2013-04-01	null
6	103	Jill Wong	32	Female	52000.0	2018-07-01	null
7	101	James Johnson	42	Male	null	2012-03-15 007,101,James Joh...	
8	102	Kate Kim	29	Female	51000.0	2019-10-01	null
9	103	Tom Tan	33	Male	58000.0	2016-06-01	null
10	104	Lisa Lee	27	Female	47000.0	2018-08-01	null
11	104	David Park	38	Male	65000.0	null 011,104,David Par...	
12	105	Susan Chen	31	Female	54000.0	2017-02-15	null
13	106	Brian Kim	45	Male	75000.0	2011-07-01	null
14	107	Emily Lee	26	Female	46000.0	2019-01-01	null
15	106	Michael Lee	37	Male	63000.0	2014-09-30	null
16	107	Kelly Zhang	30	Female	49000.0	2018-04-01	null
17	105	George Wang	34	Male	57000.0	2016-03-15	null
18	104	Nancy Liu	29	Female	50000.0	2017-06-01	null
19	103	Steven Chen	36	Male	62000.0	2015-08-01	null
20	102	Grace Kim	32	Female	53000.0	2018-11-01	null

```
In [44]: # Handle BAD records - DROPMALFORMED
_schema = "employee_id int, department_id int, name string, age int, gender string, salary double, hire_date date"
df_m = spark.read.format("csv").option("header", True).option("mode", "DROPMALFORMED").schema(_schema).load("data/input/emp_new.csv")
df_m.printSchema()
```

```
In [45]: df_m.printSchema()
```

```
root
|-- employee_id: integer (nullable = true)
|-- department_id: integer (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
|-- gender: string (nullable = true)
|-- salary: double (nullable = true)
|-- hire_date: date (nullable = true)
```

```
In [46]: df_m.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
1	101	John Doe	30	Male	50000.0	2015-01-01
2	101	Jane Smith	25	Female	45000.0	2016-02-15
3	102	Bob Brown	35	Male	55000.0	2014-05-01
4	102	Alice Lee	28	Female	48000.0	2017-09-30
5	103	Jack Chan	40	Male	60000.0	2013-04-01
6	103	Jill Wong	32	Female	52000.0	2018-07-01
8	102	Kate Kim	29	Female	51000.0	2019-10-01
9	103	Tom Tan	33	Male	58000.0	2016-06-01
10	104	Lisa Lee	27	Female	47000.0	2018-08-01
12	105	Susan Chen	31	Female	54000.0	2017-02-15
13	106	Brian Kim	45	Male	75000.0	2011-07-01
14	107	Emily Lee	26	Female	46000.0	2019-01-01
15	106	Michael Lee	37	Male	63000.0	2014-09-30
16	107	Kelly Zhang	30	Female	49000.0	2018-04-01
17	105	George Wang	34	Male	57000.0	2016-03-15
18	104	Nancy Liu	29	Female	50000.0	2017-06-01
19	103	Steven Chen	36	Male	62000.0	2015-08-01
20	102	Grace Kim	32	Female	53000.0	2018-11-01

```
In [47]: # Handle BAD records - FAILFAST
_schema = "employee_id int, department_id int, name string, age int, gender string, salary double, hire_date date"
df_m = spark.read.format("csv").option("header", True).option("mode", "FAILFAST").schema(_schema).load("data/input/emp_new.csv")
```

```
In [48]: df_m.printSchema()
```

```
root
|-- employee_id: integer (nullable = true)
|-- department_id: integer (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
|-- gender: string (nullable = true)
|-- salary: double (nullable = true)
|-- hire_date: date (nullable = true)
```



```
In [49]: df_m.show()
```

```
Py4JJavaError                                     Traceback (most recent call last)
Cell In[49], line 1
----> 1 df_m.show()

File /spark/python/pyspark/sql/dataframe.py:606, in DataFrame.show(self, n, truncate, vertical)
   603     raise TypeError("Parameter 'vertical' must be a bool")
   605 if isinstance(truncate, bool) and truncate:
--> 606     print(self._jdf.showString(n, 20, vertical))
   607 else:
   608     try:

File /spark/python/lib/py4j-0.10.9.5-src.zip/py4j/java_gateway.py:1321, in JavaMember.__call__(self, *args)
 1315 command = proto.CALL_COMMAND_NAME + \
 1316     self.command_header + \
 1317     args_command + \
 1318     proto.END_COMMAND_PART
```



```
In [50]: # BONUS TIP
# Multiple options

_options = {
    "header" : "true",
    "inferSchema" : "true",
    "mode" : "PERMISSIVE"
}

df = (spark.read.format("csv").options(**_options).load("data/input/emp.csv"))
```



```
In [51]: df.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
1	101	John Doe	30	Male	50000	2015-01-01 00:00:00
2	101	Jane Smith	25	Female	45000	2016-02-15 00:00:00
3	102	Bob Brown	35	Male	55000	2014-05-01 00:00:00
4	102	Alice Lee	28	Female	48000	2017-09-30 00:00:00
5	103	Jack Chan	40	Male	60000	2013-04-01 00:00:00
6	103	Jill Wong	32	Female	52000	2018-07-01 00:00:00
7	101	James Johnson	42	Male	70000	2012-03-15 00:00:00
8	102	Kate Kim	29	Female	51000	2019-10-01 00:00:00
9	103	Tom Tan	33	Male	58000	2016-06-01 00:00:00
10	104	Lisa Lee	27	Female	47000	2018-08-01 00:00:00
11	104	David Park	38	Male	65000	2015-11-01 00:00:00
12	105	Susan Chen	31	Female	54000	2017-02-15 00:00:00
13	106	Brian Kim	45	Male	75000	2011-07-01 00:00:00
14	107	Emily Lee	26	Female	46000	2019-01-01 00:00:00
15	106	Michael Lee	37	Male	63000	2014-09-30 00:00:00
16	107	Kelly Zhang	30	Female	49000	2018-04-01 00:00:00
17	105	George Wang	34	Male	57000	2016-03-15 00:00:00
18	104	Nancy Liu	29	Female	50000	2017-06-01 00:00:00
19	103	Steven Chen	36	Male	62000	2015-08-01 00:00:00
20	102	Grace Kim	32	Female	53000	2018-11-01 00:00:00


```
In [9]: spark.stop()
```

How many jobs created when we specify the schema during data read?

- When you specify the schema during data reading in Apache Spark, no additional jobs are created for schema inference. This behavior can significantly improve performance because schema inference requires Spark to scan part of the data, which would otherwise trigger additional jobs.

Read Mode for csv:- Useful to handle Bad Records, Schema is mandatory.

Permissive Mode:- Default Mode for data read. We don't need to specify mode explicitly in code.

Drop Malformed Mode:- Drops the Bad Records.

Fail Fast Mode:- Fails the Job as soon a Bad Record is identified.

Spark Hands On

1. Spark Complex Data Formats – Parquet, ORC & AVRO
2. Understand Row and Columnar Data Storage formats
3. Read Parquet and ORC data files
4. Benefits of Parquet Columnar data format

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Reading Complex Data Formats")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Reading Complex Data Formats

```
In [8]: # Read Parquet Sales data
df_parquet = spark.read.format("parquet").load("data/input/sales_total_parquet/*.parquet")
```

```
In [9]: df_parquet.printSchema()
```

```
root
 |-- transacted_at: timestamp (nullable = true)
 |-- trx_id: integer (nullable = true)
 |-- retailer_id: integer (nullable = true)
 |-- description: string (nullable = true)
 |-- amount: double (nullable = true)
 |-- city_id: integer (nullable = true)
```

```
In [10]: df_parquet.show()
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-12-27 19:00:00	330765426	887300947	Kroger ccd id: ...	33.56	2068475652
2017-11-26 21:00:00	1377679664	1070485878	Amazon.com ccd...	24.43	1640819540
2017-12-12 23:00:00	472018705	2001148981	unkn Columbia	1.24	481821583
2017-05-19 00:00:00	1127671830	847200066	Wal-Mart	2155.48	2074005445
2017-11-17 21:00:00	233137169	847200066	Wal-Mart	4.13	2043825401
2017-12-15 12:00:00	603124844	887300947	Kroger ccd id: ...	31.92	1640819540
2017-11-08 12:00:00	1591888712	143327090	Menard 11-08	42.3	2043825401
2017-12-23 12:00:00	1775468459	887300947	Kroger arc id: 1...	284.8	2055198208
2017-09-01 13:00:00	1020833609	2120842315	Burger King ccd...	20.2	1141716004
2017-11-30 12:00:00	1714628652	1898522855	Target ppd id:...	488.42	569532635
2017-11-26 23:00:00	1533320464	62988535	Bed Bath & Beyond...	6.1	1098014353
2017-11-28 21:00:00	2057742958	562903918	McDonald's pp...	2051.68	1487857621
2017-01-25 19:00:00	123604891	847200066	Wal-Mart ccd i...	23.53	45522086
2017-05-21 21:00:00	1660120794	865681996	Nordstrom arc...	708.81	28424447
2017-12-18 22:00:00	1882911667	997626433	Sears Anka...	160.29	1698762556
2017-11-24 20:00:00	734317675	511877722	Best Buy arc i...	1104.7	2001708947
2017-12-21 18:00:00	513313448	887300947	unkn arc id: 9...	32.7	2023698313
2017-12-13 19:00:00	1380128813	582210968	unkn ppd id: ...	19.88	1359730291
2017-11-15 20:00:00	941860469	887300947	Kroger arc id...	55.69	31398417
2017-12-06 23:00:00	1957163946	400404203	unkn arc id: 97...	9.24	1284771958

only showing top 20 rows

```
In [11]: # Read ORC Sales data
df_orc = spark.read.format("orc").load("data/input/sales_total_orc/*.orc")
```

```
In [12]: df_orc.printSchema()
```

```
root
 |-- transacted_at: timestamp (nullable = true)
 |-- trx_id: integer (nullable = true)
 |-- retailer_id: integer (nullable = true)
 |-- description: string (nullable = true)
 |-- amount: double (nullable = true)
 |-- city_id: integer (nullable = true)
```

```
In [13]: df_orc.show()

+-----+-----+-----+-----+-----+
| transacted_at| trx_id|retailer_id| description| amount| city_id|
+-----+-----+-----+-----+-----+
|2017-12-27 19:00:00| 330765426| 887300947|Kroger ccd id: ...| 33.56|2068475652|
|2017-11-26 21:00:00|1377679664| 1070485878|Amazon.com ccd...| 24.43|1640819540|
|2017-12-12 23:00:00| 472018705| 2001148981| unkn Columbia| 1.24| 481821583|
|2017-05-19 19:00:00|1127671830| 847200066| Wal-Mart|2155.48|2074005445|
|2017-11-17 21:00:00| 233137169| 847200066| Wal-Mart| 4.13|2043825401|
|2017-12-15 12:00:00| 603124844| 887300947|Kroger ccd id: ...| 31.92|1640819540|
|2017-11-08 12:00:00|1591888712| 143327090| Menard 11-08| 42.3|2043825401|
|2017-12-23 12:00:00|1775468459| 887300947|Kroger arc id: 1...| 284.8|2055198208|
|2017-09-01 13:00:00|1020833609| 2120842315|Burger King ccd...| 20.2|1141716004|
|2017-11-30 12:00:00|1714628652| 1898522855|Target ppd id:...| 488.42| 569532635|
|2017-11-26 23:00:00|1533320464| 62988535|Bed Bath & Beyond...| 6.1|1098014353|
|2017-11-28 21:00:00|2057742958| 562903918|McDonald's pp...|2051.68|1487857621|
|2017-01-25 19:00:00| 123604891| 847200066|Wal-Mart ccd i...| 23.53| 45522086|
|2017-05-21 21:00:00|1660120794| 865681996|Nordstrom arc...| 708.81| 28424447|
|2017-12-18 22:00:00|1882911667| 997626433|Sears Anka...| 160.29|1698762556|
|2017-11-24 20:00:00| 734317675| 511877722|Best Buy arc i...| 1104.7|2001708947|
|2017-12-21 18:00:00| 513313448| 887300947|unkn arc id: 9...| 32.7|2023698313|
|2017-12-13 19:00:00|1380128813| 582210968|unkn ppd id: ...| 19.88|1359730291|
|2017-11-15 20:00:00| 941860469| 887300947|Kroger arc id...| 55.69| 31398417|
|2017-12-06 23:00:00|1957163946| 400404203|unkn arc id: 97...| 9.24|1284771958|
+-----+-----+-----+-----+-----+
only showing top 20 rows
```



```
In [15]: # Benefits of Columnar Storage

# Lets create a simple Python decorator - {get_time} to get the execution timings
# If you dont know about Python decorators - check out : https://www.geeksforgeeks.org/decorators-in-python/
import time

def get_time(func):
    def inner_get_time() -> str:
        start_time = time.time()
        func()
        end_time = time.time()
        return f"Execution time: {(end_time - start_time)*1000} ms"
    print(inner_get_time())
```



```
In [16]: @get_time
def x():
    df = spark.read.format("parquet").load("data/input/sales_data.parquet")
    df.count()

Execution time: 672.5492477416992 ms
```



```
In [17]: @get_time
def x():
    df = spark.read.format("parquet").load("data/input/sales_data.parquet")
    df.select("trx_id").count()

Execution time: 348.848819732666 ms
```



```
In [ ]: # BONUS TIP
# RECURSIVE READ

sales_recursive
|__ sales_1\1.parquet
|__ sales_1\sales_2\2.parquet
```



```
In [18]: df_1 = spark.read.format("parquet").load("data/input/sales_recursive/sales_1/1.parquet")
df_1.show()
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-11-24 19:00:00	1734117021	644879053	unkn	ppd id: 7...	8.58 930259917


```
In [19]: df_1 = spark.read.format("parquet").load("data/input/sales_recursive/sales_1/sales_2/2.parquet")
df_1.show()
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-11-24 19:00:00	1734117123	1953761884	unkn	ppd id: 15...	19.55 45522086

```
In [22]: df_1 = spark.read.format("parquet").option("recursiveFileLookup", True).load("data/input/sales_recursive/")
df_1.show()
```

```
+-----+-----+-----+-----+
| transacted_at| trx_id|retailer_id| description|amount| city_id|
+-----+-----+-----+-----+
|2017-11-24 19:00:00|1734117123| 1953761884|unkn    ppd id: 15...| 19.55| 45522086|
|2017-11-24 19:00:00|1734117021| 644879053|unkn    ppd id: 7...| 8.58|930259917|
+-----+-----+-----+-----+
```

Row vs Columnar Data Format

Benefit of Columnar Format:- The columnar format lets the reader read, decompress, and process only the columns that are required for the current query.

Parquet vs ORC vs AVRO

PROP/FILE FORMAT	PARQUET	ORC	AVRO
COMPRESSION	BETTER	BEST	GOOD
READ/WRITE	READ	HEAVY READ	WRITE
ROW/COLUMNAR	COLUMN	COLUMN	ROW
SCHEMA EVOLUTION	GOOD	BETTER	BEST
EXAMPLE USE	DELTA LAKE/SPARK	HIVE	KAFKA

Parquet stores the data along with the data.

Demo for Columnar Data Benefit:- Performance benefit while reading Columnar data.

Spark Hands On

1. Reading JSON files in Spark
2. Parsing data from complex JSON structure
3. Important JSON functions – to_json, from_json, explode etc
4. Understanding JSON schema

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Reading and Parsing JSON Files/Data")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Reading and Parsing JSON Files/Data

```
In [2]: # Read Single Line JSON file
df_single = spark.read.format("json").load("data/input/order_singleline.json")
```

```
In [3]: df_single.printSchema()
```

```
root
 |-- contact: array (nullable = true)
 |   |-- element: long (containsNull = true)
 |-- customer_id: string (nullable = true)
 |-- order_id: string (nullable = true)
 |-- order_line_items: array (nullable = true)
 |   |-- element: struct (containsNull = true)
 |   |   |-- amount: double (nullable = true)
 |   |   |-- item_id: string (nullable = true)
 |   |   |-- qty: long (nullable = true)
```

```
In [4]: df_single.show()
```

```
+-----+-----+-----+
| contact|customer_id|order_id| order_line_items|
+-----+-----+-----+
|[90000010000, 9000...]|      C001|     0101|[{"102.45, I001, 6..."]
+-----+-----+-----+
```

```
In [5]: # Read Multiline JSON file
```

```
df_multi = spark.read.format("json").option("multiline", True).load("data/input/order_multiline.json")
```

```
In [6]: df_multi.printSchema()
```

```
root
|-- contact: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- customer_id: string (nullable = true)
|-- order_id: string (nullable = true)
|-- order_line_items: array (nullable = true)
|   |-- element: struct (containsNull = true)
|   |   |-- amount: double (nullable = true)
|   |   |-- item_id: string (nullable = true)
|   |   |-- qty: long (nullable = true)
```

```
In [7]: df_multi.show()
```

```
+-----+-----+-----+
|       contact|customer_id|order_id|    order_line_items|
+-----+-----+-----+
|[9000010000, 9000...|      C001|    0101|[{:102.45, I001, 6...|
+-----+-----+-----+
```

```
In [8]: df = spark.read.format("text").load("data/input/order_singleline.json")
```

```
In [9]: df.printSchema()
```

```
root
|-- value: string (nullable = true)
```

```
In [11]: df.show(truncate=False)
```

```
+-----+
|value
|
+-----+
|{"order_id":"0101","customer_id":"C001","order_line_items":[{"item_id":"I001","qty":6,"amount":102.45},{"item_id":"I003","qty":2,"amount":2.01}],"contact":[9000010000,9000010001]}
+-----+
```

```
In [17]: # With Schema
```

```
_schema = "customer_id string, order_id string, contact array<long>"  
df_schema = spark.read.format("json").schema(_schema).load("data/input/order_singleline.json")
```

```
In [18]: df_schema.show()
```

```
+-----+-----+
|customer_id|order_id|    contact|
+-----+-----+
|      C001|    0101|[9000010000, 9000...|
+-----+-----+
```

```
In [11]: df.show(truncate=False)
+-----+
| value
| +-----+
| | {"order_id": "O101", "customer_id": "C001", "order_line_items": [{"item_id": "I001", "qty": 6, "amount": 102.45}, {"item_id": "I003", "qty": 2, "amount": 2.01}], "contact": [9000010000, 9000010001]}
| +-----+
+-----+
```

```
In [17]: # With Schema
_schema = "customer_id string, order_id string, contact array<long>"
df_schema = spark.read.format("json").schema(_schema).load("data/input/order_singleline.json")
```

```
In [18]: df_schema.show()
+-----+-----+
| customer_id|order_id| contact
+-----+-----+
| C001| O101|[9000010000, 9000...
+-----+-----+
```

```
In [23]: df_schema_new.show()
+-----+-----+-----+
| contact|customer_id|order_id| order_line_items
+-----+-----+-----+
|[9000010000, 9000...]| C001| O101|[{"amount": 102.45, "item_id": "I001", "qty": 6...}]
+-----+-----+-----+
```

```
In [26]: # Function from_json to read from a column
_schema = "contact array<string>, customer_id string, order_id string, order_line_items array<struct<amount double, item_id string>>"
from pyspark.sql.functions import from_json
df_expanded = df.withColumn("parsed", from_json(df.value, _schema))
```

```
In [27]: df_expanded.printSchema()
root
 |-- value: string (nullable = true)
 |-- parsed: struct (nullable = true)
 |   |-- contact: array (nullable = true)
 |   |   |-- element: string (containsNull = true)
 |   |-- customer_id: string (nullable = true)
 |   |-- order_id: string (nullable = true)
 |   |-- order_line_items: array (nullable = true)
 |   |   |-- element: struct (containsNull = true)
 |   |   |   |-- amount: double (nullable = true)
 |   |   |   |-- item_id: string (nullable = true)
 |   |   |   |-- qty: long (nullable = true)
```

```
In [28]: df_expanded.show()

+-----+-----+
|      value|      parsed|
+-----+-----+
|[{"order_id":"0101...|[9000010000, 900...|
+-----+-----+


In [29]: # Function to_json to parse a JSON string
from pyspark.sql.functions import to_json

df_unparsed = df_expanded.withColumn("unparsed", to_json(df_expanded.parsed))

In [30]: df_unparsed.printSchema()

root
 |-- value: string (nullable = true)
 |-- parsed: struct (nullable = true)
 |   |-- contact: array (nullable = true)
 |   |   |-- element: string (containsNull = true)
 |   |-- customer_id: string (nullable = true)
 |   |-- order_id: string (nullable = true)
 |   |-- order_line_items: array (nullable = true)
 |   |   |-- element: struct (containsNull = true)
 |   |   |   |-- amount: double (nullable = true)
 |   |   |   |-- item_id: string (nullable = true)
 |   |   |   |-- qty: long (nullable = true)
 |-- unparsed: string (nullable = true)

In [32]: df_unparsed.select("unparsed").show(truncate=False)

+-----+
|unparsed
|+-----+
|[{"contact": ["9000010000", "9000010001"], "customer_id": "C001", "order_id": "0101", "order_line_items": [{"amount": 102.45, "item_id": "I001", "qty": 6}, {"amount": 2.01, "item_id": "I003", "qty": 2}]}]
+-----+


In [36]: # Get values from Parsed JSON

df_1 = df_expanded.select("parsed.*")

In [38]: from pyspark.sql.functions import explode

df_2 = df_1.withColumn("expanded_line_items", explode("order_line_items"))

In [39]: df_2.show()

+-----+-----+-----+-----+-----+
|      contact|customer_id|order_id|order_line_items|expanded_line_items|
+-----+-----+-----+-----+-----+
|[9000010000, 9000...|     C001|    0101|[{"102.45", "I001", 6...}|  {102.45, I001, 6}|
|[9000010000, 9000...|     C001|    0101|[{"102.45", "I001", 6...}|  {2.01, I003, 2}|
+-----+-----+-----+-----+-----+
```

```
In [48]: df_3 = df_2.select("contact", "customer_id", "order_id", "expanded_line_items.*")

In [49]: df_3.show()

+-----+-----+-----+-----+
| contact|customer_id|order_id|amount|item_id|qty|
+-----+-----+-----+-----+
|[9000010000, 9000...|    C001| 0101|102.45|  I001| 6|
|[9000010000, 9000...|    C001| 0101| 2.01|  I003| 2|
+-----+-----+-----+-----+


In [50]: # Explode Array fields
df_final = df_3.withColumn("contact_expanded", explode("contact"))

In [51]: df_final.printSchema()

root
 |-- contact: array (nullable = true)
 |  |-- element: string (containsNull = true)
 |-- customer_id: string (nullable = true)
 |-- order_id: string (nullable = true)
 |-- amount: double (nullable = true)
 |-- item_id: string (nullable = true)
 |-- qty: long (nullable = true)
 |-- contact_expanded: string (nullable = true)

In [53]: df_final.drop("contact").show()

+-----+-----+-----+-----+
|customer_id|order_id|amount|item_id|qty|contact_expanded|
+-----+-----+-----+-----+
|      C001| 0101|102.45|  I001| 6| 9000010000|
|      C001| 0101|102.45|  I001| 6| 9000010001|
|      C001| 0101| 2.01|  I003| 2| 9000010000|
|      C001| 0101| 2.01|  I003| 2| 9000010001|
+-----+-----+-----+-----+
```

Python Datatype – List/Array or Struct/Dict are important.

Reading JSON Files – Default is single line JSON. In case of multiline we need to specify an option.

Read JSON data in Single Column – We will use format as TEXT.

Read with Schema – Reading JSON file with Schema.

Write JSON Schema – How to write complex JSON schema in string ddl format?

from_json function – Parse the JSON from String

to_json function – Converts Parsed JSON data to String JSON text.

Flatten JSON data – Expand and explode the JSON data to simple tabular structure.

Dot Notation – It can be used to extract\expand struct/dict data.

Spark Hands On

1. Write Data in Spark
2. Understand how Spark writes data with help of Spark UI
3. Important write modes available
4. How writing data with partition works

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Writing data")
    .master("local[*]")
    .getOrCreate()
)

spark
```

```
Out[1]: SparkSession - in-memory
```

SparkContext

[Spark UI](#)

Version	v3.3.0
Master	local[*]
AppName	Writing data

```
In [2]: # Spark available cores with defaultParallelism in Spark UI  
spark.sparkContext.defaultParallelism
```

```
Out[2]: 8
```

```
In [3]: # Emp Data & Schema  
  
emp_data = [  
    ["001", "101", "John Doe", "30", "Male", "50000", "2015-01-01"],  
    ["002", "101", "Jane Smith", "25", "Female", "45000", "2016-02-15"],  
    ["003", "102", "Bob Brown", "35", "Male", "55000", "2014-05-01"],  
    ["004", "102", "Alice Lee", "28", "Female", "48000", "2017-09-30"],  
    ["005", "103", "Jack Chan", "40", "Male", "60000", "2013-04-01"],  
    ["006", "103", "Jill Wong", "32", "Female", "52000", "2018-07-01"],  
    ["007", "101", "James Johnson", "42", "Male", "70000", "2012-03-15"],  
    ["008", "102", "Kate Kim", "29", "Female", "51000", "2019-10-01"],  
    ["009", "103", "Tom Tan", "33", "Male", "58000", "2016-06-01"],  
    ["010", "104", "Lisa Lee", "27", "Female", "47000", "2018-08-01"],  
    ["011", "104", "David Park", "38", "Male", "65000", "2015-11-01"],  
    ["012", "105", "Susan Chen", "31", "Female", "54000", "2017-02-15"],  
    ["013", "106", "Brian Kim", "45", "Male", "75000", "2011-07-01"],  
    ["014", "107", "Emily Lee", "26", "Female", "46000", "2019-01-01"],  
    ["015", "106", "Michael Lee", "37", "Male", "63000", "2014-09-30"],  
    ["016", "107", "Kelly Zhang", "30", "Female", "49000", "2018-04-01"],  
    ["017", "105", "George Wang", "34", "Male", "57000", "2016-03-15"],  
    ["018", "104", "Nancy Liu", "29", "Female", "50000", "2017-06-01"],  
    ["019", "103", "Steven Chen", "36", "Male", "62000", "2015-08-01"],  
    ["020", "102", "Grace Kim", "32", "Female", "53000", "2018-11-01"]  
]  
  
emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date str"
```

```
In [4]: # Create emp DataFrame  
emp = spark.createDataFrame(data=emp_data, schema=emp_schema)
```

```
In [8]: # Get number of partitions and show data  
  
emp.rdd.getNumPartitions()  
emp.show()
```

employee_id	department_id	name	age	gender	salary	hire_date
001	101	John Doe	30	Male	50000	2015-01-01
002	101	Jane Smith	25	Female	45000	2016-02-15
003	102	Bob Brown	35	Male	55000	2014-05-01
004	102	Alice Lee	28	Female	48000	2017-09-30
005	103	Jack Chan	40	Male	60000	2013-04-01
006	103	Jill Wong	32	Female	52000	2018-07-01
007	101	James Johnson	42	Male	70000	2012-03-15
008	102	Kate Kim	29	Female	51000	2019-10-01
009	103	Tom Tan	33	Male	58000	2016-06-01
010	104	Lisa Lee	27	Female	47000	2018-08-01
011	104	David Park	38	Male	65000	2015-11-01
012	105	Susan Chen	31	Female	54000	2017-02-15
013	106	Brian Kim	45	Male	75000	2011-07-01
014	107	Emily Lee	26	Female	46000	2019-01-01
015	106	Michael Lee	37	Male	63000	2014-09-30
016	107	Kelly Zhang	30	Female	49000	2018-04-01
017	105	George Wang	34	Male	57000	2016-03-15
018	104	Nancy Liu	29	Female	50000	2017-06-01
019	103	Steven Chen	36	Male	62000	2015-08-01
020	102	Grace Kim	32	Female	53000	2018-11-01

```
In [9]: # Write the data in parquet format
emp.write.format("parquet").save("data/output/11/2/emp.parquet")
```

```
In [10]: # View data partition information
from pyspark.sql.functions import spark_partition_id

emp.withColumn("partition_id", spark_partition_id()).show()
```

employee_id	department_id	name	age	gender	salary	hire_date	partition_id
001	101	John Doe	30	Male	50000	2015-01-01	0
002	101	Jane Smith	25	Female	45000	2016-02-15	0
003	102	Bob Brown	35	Male	55000	2014-05-01	1
004	102	Alice Lee	28	Female	48000	2017-09-30	1
005	103	Jack Chan	40	Male	60000	2013-04-01	2
006	103	Jill Wong	32	Female	52000	2018-07-01	2
007	101	James Johnson	42	Male	70000	2012-03-15	3
008	102	Kate Kim	29	Female	51000	2019-10-01	3
009	103	Tom Tan	33	Male	58000	2016-06-01	3
010	104	Lisa Lee	27	Female	47000	2018-08-01	3
011	104	David Park	38	Male	65000	2015-11-01	4
012	105	Susan Chen	31	Female	54000	2017-02-15	4
013	106	Brian Kim	45	Male	75000	2011-07-01	5
014	107	Emily Lee	26	Female	46000	2019-01-01	5
015	106	Michael Lee	37	Male	63000	2014-09-30	6
016	107	Kelly Zhang	30	Female	49000	2018-04-01	6
017	105	George Wang	34	Male	57000	2016-03-15	7
018	104	Nancy Liu	29	Female	50000	2017-06-01	7
019	103	Steven Chen	36	Male	62000	2015-08-01	7
020	102	Grace Kim	32	Female	53000	2018-11-01	7

```
In [11]: emp.write.format("csv").option("header", True).save("data/output/11/3/emp.csv")
```

```
In [12]: # Write the data with Partition to output Location
emp.write.format("csv").partitionBy("department_id").option("header", True).save("data/output/11/4/emp.csv")
```

```
In [16]: # Write Modes - append, overwrite, ignore and error
emp.write.format("csv").mode("error").option("header", True).save("data/output/11/3/emp.csv")
```

```
AnalysisException                                     Traceback (most recent call last)
/tmp/ipykernel_1108/3271497296.py in <module>
      1 # Write Modes - append, overwrite, ignore and error
      2
----> 3 emp.write.format("csv").mode("error").option("header", True).save("data/output/11/3/emp.csv")

/spark/python/pyspark/sql/readwriter.py in save(self, path, format, mode, partitionBy, **options)
    966         self._jwrite.save()
    967     else:
--> 968         self._jwrite.save(path)
    969
    970     @since(1.4)

/spark/python/lib/py4j-0.10.9.5-src.zip/py4j/java_gateway.py in __call__(self, *args)
   1320         answer = self.gateway_client.send_command(command)
   1321         return_value = get_return_value(
--> 1322             answer, self.gateway_client, self.target_id, self.name)
   1323
   1324     for temp_arg in temp_args:

/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
   194         # Hide where the exception came from that shows a non-Pythonic
   195         # JVM exception message.
--> 196         raise converted from None
   197     else:
   198         raise

AnalysisException: path file:/home/jupyter/pyspark-zero-to-hero/data/output/11/3/emp.csv already exists.
```

```
In [17]: # Bonus TIP
# What if we need to write only 1 output file to share with DownStream?

emp.repartition(1).write.format("csv").option("header", True).save("data/output/11/5/emp.csv")
```

How Spark writes data?

- **Cluster with 1 node, 2 Executors and each Executor with 2 cores.**
- **Total cores/Parallelism : 4**

Number of Files – The processed data from each partition will be written to individual files.

Default Parallelism – Total number of Cores available for processing data in parallel.

Spark creates the folder.

Spark Partition folders – It allows Spark to skip un-necessary data read in case you need data based on specific partition.

Note – Spark doesn't write department_id column in files as it is the partition column (info from folder name). You can see it is missing in the files.

Spark Hands On

1. How Spark executes code on Clusters
2. What are different Resource/Cluster Managers
3. Deployment modes for Spark
4. Spark Properties/Config for Cluster Management

```
In [7]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Cluster Execution")
    .master("spark://17e348267994:7077")
    .config("spark.executor.instances", 4)
    .config("spark.executor.cores", 4)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[7]: **SparkSession - in-memory**

SparkContext

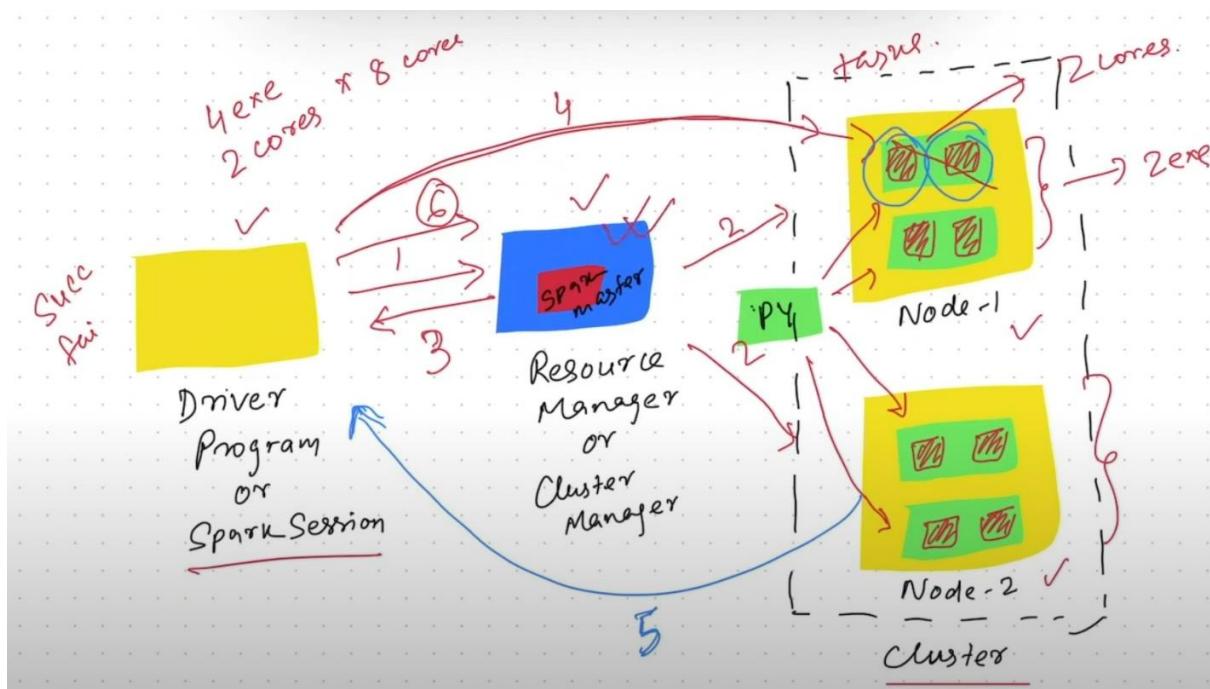
Spark UI

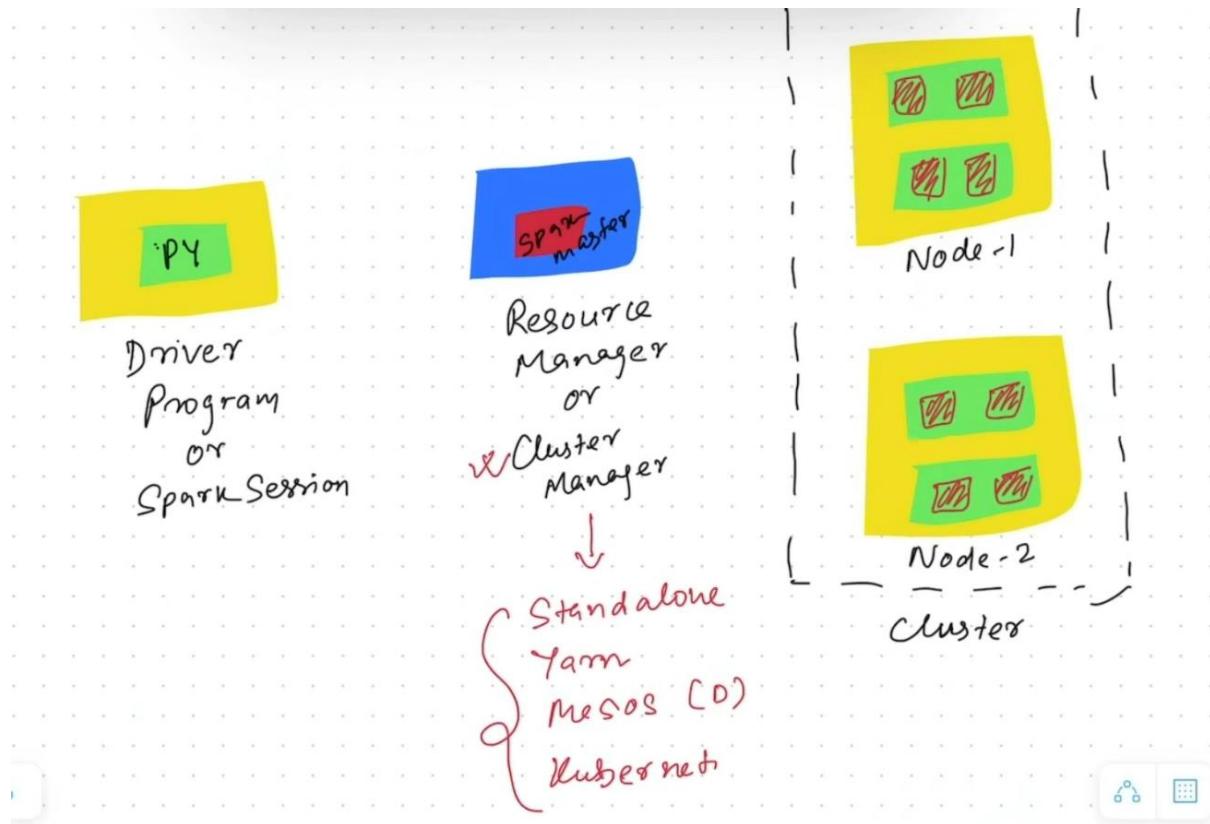
Version	v3.3.0
Master	spark://17e348267994:7077
AppName	Cluster Execution

```
In [8]: # Create a sample data frame
df = spark.range(10)
```

```
In [9]: # Write the data of the data frame
df.write.format("csv").option("header", True).save("/data/output/15/3/range.csv")
```

```
In [10]: # Stop Spark Session
spark.stop()
```





Spark Submit – Command is used to execute Spark application on Cluster.

Note – Spark Submit command can be supplied from the installed Spark folder, check your setting before executing the code.

Note: For Standalone clusters:

the –num-executors parameter may not work always.

So, to control the number of executors:

1. Define number of cores per executors with –executor-cores parameter(spark.executor.cores)
2. Control max number of cores for execution with –total-executor-cores parameter(spark.cores.max)

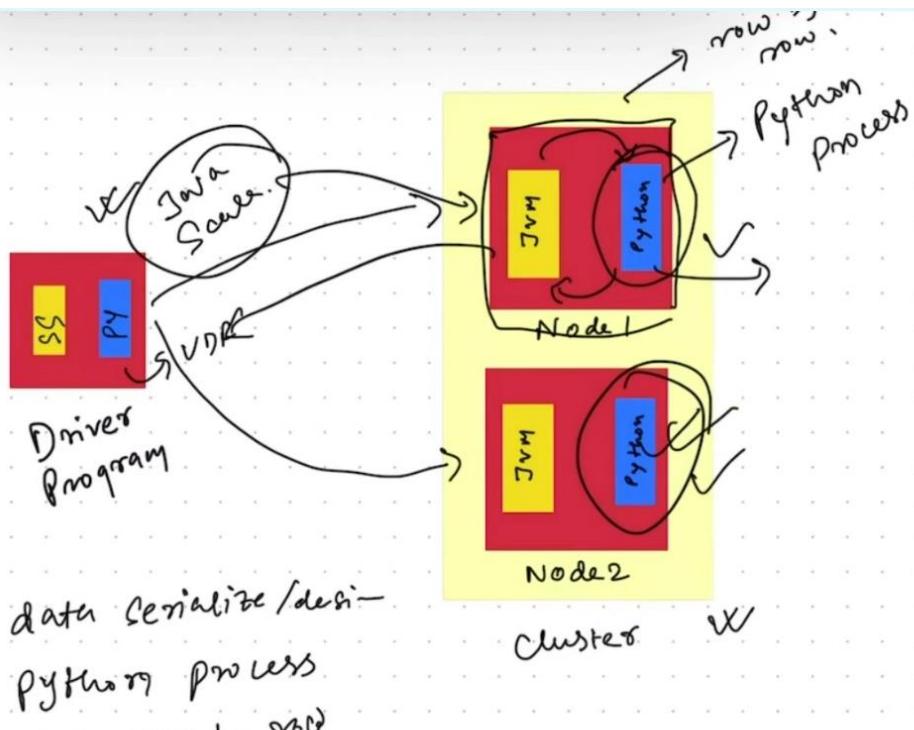
If you need 3 executors with 2 cores(you don't need to use –num-executors) --executor-cores 2 –total-executor-cores 6

--num-executors parameter can be used to control number of executor for YARN resource manager.

Note: Spark Submit commands are often scheduled or triggered to execute jobs on Clusters in Production setting.

Spark Hands On

1. UDF – User Defined Functions for Spark
2. How Spark works with UDF
3. Demerits of Python UDF



What are JVM here?

-JVM are the executors requested by the Spark Application.

```
In [13]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("User Defined Functions")
    .master("spark://17e348267994:7077")
    .config("spark.executor.cores", 2)
    .config("spark.cores.max", 6)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[13]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	spark://17e348267994:7077
AppName	User Defined Functions

```
In [15]: # Read employee data

emp_schema = "employee_id string, department_id string, name string, age string, gender string, salary string, hire_date string"
emp = spark.read.format("csv").option("header", True).schema(emp_schema).load("/data/output/3/emp.csv")

emp.rdd.getNumPartitions()
```

Out[15]: 2

```
In [19]: # Create a function to generate 10% of Salary as Bonus

def bonus(salary):
    return int(salary) * 0.1
```

```
In [20]: # Register as UDF
from pyspark.sql.functions import udf

bonus_udf = udf(bonus)

spark.udf.register("bonus_sql_udf", bonus, "double")
```

Out[20]: <function __main__.bonus>

```
In [24]: # Create new column as bonus using UDF
from pyspark.sql.functions import expr

emp.withColumn("bonus", expr("bonus_sql_udf(salary)").show()
```

employee_id	department_id	name	age	gender	salary	hire_date	bonus
017	105	George Wang	34	Male	57000	2016-03-15	5700.0
018	104	Nancy Liu	29	Female	50000	2017-06-01	5000.0
019	103	Steven Chen	36	Male	62000	2015-08-01	6200.0
020	102	Grace Kim	32	Female	53000	2018-11-01	5300.0
007	101	James Johnson	42	Male	70000	2012-03-15	7000.0
008	102	Kate Kim	29	Female	51000	2019-10-01	5100.0
009	103	Tom Tan	33	Male	58000	2016-06-01	5800.0
010	104	Lisa Lee	27	Female	47000	2018-08-01	4700.0
015	106	Michael Lee	37	Male	63000	2014-09-30	6300.0
016	107	Kelly Zhang	30	Female	49000	2018-04-01	4900.0
011	104	David Park	38	Male	65000	2015-11-01	6500.0
012	105	Susan Chen	31	Female	54000	2017-02-15	5400.0
013	106	Brian Kim	45	Male	75000	2011-07-01	7500.0
014	107	Emily Lee	26	Female	46000	2019-01-01	4600.0
001	101	John Doe	30	Male	50000	2015-01-01	5000.0
002	101	Jane Smith	25	Female	45000	2016-02-15	4500.0
003	102	Bob Brown	35	Male	55000	2014-05-01	5500.0
004	102	Alice Lee	28	Female	48000	2017-09-30	4800.0
005	103	Jack Chan	40	Male	60000	2013-04-01	6000.0
006	103	Jill Wong	32	Female	52000	2018-07-01	5200.0

```
In [25]: # Create new column as bonus without UDF
emp.withColumn("bonus", expr("salary * 0.1")).show()
```

employee_id	department_id	name	age	gender	salary	hire_date	bonus
017	105	George Wang	34	Male	57000	2016-03-15	5700.0
018	104	Nancy Liu	29	Female	50000	2017-06-01	5000.0
019	103	Steven Chen	36	Male	62000	2015-08-01	6200.0
020	102	Grace Kim	32	Female	53000	2018-11-01	5300.0
007	101	James Johnson	42	Male	70000	2012-03-15	7000.0
008	102	Kate Kim	29	Female	51000	2019-10-01	5100.0
009	103	Tom Tan	33	Male	58000	2016-06-01	5800.0
010	104	Lisa Lee	27	Female	47000	2018-08-01	4700.0
015	106	Michael Lee	37	Male	63000	2014-09-30	6300.0
016	107	Kelly Zhang	30	Female	49000	2018-04-01	4900.0
011	104	David Park	38	Male	65000	2015-11-01	6500.0
012	105	Susan Chen	31	Female	54000	2017-02-15	5400.0
013	106	Brian Kim	45	Male	75000	2011-07-01	7500.0
014	107	Emily Lee	26	Female	46000	2019-01-01	4600.0
001	101	John Doe	30	Male	50000	2015-01-01	5000.0
002	101	Jane Smith	25	Female	45000	2016-02-15	4500.0
003	102	Bob Brown	35	Male	55000	2014-05-01	5500.0
004	102	Alice Lee	28	Female	48000	2017-09-30	4800.0
005	103	Jack Chan	40	Male	60000	2013-04-01	6000.0
006	103	Jill Wong	32	Female	52000	2018-07-01	5200.0

```
In [26]: # Stop Spark Session
spark.stop()
```

Spark Hands On

1. Understand Spark Explain Plan
2. Understand DAG
3. Understand Shuffle and Impact on Stages/Tasks?
4. What are Data Frames made of ?

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Understand Plans and DAG")
    .master("local[*]")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Understand Plans and DAG

```
In [2]: # Disable AQE and Broadcast join

spark.conf.set("spark.sql.adaptive.enabled", False)
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", False)
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
```

```
In [3]: # Check default Parallelism

spark.sparkContext.defaultParallelism
```

Out[3]: 8

```
In [4]: # Create dataframes

df_1 = spark.range(4, 200, 2)
df_2 = spark.range(2, 200, 4)
```

```
In [6]: df_2.rdd.getNumPartitions()
```

Out[6]: 8

```
In [7]: # Re-partition data

df_3 = df_1.repartition(5)
df_4 = df_2.repartition(7)
```

```
In [9]: df_4.rdd.getNumPartitions()
```

Out[9]: 7

```
In [10]: # Join the dataframes

df_joined = df_3.join(df_4, on="id")
```

```
In [11]: # Get the sum of ids

df_sum = df_joined.selectExpr("sum(id) as total_sum")
```

```
In [12]: # View data
df_sum.show()

+-----+
|total_sum|
+-----+
|      4998|
+-----+
```

```
In [13]: # Explain plan  
df_sum.explain()
```

```
== Physical Plan ==  
*(6) HashAggregate(keys=[], functions=[sum(id#0L)])  
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#182]  
  +- *(5) HashAggregate(keys=[], functions=[partial_sum(id#0L)])  
    +- *(5) Project [id#0L]  
      +- *(5) SortMergeJoin [id#0L], [id#2L], Inner  
        :- *(2) Sort [id#0L ASC NULLS FIRST], false, 0  
          :  +- Exchange hashpartitioning(id#0L, 200), ENSURE_REQUIREMENTS, [id=#166]  
          :    +- Exchange RoundRobinPartitioning(5), REPARTITION_BY_NUM, [id=#165]  
          :      +- *(1) Range (4, 200, step=2, splits=8)  
        +- *(4) Sort [id#2L ASC NULLS FIRST], false, 0  
          +- Exchange hashpartitioning(id#2L, 200), ENSURE_REQUIREMENTS, [id=#173]  
            +- Exchange RoundRobinPartitioning(7), REPARTITION_BY_NUM, [id=#172]  
              +- *(3) Range (2, 200, step=4, splits=8)
```

```
In [14]: # Union the data again to see the skipped stages  
df_union = df_sum.union(df_4)
```

```
In [15]: df_union.show()
```

total_sum
4998
14
38
50
74
110
130
154
186
10
30
54
98
118
138
158
178
6
42
70

only showing top 20 rows

```
In [16]: # Explain plan  
df_union.explain()
```

```
== Physical Plan ==  
Union  
:- *(6) HashAggregate(keys=[], functions=[sum(id#0L)])  
  :- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#420]  
  :  +- *(5) HashAggregate(keys=[], functions=[partial_sum(id#0L)])  
  :    +- *(5) Project [id#0L]  
  :      +- *(5) SortMergeJoin [id#0L], [id#2L], Inner  
  :        :- *(2) Sort [id#0L ASC NULLS FIRST], false, 0  
  :          :  +- Exchange hashpartitioning(id#0L, 200), ENSURE_REQUIREMENTS, [id=#404]  
  :            +- Exchange RoundRobinPartitioning(5), REPARTITION_BY_NUM, [id=#403]  
  :              +- *(1) Range (4, 200, step=2, splits=8)  
  :      +- *(4) Sort [id#2L ASC NULLS FIRST], false, 0  
  :        +- Exchange hashpartitioning(id#2L, 200), ENSURE_REQUIREMENTS, [id=#411]  
  :          +- Exchange RoundRobinPartitioning(7), REPARTITION_BY_NUM, [id=#410]  
  :            +- *(3) Range (2, 200, step=4, splits=8)  
+- ReusedExchange [id#20L], Exchange RoundRobinPartitioning(7), REPARTITION_BY_NUM, [id=#410]
```

```
In [17]: # DataFrame to RDD  
df_1.rdd
```

```
Out[17]: MapPartitionsRDD[5] at javaToPython at NativeMethodAccessorImpl.java:0
```

Shuffle or Exchange – Shuffle divides the JOB into Stages.

Default Shuffle Partition – By default the shuffle partition value is 200. This can be controlled from Spark config.

Spark Hands On

1. Spark Pipelining
2. Understand Shuffle Operation and Shuffle data
3. Optimize Shuffle in Spark

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Optimizing Shuffles")
    .master("spark://17e348267994:7077")
    .config("spark.cores.max", 16)
    .config("spark.executor.cores", 4)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext	
Spark UI	
Version	v3.3.0
Master	spark://17e348267994:7077
AppName	Optimizing Shuffles

```
In [2]: # Check Spark defaultParallelism
spark.sparkContext.defaultParallelism
```

```
Out[2]: 16
```

```
In [3]: # Disable AQE and Broadcast join
spark.conf.set("spark.sql.adaptive.enabled", False)
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", False)
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
```

```
In [4]: # Read EMP CSV file with 10M records
_schema = "first_name string, last_name string, job_title string, dob string, email string, phone string, salary double, dept"
emp = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/datasets/employee_records.csv")
```

```
In [6]: # Find out avg salary as per dept
from pyspark.sql.functions import avg
emp_avg = emp.groupBy("department_id").agg(avg("salary").alias("avg_sal"))
```

```
In [15]: # Write data for performance Benchmarking
emp_avg.write.format("noop").mode("overwrite").save()
```

```
In [14]: # Check Spark Shuffle Partition setting
spark.conf.get("spark.sql.shuffle.partitions")
```

```
Out[14]: '16'
```

```
In [13]: spark.conf.set("spark.sql.shuffle.partitions", 16)
```

```
In [9]: from pyspark.sql.functions import spark_partition_id
emp.withColumn("partition_id", spark_partition_id()).where("partition_id = 0").show()
```

first_name	last_name	job_title	dob	email	phone	salary	department_id	partition_id
Richard	Morrison	Public relations ...	1973-05-05	melissagarcia@example...	(699)525-4827	512653.0	8	0
Bobby	McCarthy	Barrister's clerk	1974-04-25	llara@example.net	(750)846-1602	x7458	999836.0	7
Dennis	Norman	Land/geomatics su...	1990-06-24	jturner@example.net	873.820.0518	x825	131900.0	10
John	Monroe	Retail buyer	1968-06-16	erik33@example.net	820-813-0557	x624	485506.0	1
Michelle	Elliott	Air cabin crew	1975-03-31	tiffanyjohnston@e...	(705)900-5337	604738.0	8	0
Ashley	Montoya	Cartographer	1976-01-16	patrickalexandra@...	211.440.5466	483339.0	6	0
Nathaniel	Smith	Quality manager	1985-06-28	lori44@example.net	936-403-3179	419644.0	7	0
Faith	Cummings	Industrial/produc...	1978-07-01	ygordon@example.org	(889)246-5588	205939.0	7	0

```

0| Margaret| Sutton|Administrator, ed...|1975-08-16| diana44@example.net|001-647-530-5036x...|671167.0| 8|
0| Mary| Sutton| Freight forwarder|1979-12-28| ryan36@example.com| 422.562.7254x3159|993829.0| 7|
0| Jake| King| Lexicographer|1994-07-11|monica93@example.org|+1-535-652-9715x6...|702101.0| 4|
0| Heather| Haley| Music tutor|1981-06-01|stephanie65@example...| (652)815-7973x298|570960.0| 6|
0| Thomas| Thomas|Chartered managem...|2001-07-17|pwilliams@example...|001-245-848-0028x...|339441.0| 6|
0| Leonard| Carlson| Art therapist|1990-10-18|gabrielmurray@example...| 9247590563|469728.0| 8|
0| Mark| Wood| Market researcher|1963-10-13|nicholas76@example...| 311.439.1606x3342|582291.0| 4|
0| Tracey|Washington|Travel agency man...|1986-05-07| mark07@example.com| 001-912-206-6456|146456.0| 4|
0| Rachael| Rodriguez| Media buyer|1966-12-02|griffinmary@example...| +1-791-344-7586x548|544732.0| 1|
0| Tara| Liu| Financial adviser|1998-10-12|alexandraobrien@example...| 216.696.6061|399503.0| 3|
0| Ana| Joseph| Retail manager|1995-01-10| rmorse@example.org| (726)363-7526x9965|761988.0| 10|
0| Richard| Hall|Engineer, civil (...|1967-03-02|brandoncardenas@example...| (964)451-9007x22496|660659.0| 4|
0|
+-----+
-----+
only showing top 20 rows

```

```
In [16]: # Read the partitioned data
emp_part = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/emp_partitioned.csv")
```

```
In [17]: emp_avg = emp_part.groupBy("department_id").agg(avg("salary").alias("avg_sal"))
```

```
In [18]: emp_avg.write.format("noop").mode("overwrite").save()
```

Shuffle Files:

Are serialized in Tungsten Binary Format(Unsafe Row). These files can directly be read in memory thus improving read performance.

Adaptive Query Engine (AQE):

By default provides some performance benefits on Shuffle.

Spark Hands On

1. Spark Data Persistence (Cache and Persist)
2. Understand How Caching works in Spark
3. Different levels of Caching

```
In [2]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Understand Caching")
    .master("local[*]")
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[2]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.3.0
Master	local[*]
AppName	Understand Caching

```
In [3]: # Read Sales CSV Data - 752MB Size ~ 7.2M Records

_schema = "transacted_at string, trx_id string, retailer_id string, description string, amount double, city_id string"
df = spark.read.format("csv").schema(_schema).option("header", True).load("data/input/new_sales.csv")
```

In [4]: df.where("amount > 300").show()

```
+-----+-----+-----+-----+-----+
| transacted_at | trx_id | retailer_id | description | amount | city_id |
+-----+-----+-----+-----+-----+
| 2017-11-24T19:00:... | 1734117022 | 847200066 | Wal-Mart ppd id:... | 1737.26 | 1646415505 |
| 2017-11-24T19:00:... | 1734117030 | 1953761884 | Home Depot pp... | 384.5 | 287177635 |
| 2017-11-24T19:00:... | 1734117153 | 847200066 | unkn Kings... | 2907.57 | 1483931123 |
| 2017-11-24T19:00:... | 1734117241 | 486576507 | iTunes | 2912.67 | 1663872965 |
| 2017-11-24T19:00:... | 2076947146 | 511877722 | unkn ccd id: ... | 1915.35 | 1698762556 |
| 2017-11-24T19:00:... | 2076947113 | 1996661856 | AutoZone arc id:... | 1523.6 | 1759612211 |
| 2017-11-24T19:00:... | 2076946994 | 1898522855 | Target ppd id:... | 2589.93 | 2074005445 |
| 2017-11-24T19:00:... | 2076946121 | 562903918 | unkn ccd id: 5... | 315.86 | 1773943669 |
| 2017-11-24T19:00:... | 2076946063 | 1070485878 | Amazon.com arc ... | 785.27 | 1126623009 |
| 2017-11-24T19:00:... | 2076944979 | 1654681099 | Delhaize America ... | 303.1 | 1243655802 |
| 2017-11-24T19:00:... | 2076944941 | 1157343460 | unkn ppd id: 1... | 2853.33 | 1141716004 |
| 2017-11-24T19:00:... | 2076944228 | 1522061472 | YUM! Brands | 1737.45 | 592064091 |
| 2017-11-24T19:00:... | 2076944195 | 1070485878 | unkn ppd id: 11... | 2440.55 | 1525790470 |
| 2017-11-24T19:00:... | 2076944142 | 847200066 | Wal-Mart ppd id:... | 331.63 | 1345953582 |
| 2017-11-24T19:00:... | 2076944073 | 2077350195 | Walgreen arc ... | 396.9 | 2001708947 |
| 2017-11-24T19:00:... | 2076943052 | 103953879 | Rite Aid arc id:... | 1910.8 | 1998549640 |
| 2017-11-24T19:00:... | 2076942340 | 643354906 | BJ's | 372.7 | 115209716 |
| 2017-11-24T19:00:... | 2076942282 | 1445595477 | Meijer ccd id:... | 366.9 | 1717498102 |
| 2017-11-24T19:00:... | 2076942274 | 2001148981 | unkn ppd id: 2... | 333.41 | 559832710 |
| 2017-11-24T19:00:... | 2076942246 | 9225731 | AT&T Wireless pp... | 396.9 | 407629665 |
+-----+-----+-----+-----+-----+
only showing top 20 rows
```

```
In [12]: # Cache DataFrame (cache or persist)

df_cache = df.where("amount > 100").cache()
```

In [13]: df_cache.count()

Out[13]: 2549058

```
In [14]: df.where("amount > 50").show()

+-----+-----+-----+-----+-----+
| transacted_at| trx_id|retailer_id| description| amount| city_id|
+-----+-----+-----+-----+-----+
|2017-11-24T19:00:...|1995601912| 2077350195|Walgreen      11-25| 197.23| 216510442|
|2017-11-24T19:00:...|1734117022| 847200066|Wal-Mart ppd id:...|1737.26|1646415505|
|2017-11-24T19:00:...|1734117030| 1953761884|Home Depot     pp...| 384.5| 287177635|
|2017-11-24T19:00:...|1734117089| 1898522855| Target        11-25| 66.33|1855530529|
|2017-11-24T19:00:...|1734117117| 997626433|Sears ppd id: 85...| 298.87| 957346984|
|2017-11-24T19:00:...|1734117153| 847200066|unkn       Kings...|2907.57|1483931123|
|2017-11-24T19:00:...|1734117212| 1996661856|unkn ppd id: 4...| 140.38| 336763936|
|2017-11-24T19:00:...|1734117241| 486576507| iTunes        |2912.67|1663872965|
|2017-11-24T19:00:...|2076947148| 847200066|Wal-Mart     ...| 62.83|1556600840|
|2017-11-24T19:00:...|2076947146| 511877722|unkn ccd id: ...|1915.35|1698762556|
|2017-11-24T19:00:...|2076947113| 1996661856|AutoZone     arc id:...| 1523.6|1759612211|
|2017-11-24T19:00:...|2076946994| 1898522855|Target ppd id:...|2589.93|2074005445|
|2017-11-24T19:00:...|2076946899| 644879053|unkn ccd id: ...| 91.91|2068475652|
|2017-11-24T19:00:...|2076946121| 562903918|unkn ccd id: 5...| 315.86|1773943669|
|2017-11-24T19:00:...|2076946063| 1070485878|Amazon.com    arc ...| 785.27|1126623009|
|2017-11-24T19:00:...|2076945932| 87529419|Albertsons arc i...| 284.16|1813967666|
|2017-11-24T19:00:...|2076945195| 562903918|unkn Harris...| 165.75| 216135201|
|2017-11-24T19:00:...|2076945156| 562903918|McDonald's ccd i...| 119.4| 576697624|
|2017-11-24T19:00:...|2076945132| 83040202|GameStop     arc i...| 242.75| 870108334|
|2017-11-24T19:00:...|2076945098| 2139149619|Trader Joe's   ...| 114.17| 7441192|
+-----+-----+-----+-----+-----+
only showing top 20 rows
```

```
In [20]: # MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER, DISK_ONLY, MEMORY_ONLY_2, MEMORY_AND_DISK_2
import pyspark
```

```
df.persist = df.persist(pyspark.StorageLevel.MEMORY_ONLY_2)
```

```
In [21]: df.persist.write.format("noop").mode("overwrite").save()
```

```
In [22]: # Remove Cache
spark.catalog.clearCache()
```

Note:

For cache we need and action and count and write are always preferred as they will scan the whole dataset resulting in proper caching.

Storage Level:

Default storage level for cache in MEMORY_AND_DISK for dataframes and datasets.

Spark Hands On

1. Distributed Shared Variables
2. Broadcast Variable
3. Accumulators

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Distributed Shared Variables")
    .master("spark://17e348267994:7077")
    .config("spark.cores.max", 16)
    .config("spark.executor.cores", 4)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext

[Spark UI](#)

Version	v3.3.0
Master	spark://17e348267994:7077
AppName	Distributed Shared Variables

```
In [2]: # Read EMP CSV data
```

```
_schema = "first_name string, last_name string, job_title string, dob string, email string, phone string, salary double, dep
emp = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/datasets/employee_records.csv")
```

```
In [3]: # Variable (Lookup)
```

```
dept_names = {1 : 'Department 1',
              2 : 'Department 2',
              3 : 'Department 3',
              4 : 'Department 4',
              5 : 'Department 5',
              6 : 'Department 6',
              7 : 'Department 7',
              8 : 'Department 8',
              9 : 'Department 9',
              10 : 'Department 10'}
```

```
In [4]: # Broadcast the variable
```

```
broadcast_dept_names = spark.sparkContext.broadcast(dept_names)
```

```
In [6]: # Check the value of the variable  
broadcast_dept_names.value
```

```
Out[6]: {1: 'Department 1',  
         2: 'Department 2',  
         3: 'Department 3',  
         4: 'Department 4',  
         5: 'Department 5',  
         6: 'Department 6',  
         7: 'Department 7',  
         8: 'Department 8',  
         9: 'Department 9',  
        10: 'Department 10'}
```

```
In [11]: # Create UDF to return Department name  
  
from pyspark.sql.functions import udf, col  
  
@udf  
def get_dept_names(dept_id):  
    return broadcast_dept_names.value.get(dept_id)
```

```
In [12]: emp_final = emp.withColumn("dept_name", get_dept_names(col("department_id")))
```

```
In [13]: emp_final.show()
```

first_name	last_name	job_title	dob	email	phone	salary	department_id	dept_name
Richard	Morrison	Public relations ...	1973-05-05	melissagarcia@example.com	(699)525-4827	512653.0	8	Department 8
Bobby	McCarthy	Barrister's clerk	1974-04-25	llara@example.net	(750)846-1602x7458	999836.0	7	Department 7
Dennis	Norman	Land/geomatics su...	1990-06-24	jturner@example.net	873.820.0518x825	131900.0	10	Department 10
John	Monroe	Retail buyer	1968-06-16	erik33@example.net	820-813-0557x624	485506.0	1	Department 1
Michelle	Elliott	Air cabin crew	1975-03-31	tiffanyjohnston@example.com	(705)900-5337	604738.0	8	Department 8
Ashley	Montoya	Cartographer	1976-01-16	patrickalexandra@example.com	211.440.5466	483339.0	6	Department 6
Nathaniel	Smith	Quality manager	1985-06-28	lori44@example.net	936-403-3179	419644.0	7	Department 7
Faith	Cummings	Industrial/produc...	1978-07-01	ygordon@example.org	(889)246-5588	205939.0	7	Department 7
Margaret	Sutton	Administrator, ed...	1975-08-16	diana44@example.net	001-647-530-5036x...	671167.0	8	Department 8
Mary	Sutton	Freight forwarder	1979-12-28	ryan36@example.com	422.562.7254x3159	993829.0	7	Department 7
Jake	King	Lexicographer	1994-07-11	monica93@example.org	+1-535-652-9715x6...	702101.0	4	Department 4
Heather	Haley	Music tutor	1981-06-01	stephanie65@example.com	(652)815-7973x298	570960.0	6	Department 6
Thomas	Thomas	Chartered managem...	2001-07-17	pwilliams@example.com	001-245-848-0028x...	339441.0	6	Department 6
Leonard	Carlson	Art therapist	1990-10-18	gabrielmurray@example.com	9247590563	469728.0	8	Department 8
Mark	Wood	Market researcher	1963-10-13	nicholas76@example.com	311.439.1606x3342	582291.0	4	Department 4

```

+-----+
|      Mark|       Wood| Market researcher|1963-10-13|nicholas76@example.com| 311.439.1606x3342|582291.0|        4| Depa
rtment 4|
| Tracey|Washington|Travel agency man...|1986-05-07| mark07@example.com| 001-912-206-6456|146456.0|        4| Depa
rtment 4|
| Rachael| Rodriguez| Media buyer|1966-12-02|griffinmary@example.com| +1-791-344-7586x548|544732.0|        1| Depa
rtment 1|
| Tara|       Liu| Financial adviser|1998-10-12|alexandraobrien@example.com| 216.696.6061|399503.0|        3| Depa
rtment 3|
| Ana| Joseph| Retail manager|1995-01-10| rmorse@example.org| (726)363-7526x9965|761988.0|        10|Depar
tment 10|
| Richard| Hall|Engineer, civil (...|1967-03-02|brandoncardenas@example.com| (964)451-9007x22496|660659.0|        4| Depa
rtment 4|
+-----+
only showing top 20 rows

```

```
In [15]: # Calculate total salary of Department 6

from pyspark.sql.functions import sum
```

```
emp.where("department_id = 6").groupBy("department_id").agg(sum("salary").cast("long")).show()
```

department_id	CAST(sum(salary) AS BIGINT)
6	50294510721

```
In [22]: # Accumulators

dept_sal = spark.sparkContext.accumulator(0)
```

```
In [23]: # Use foreach

def calculate_salary(department_id, salary):
    if department_id == 6:
        dept_sal.add(salary)

emp.foreach(lambda row : calculate_salary(row.department_id, row.salary))
```

```
In [24]: # View total value

dept_sal.value
```

Out[24]: 50294510721.0

```
In [25]: # Stop Spark Session

spark.stop()
```

Note:

Data partitions will be read by all executors in parallel thus is will be in different executors.

Joins can lead to shuffle if not optimized correctly.

Spark Hands On

1. How to Optimize Joins – Big vs Small and Big vs Big table
2. ShuffleHash, SortMerge and BroadCast Joins
3. Bucketing Strategy to load data faster
4. Spark UI to understand tasks/partitions with join data

```
In [2]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Optimizing Joins")
    .master("spark://17e348267994:7077")
    .config("spark.cores.max", 16)
    .config("spark.executor.cores", 4)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[2]: **SparkSession - in-memory**

SparkContext
Spark UI
Version v3.3.0
Master spark://17e348267994:7077
AppName Optimizing Joins

```
In [3]: # Disable AQE and Broadcast join

spark.conf.set("spark.sql.adaptive.enabled", False)
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", False)
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
```

Join Big and Small table - SortMerge vs Broadcast Join

```
In [3]: # Read EMP CSV data

_schema = "first_name string, last_name string, job_title string, dob string, email string, phone string, salary double, dep"
emp = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/datasets/employee_records.csv")
```

```
In [4]: # Read DEPT CSV data

_dept_schema = "department_id int, department_name string, description string, city string, state string, country string"
dept = spark.read.format("csv").schema(_dept_schema).option("header", True).load("/data/input/datasets/department_data.csv")
```

```
In [8]: # Join Datasets
from pyspark.sql.functions import broadcast

df_joined = emp.join(broadcast(dept), on=emp.department_id==dept.department_id, how="left_outer")
```

```
In [9]: df_joined.write.format("noop").mode("overwrite").save()

In [10]: df_joined.explain()

== Physical Plan ==
*(2) BroadcastHashJoin [department_id#7], [department_id#16], LeftOuter, BuildRight, false
:- FileScan csv [first_name#0,last_name#1,job_title#2,dob#3,email#4,phone#5,salary#6,department_id#7] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/data/input/datasets/employee_records.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<first_name:string,last_name:string,job_title:string,dob:string,email:string,phon e:string,s...
+- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [id#160]
  +- *(1) Filter isnotnull(department_id#16)
    + FileScan csv [department_id#16,department_name#17,description#18,city#19,state#20,country#21] Batched: false, DataFilters: [isnotnull(department_id#16)], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/data/input/datasets/department _data.csv], PartitionFilters: [], PushedFilters: [IsNotNull(department_id)], ReadSchema: struct<department_id:int,department_n ame:string,description:string,city:string,state:string,count...
```

Join Big and Big table - SortMerge without Buckets

```
In [9]: # Read Sales data

sales_schema = "transacted_at string, trx_id string, retailer_id string, description string, amount double, city_id string"
sales = spark.read.format("csv").schema(sales_schema).option("header", True).load("/data/input/datasets/new_sales.csv")

In [10]: # Read City data

city_schema = "city_id string, city string, state string, state_abv string, country string"
city = spark.read.format("csv").schema(city_schema).option("header", True).load("/data/input/datasets/cities.csv")

In [7]: # Join Data

df_sales_joined = sales.join(city, on=sales.city_id==city.city_id, how="left_outer")

In [8]: df_sales_joined.write.format("noop").mode("overwrite").save()

In [ ]: # Explain Plan
```

Write Sales and City data in Buckets

```
In [13]: # Write Sales data in Buckets

sales.write.format("csv").mode("overwrite").bucketBy(4, "city_id").option("header", True).option("path", "/data/input/datasets/buckets/sales")
|-----+
| 1     |
|-----+-----+-----+
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|-----+-----+-----+-----+-----+-----+-----+-----+
```

```
In [14]: # Write City data in Buckets

city.write.format("csv").mode("overwrite").bucketBy(4, "city_id").option("header", True).option("path", "/data/input/datasets/buckets/city")
|-----+
| 1     |
|-----+-----+-----+
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|-----+-----+-----+-----+-----+-----+-----+-----+
```

```
In [15]: # Check tables

spark.sql("show tables in default").show()
```

namespace	tableName	isTemporary
default	city_bucket	false
default	sales_bucket	false

Join Sales and City data - SortMerge with Bucket

```
In [16]: # Read Sales table

sales_bucket = spark.read.table("sales_bucket")
```

```
In [17]: # Read City table
city_bucket = spark.read.table("city_bucket")

In [19]: # Join datasets
df_joined_bucket = sales_bucket.join(city_bucket, on=sales_bucket.city_id==city_bucket.city_id, how="left_outer")

In [20]: # Write dataset
df_joined_bucket.write.format("noop").mode("overwrite").save()

In [21]: df_joined_bucket.explain()

== Physical Plan ==
*(3) SortMergeJoin [city_id#176], [city_id#183], LeftOuter
:- *(1) Sort [city_id#176 ASC NULLS FIRST], false, 0
:  +- FileScan csv default.sales_bucket[transacted_at#171, trx_id#172, retailer_id#173, description#174, amount#175, city_id#176]
Batched: false, Bucketed: true, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/data/input/datasets/sales_bucket.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<transacted_at:string, trx_id:string, retailer_id:string, description:string, amount:double, city_id:string>, SelectedBucketsCount: 4 out of 4
+- *(2) Sort [city_id#183 ASC NULLS FIRST], false, 0
   +- *(2) Filter isnotnull(city_id#183)
      +- FileScan csv default.city_bucket[city_id#183, city#184, state#185, state_abv#186, country#187] Batched: false, Bucketed: true, DataFilters: [isnotnull(city_id#183)], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/data/input/datasets/city_bucket.csv], PartitionFilters: [], PushedFilters: [IsNotNull(city_id)], ReadSchema: struct<city_id:string, city:string, state:string, state_abv:string, country:string>, SelectedBucketsCount: 4 out of 4

In [ ]: # View how tasks are reading Bucket data
```

Spark Hands On

1. Resource Management on Cluster
2. Static and Dynamic Allocation
3. How to configure Dynamic Allocation on Standalone Cluster.
4. How Dynamic Allocation is different than Databricks Cluster Scale UP

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Dynamic Allocation")
    .master("spark://197e20b418a6:7077")
    .config("spark.executor.cores", 2)
    .config("spark.executor.memory", "512M")
    .config("spark.dynamicAllocation.enabled", True)
    .config("spark.dynamicAllocation.minExecutors", 0)
    .config("spark.dynamicAllocation.maxExecutors", 5)
    .config("spark.dynamicAllocation.initialExecutors", 1)
    .config("spark.dynamicAllocation.shuffleTracking.enabled", True)
    .config("spark.dynamicAllocation.executorIdleTimeout", "60s")
    .config("spark.dynamicAllocation.cachedExecutorIdleTimeout", "60s")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

[SparkContext](#)

[Spark UI](#)

Version	v3.3.0
Master	spark://197e20b418a6:7077
AppName	Dynamic Allocation

```
In [2]: # Read Sales data

sales_schema = "transacted_at string, trx_id string, retailer_id string, description string, amount double, city_id string"
sales = spark.read.format("csv").schema(sales_schema).option("header", True).load("/data/input/new_sales.csv")
```

```
In [3]: # Read City data

city_schema = "city_id string, city string, state string, state_abv string, country string"
city = spark.read.format("csv").schema(city_schema).option("header", True).load("/data/input/cities.csv")
```

```
In [4]: # Join Data

df_sales_joined = sales.join(city, on=sales.city_id==city.city_id, how="left_outer")
```

```
In [5]: df_sales_joined.write.format("noop").mode("overwrite").save()
```

```
In [ ]: # Difference between Scale UP in Databricks and Dynamic Allocation
```

Spark Hands On

1. What is Skewness ?
2. How Skewness leads to Spillage ?
3. Fix Skewness with Salting Technique

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Optimizing Skewness and Spillage")
    .master("spark://197e20b418a6:7077")
    .config("spark.cores.max", 8)
    .config("spark.executor.cores", 4)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

Out[1]: **SparkSession - in-memory**

SparkContext	
Spark UI	View
Version	v3.3.0
Master	spark://197e20b418a6:7077
AppName	Optimizing Skewness and Spillage


```
In [2]: # Disable AQE and Broadcast join

spark.conf.set("spark.sql.adaptive.enabled", False)
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", False)
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
```



```
In [3]: # Read Employee data
_schema = "first_name string, last_name string, job_title string, dob string, email string, phone string, salary double, department_id int, department_name string, description string, city string, state string, country string"
emp = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/employee_records_skewed.csv")
```



```
In [4]: # Read DEPT CSV data
_dept_schema = "department_id int, department_name string, description string, city string, state string, country string"
dept = spark.read.format("csv").schema(_dept_schema).option("header", True).load("/data/input/department_data.csv")
```



```
In [5]: # Join Datasets

df_joined = emp.join(dept, on=emp.department_id==dept.department_id, how="left_outer")
```



```
In [6]: df_joined.write.format("noop").mode("overwrite").save()
```



```
In [ ]: #Explain Plan

df_joined.explain()
```



```
In [7]: # Check the partition details to understand distribution
from pyspark.sql.functions import spark_partition_id, count, lit

part_df = df_joined.withColumn("partition_num", spark_partition_id()).groupBy("partition_num").agg(count(lit(1)).alias("count"))
part_df.show()
```

partition_num	count
103	19860
122	420474
43	19899
107	19928
49	20006
51	19829
102	20099
66	20172
174	20229
89	419504

```
In [8]: # Verify Employee data based on department_id
from pyspark.sql.functions import count, lit, desc, col

emp.groupBy("department_id").agg(count(lit(1))).show()
```

department_id	count(1)
1	19899
6	20006
3	19829
5	20172
9	419504
4	20099
8	19860
7	19928
10	420474
2	20229

```
In [27]: # Set shuffle partitions to a lesser number - 16

spark.conf.set("spark.sql.shuffle.partitions", 32)
```

```
In [28]: # Let prepare the salt
import random
from pyspark.sql.functions import udf

# UDF to return a random number every time and add to Employee as salt
@udf
def salt_udf():
    return random.randint(0, 32)

# Salt Data Frame to add to department
salt_df = spark.range(0, 32)
salt_df.show()
```

id
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

```
In [29]: # Salted Employee
from pyspark.sql.functions import lit, concat

salted_emp = emp.withColumn("salted_dept_id", concat("department_id", lit("_"), salt_udf()))

salted_emp.show()
```

first_name last_name	job_title	dob	email	phone	salary	department_id salted_dept_id
Samantha Brown	Diagnostic radiog...	1966-06-11	jwatson@example.com	(428)806-5154	439679.0	3 3_5
Justin Castaneda	Human resources o...	1996-11-11	sdavis@example.org	001-581-642-9621	97388.0	4 4_23
Carl Peterson	Proofreader	1984-11-23	andrew20@example.net	241-871-9102x3835	287728.0	1 1_5
Catherine Lane	Location manager	1966-06-21	elizabethalexande...	470.866.4415x0739	174151.0	3 3_25
Aaron Delgado	Teacher, secondar...	1972-10-11	uwilliams@example...	384.336.5759x4831	209013.0	8 8_17
Michelle Hill	Customer service ...	1984-01-15	antoniojoseph@exa...	368.485.0685x793	764126.0	8 8_15
Kristina Martin	IT consultant	1964-02-23	autumn05@example.com	(625)327-0615	563768.0	1 1_9
Carol Nichols	Phytotherapist	1969-02-14	crawfordsarah@exa...	422-490-1069x38089	156689.0	10 10_19
Peter Hill	Cytogeneticist	1964-09-23	xholt@example.org	935.573.8160	957436.0	5 5_23
Benjamin Lopez	Agricultural engi...	1966-01-20	ryan46@example.org	+1-256-376-8069x339	891725.0	1 1_5
Susan Savage	Optician, dispensing	1996-07-27	taylorjoshua@exam...	+1-393-821-5515x816	198396.0	6 6_5
Robert Cox	Occupational ther...	1993-06-27	anthony00@example...	8008487748	907659.0	3 3_32
Evan Terry	Local government ...	1982-01-03	srodriguez@exAMPL...	220-913-4625	693419.0	5

```
In [30]: # Salted Department

salted_dept = dept.join(salt_df, how="cross").withColumn("salted_dept_id", concat("department_id", lit("_"), "id"))

salted_dept.where("department_id = 9").show()
```

department_id	department_name	description	city state country	id salted_dept_id
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 0	9_0	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 1	9_1	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 2	9_2	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 3	9_3	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 4	9_4	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 5	9_5	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 6	9_6	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 7	9_7	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 8	9_8	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 9	9_9	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 10	9_10	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 11	9_11	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 12	9_12	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 13	9_13	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 14	9_14	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 15	9_15	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 16	9_16	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 17	9_17	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 18	9_18	
9 Mcmahon, Terrell	De-engineered hig...	Marychester MN Italy 19	9_19	

only showing top 20 rows

```
In [31]: # Lets make the salted join now
salted_joined_df = salted_emp.join(salted_dept, on=salted_emp.salted_dept_id==salted_dept.salted_dept_id, how="left_outer")
```

```
In [ ]:
```

```
In [33]: salted_joined_df.write.format("noop").mode("overwrite").save()
```

```
In [32]: # Check the partition details to understand distribution
from pyspark.sql.functions import spark_partition_id, count

part_df = salted_joined_df.withColumn("partition_num", spark_partition_id()).groupBy("partition_num").agg(count(lit(1)).alias("count"))

part_df.show()

+-----+----+
|partition_num|count|
+-----+----+
|      18|30975|
|      12|28636|
|      10|17942|
|      27|58641|
|      1|28039|
|      3|31642|
|      20|29552|
|      29| 4860|
|      13|20105|
|      14|18831|
|      23|81155|
|      6|18818|
|      9|55094|
|      11|44418|
|      26| 3006|
|      7|30275|
|      30| 4220|
|      28|16504|
|      0|30887|
|      8|30997|
+-----+----+
only showing top 20 rows
```

What is skewness?

-Basically unbalanced data not distributed properly.

2 types of spillage -Spillage memory & Spillage disk.

How to identify Skewness?

-We can find the partition count for the shuffled data.

Salted keys: Creation of new salted keys for joining at both tables.

Note: Salting Technique doesn't need to be used everytime, make sure to use only when you have issues of memory due to spillage(mostly out of memory errors).

This was just a demo to show you how to fix spillage.

Spark Hands On

1. What is AQE (Adaptive Query Execution) ?
2. Benefits of AQE
3. Impact of AQE in Skew Joins, Sort Merge Joins & Broadcast Joins
4. Coalescing post-shuffle partitions with AQE

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("AQE in Spark")
    .master("spark://197e20b418a6:7077")
    .config("spark.cores.max", 8)
    .config("spark.executor.cores", 4)
    .config("spark.executor.memory", "512M")
    .getOrCreate()
)

spark
```

```
Out[1]: SparkSession - in-memory
```

SparkContext
Spark UI
Version v3.3.0
Master spark://197e20b418a6:7077
AppName AQE in Spark

```
In [2]: # Disable AQE and Broadcast join
spark.conf.set("spark.sql.adaptive.enabled", False)
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", False)
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
```

```
In [3]: # Read Employee data
_schema = "first_name string, last_name string, job_title string, dob string, email string, phone string, salary double, dep"
emp = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/employee_records_skewed.csv")
```

```
In [4]: # Read DEPT CSV data
_dept_schema = "department_id int, department_name string, description string, city string, state string, country string"
dept = spark.read.format("csv").schema(_dept_schema).option("header", True).load("/data/input/department_data.csv")
```

```
In [14]: # Join Datasets
df_joined = emp.join(dept, on=emp.department_id==dept.department_id, how="left_outer")
```

```
In [15]: df_joined.write.format("noop").mode("overwrite").save()
```

```
In [6]: #Explain Plan
df_joined.explain()

== Physical Plan ==
*(4) SortMergeJoin [department_id#7], [department_id#16], LeftOuter
:- *(1) Sort [department_id#7 ASC NULLS FIRST], false, 0
:  +- Exchange hashpartitioning(department_id#7, 200), ENSURE_REQUIREMENTS, [id#17]
:    +- FileScan csv [first_name#0,last_name#1,job_title#2,dob#3,email#4,phone#5,salary#6,department_id#7] Batched: false, D
ataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/data/input/employee_records_skewed.csv], PartitionFil
ters: [], PushedFilters: [], ReadSchema: struct<first_name:string,last_name:string,job_title:string,dob:string,email:string,p
hone:string,s...
+- *(3) Sort [department_id#16 ASC NULLS FIRST], false, 0
   +- Exchange hashpartitioning(department_id#16, 200), ENSURE_REQUIREMENTS, [id#29]
      +- *(2) Filter isnotnull(department_id#16)
         +- FileScan csv [department_id#16,department_name#17,description#18,city#19,state#20,country#21] Batched: false, Dat
aFilters: [isnotnull(department_id#16)], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/data/input/department_data.c
sv], PartitionFilters: [], PushedFilters: [IsNotNull(department_id)], ReadSchema: struct<department_id:int,department_name:st
ring,description:string,city:string,state:string,count...

In [12]: # Coalescing post-shuffle partitions - remove un-necessary shuffle partitions
# Skewed join optimization (balance partitions size) - join smaller partitions and split bigger partition

spark.conf.set("spark.sql.adaptive.enabled", True)
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", True)

In [13]: # Fix partition sizes to avoid Skew

spark.conf.set("spark.sql.adaptive.advisoryPartitionSizeInBytes", "8MB") #Default value: 64MB
spark.conf.set("spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes", "10MB") #Default value: 256MB

In [16]: # Converting sort-merge join to broadcast join

spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "10MB")

In [17]: # Join Datasets - without specifying specific broadcast table

df_joined = emp.join(dept, on=emp.department_id==dept.department_id, how="left_outer")

In [18]: df_joined.write.format("noop").mode("overwrite").save()
```

Skewed Partitions Optimization:

Spark AQE will take care of the partition sizing to balance the data and avoid spillage.

Spark.sql.adaptive.advisoryPartitionSizeInBytes:

The size in bytes of the shuffle partition during adaptive optimization. This can control the size of shuffle partitions post-shuffle.

Now that our task partition memory is small we will make the partitions smaller in size to 8MB.

Spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes:

A partition is considered as skewed if its size in bytes is larger than this threshold.

Spark Hands On

1. Spark SQL
2. Understand Different Catalog options
3. How to persist metadata in Spark
4. Hints in Spark SQL

```
In [1]: # Spark Session
from pyspark.sql import SparkSession

spark = (
    SparkSession
        .builder
        .appName("Spark SQL")
        .master("local[*]")
        .enableHiveSupport()
        .config("spark.sql.warehouse.dir", "/data/output/spark-warehouse")
        .getOrCreate()
)

spark
```

Out[1]: **SparkSession - hive**

[SparkContext](#)

[Spark UI](#)

Version	v3.3.0
Master	local[*]
AppName	Spark SQL

```
In [9]: # Read Employee data
_schema = "first_name string, last_name string, job_title string, dob string, email string, phone string, salary double, de
emp = spark.read.format("csv").schema(_schema).option("header", True).load("/data/input/employee_records_skewed.csv")
```

```
In [10]: # Read DEPT CSV data
_dept_schema = "department_id int, department_name string, description string, city string, state string, country string"
dept = spark.read.format("csv").schema(_dept_schema).option("header", True).load("/data/input/department_data.csv")
```

```
In [2]: # Spark Catalog (Metadata) - in-memory/hive
spark.conf.get("spark.sql.catalogImplementation")
```

```
Out[2]: 'hive'
```

```
In [3]: # Show databases
db = spark.sql("show databases")
db.show()
```

namespace
default

```
In [4]: spark.sql("show tables in default").show()
```

namespace	tableName	isTemporary
default	emp_final	false

```
In [12]: # Register dataframes are temp views
emp.createOrReplaceTempView("emp_view")
dept.createOrReplaceTempView("dept_view")
```

```
In [ ]: # Show tables/view in catalog
```

```
In [16]: # View data from table
emp_filtered = spark.sql("""
    select * from emp_view
    where department_id = 1
""")
```

```
In [17]: emp_filtered.show()
```

first_name	last_name	job_title	dob	email	phone	salary	department_id
Carl	Peterson	Proofreader	1984-11-23	andrew20@example.net	241-871-9102x3835 287728.0		1
Kristina	Martin	IT consultant	1964-02-23	autumn05@example.com	(625)327-0615 563768.0		1
Benjamin	Lopez	Agricultural engi...	1966-01-20	ryan46@example.org	+1-256-376-8069x339 891725.0		1
Leslie	Rodriguez	Horticulturist, c...	1973-06-16	thomassutton@exam...	001-630-539-4136x... 875940.0		1
Angela	Martin	Company secretary	1979-07-07	qholmes@example.org	001-267-831-8987x... 485302.0		1
Julia	Gomez	Advice worker	1963-03-02	xjackson@example.net	495-667-0287 59589.0		1
Jeremy	Hunt	Radiation protect...	1987-12-06	mraeric@example.net	6276248675 199026.0		1
Alice	Rice	Advertising art d...	1964-04-03	amy82@example.net	2208895255 967862.0		1
Clinton	Cunningham	Designer, exhibit...	1984-07-22	margaret29@example...	(526)879-8418x297 660617.0		1
Susan	Ford	Geographical info...	1985-05-06	sherrirobinson@ex...	(734)226-1079 991902.0		1
Gregory	Stevens	Engineer, buildin...	1997-11-17	sgreen@example.org	676-481-8023 986664.0		1
Shannon	Bowen	Industrial/produc...	1973-05-07	gharris@example.com	(891)559-1318x860 201522.0		1
Dylan	Sullivan	Television camera...	1999-06-23	michaelallison@ex...	001-757-795-1822x... 715727.0		1
Christopher	Lozano	Database administ...	1979-03-15	t hernandez@example...	(465)990-3733x09035 441011.0		1
William	Pope	Best boy	1983-05-07	jillianmorrison@e...	303.475.9516 632646.0		1
Kevin	Brown	Engineer, mining	1965-09-26	williamroy@example...	+1-771-254-4086x1... 61066.0		1
Michaela	Mckinney	Teacher, English ...	1997-10-04	brian65@example.net	001-384-754-4223x... 750122.0		1
Pamela	Lee	Land	1974-04-28	bartonangela@exam...	001-226-781-4414x... 157355.0		1
Robert	Davis	Museum education ...	1965-12-31	savagesamantha@ex...	001-487-554-6158 833494.0		1
Jeremy	Alvarez	Tour manager	1998-11-04	kellyebrittany@ex...	+1-960-957-6641x9... 84979.0		1

only showing top 20 rows

```
In [21]: # Create a new column dob_year and register as temp view

emp_temp = spark.sql("""
    select e.* , date_format(dob, 'yyyy') as dob_year from emp_view e
""")
```

```
In [22]: emp_temp.createOrReplaceTempView("emp_temp_view")
```

```
In [24]: spark.sql("select * from emp_temp_view").show()
```

first_name	last_name	job_title	dob	email	phone	salary	department_id	dob_ye
Samantha	Brown	Diagnostic radiog...	1966-06-11	jwatson@example.com	(428)806-5154 439679.0		3	19
Justin	Castaneda	Human resources o...	1996-11-11	sdavis@example.org	001-581-642-9621 97388.0		4	19
Carl	Peterson	Proofreader	1984-11-23	andrew20@example.net	241-871-9102x3835 287728.0		1	19
Catherine	Lane	Location manager	1966-06-21	elizabethalexande...	470.866.4415x0739 174151.0		3	19
Aaron	Delgado	Teacher, secondar...	1972-10-11	uwilliams@example...	384.336.5759x4831 209013.0		8	19
Michelle	Hill	Customer service ...	1984-01-15	antoniojoseph@exa...	368.485.0685x793 764126.0		8	19
Kristina	Martin	IT consultant	1964-02-23	autumn05@example.com	(625)327-0615 563768.0		1	19
Carol	Nichols	Phytotherapist	1969-02-14	crawfordsarah@exa...	422-490-1069x38089 156689.0		10	19
Peter	Hill	Cytogeneticist	1964-09-23	xholt@example.org	935.573.8160 957436.0		5	19
Benjamin	Lopez	Agricultural engi...	1966-01-20	ryan46@example.org	+1-256-376-8069x339 891725.0		1	19
Susan	Savage	Optician, dispensing	1996-07-27	taylorjoshua@exam...	+1-393-821-5515x816 198396.0		6	19
Robert	Cox	Occupational ther...	1993-06-27	anthony00@example...	8008487748 907659.0		3	19
Evan	Terry	Local government ...	1982-01-03	srodriguez@example...	220-913-4625 693419.0		5	19

```

82|          . . .
|  Justin| Santiago|           Make|1965-03-20| birdjoe@example.org| 801-317-7926|815251.0|    7|   19
65|
|  Rose| Gregory|        Barrister|1974-10-31|samuel27@example.net| 923-304-9438|673811.0|    2|   19
74|
| Nicholas| Short| Charity fundraiser|1998-10-03| brian12@example.com| 855.973.7301|538901.0|    4|   19
98|
| John| Hanson|Lecturer, higher ...|2001-04-12| zweiss@example.org| (453)740-2558|247223.0|    7|   20
01|
| Bryan| Turner|Public relations ...|1984-01-30| ssmith@example.com| 426.547.0413x20201|286799.0|    6|   19
84|
| Tonya| Schultz|Contracting civil...|1982-05-07|douglas54@example...| 578-916-7661|664105.0|    4|   19
82|
| Patricia| Anderson|      Set designer|1974-09-22|joshua92@example.net|001-765-729-3973x...|439446.0|    2|   19
74|
+-----+-----+-----+-----+-----+-----+-----+
--+
only showing top 20 rows

```

In [13]: # Join emp and dept - HINTS

```

emp_final = spark.sql("""
    select /*+ BROADCAST(d) */
    e.* , d.department_name
    from emp_view e left outer join dept_view d
    on e.department_id = d.department_id
""")

```

In [9]: # Show emp data

```

emp_final.show()

+-----+-----+-----+-----+-----+-----+-----+
|first_name|last_name|       job_title|       dob|        email|      phone| salary|department_id| department_name|
+-----+-----+-----+-----+-----+-----+-----+
| Samantha| Brown|Diagnostic radiog...|1966-06-11| jwatson@example.com| (428)806-5154|439679.0|    3|Pittman, Hess and...
| Justin| Castaneda|Human resources o...|1996-11-11| sdavis@example.org| 001-581-642-9621| 97388.0|    4|Smith, Snyder and...
| Carl| Peterson|Proofreader|1984-11-23|andrew20@example.net| 241-871-9102x3835|287728.0|    1|Bryan-James|
| Catherine| Lane| Location manager|1966-06-21|elizabethalexander...| 470.866.4415x0739|174151.0|    3|Pittman, Hess and...
| Aaron| Delgado|Teacher, secondar...|1972-10-11|uwilliams@example...| 384.336.5759x4831|209013.0|    8|Parker PLC|
| Michelle| Hill|Customer service ...|1984-01-15|antoniojoseph@exa...| 368.485.0685x793|764126.0|    8|Parker PLC|
| Kristina| Martin|IT consultant|1964-02-23|autumn05@example.com| (625)327-0615|563768.0|    1|Bryan-James|
| Carol| Nichols| Phytotherapist|1969-02-14|crawfordsarah@exa...| 422-490-1069x38089|156689.0|    10|Delgado-Keller|
| Peter| Hill|Cytogeneticist|1964-09-23| xholt@example.org| 935.573.8160|957436.0|    5|Hardin Inc|
| Benjamin| Lopez|Agricultural engi...|1966-01-20| ryan46@example.org| +1-256-376-8069x339|891725.0|    1|Bryan-James|
| Susan| Savage|Optician, dispensing|1996-07-27|taylorjoshua@exam...| +1-393-821-5515x816|198396.0|    6|Sanders LLC|
| Robert| Cox|Occupational ther...|1993-06-27|anthony00@example...| 8008487748|907659.0|    3|Pittman, Hess and...
| Evan| Terry|Local government ...|1982-01-03|srodriguez@exampl...| 220-913-4625|693419.0|    5|Hardin Inc|
| Justin| Santiago|           Make|1965-03-20| birdjoe@example.org| 801-317-7926|815251.0|    7|

```

```

Hardin Inc|
| Justin| Santiago| Make|1965-03-20| birdjoe@example.org| 801-317-7926|815251.0| 7|
Ward-Gordon|
| Rose| Gregory| Barrister|1974-10-31|samuel27@example.net| 923-304-9438|673811.0| 2|Smith,
Craig and ...|
| Nicholas| Short| Charity fundraiser|1998-10-03| brian12@example.com| 855.973.7301|538901.0| 4|Smith,
Snyder and...|
| John| Hanson|Lecturer, higher ...|2001-04-12| zweiss@example.org| (453)740-2558|247223.0| 7|
Ward-Gordon|
| Bryan| Turner|Public relations ...|1984-01-30| ssmith@example.com| 426.547.0413x20201|286799.0| 6|
Sanders LLC|
| Tonya| Schultz|Contracting civil...|1982-05-07|douglas54@example...| 578-916-7661|664105.0| 4|Smith,
Snyder and...|
| Patricia| Anderson| Set designer|1974-09-22|joshua92@example.net|001-765-729-3973x...|439446.0| 2|Smith,
Craig and ...|
+-----+-----+-----+-----+-----+-----+-----+
-----+
only showing top 20 rows

```

```
In [14]: # Write the data as Table
emp_final.write.format("parquet").saveAsTable("emp_final")

In [5]: # Read the data from Table
emp_new = spark.sql("select * from emp_final")
```

```
In [6]: emp_new.show()
```

first_name	last_name	job_title	dob	email	phone	salary	department_id	d
Samantha	Brown	Diagnostic radiog...	1966-06-11	jwatson@example.com	(428)806-5154	439679.0	3 Pittma	
Justin	Castaneda	Human resources o...	1996-11-11	sdavis@example.org	001-581-642-9621	97388.0	4 Smith,	
Carl	Peterson	Proofreader	1984-11-23	andrew20@example.net	241-871-9102x3835	287728.0	1	
Catherine	Lane	Location manager	1966-06-21	elizabethalexande...	470.866.4415x0739	174151.0	3 Pittma	
Aaron	Delgado	Teacher, secondar...	1972-10-11	uwilliams@example...	384.336.5759x4831	209013.0	8	
Michelle	Hill	Customer service ...	1984-01-15	antoniojoseph@exa...	368.485.0685x793	764126.0	8	
Kristina	Martin	IT consultant	1964-02-23	autumn05@example.com	(625)327-0615	563768.0	1	
Carol	Nichols	Phytotherapist	1969-02-14	crawfordsarah@exa...	422-490-1069x38089	156689.0	10	
Peter	Hill	Cytogeneticist	1964-09-23	xholt@example.org	935.573.8160	957436.0	5	
Benjamin	Lopez	Agricultural engi...	1966-01-20	ryan46@example.org	+1-256-376-8069x339	891725.0	1	
Susan	Savage	Optician, dispensing	1996-07-27	taylorjoshua@exam...	+1-393-821-5515x816	198396.0	6	
Robert	Cox	Occupational ther...	1993-06-27	anthony00@example...	8008487748	907659.0	3 Pittma	
Evan	Terry	Local government ...	1982-01-03	srodriguez@exAMPL...	220-913-4625	693419.0	5	
Justin	Santiago	Make	1965-03-20	birdjoe@example.org	801-317-7926	815251.0	7	
Rose	Gregory	Barrister	1974-10-31	samuel27@example.net	923-304-9438	673811.0	2 Smith,	
Nicholas	Short	Charity fundraiser	1998-10-03	brian12@example.com	855.973.7301	538901.0	4 Smith.	

```
In [ ]: # Persist metadata
```

```
In [8]: # Show details of metadata
spark.sql("describe extended emp_final").show()
```

col_name	data_type	comment
first_name	string	null
last_name	string	null
job_title	string	null
dob	string	null
email	string	null
phone	string	null
salary	double	null
department_id	int	null
department_name	string	null
# Detailed Table ...		
Database	default	
Table	emp_final	
Owner	root	
Created Time	Sat Jan 06 10:03:...	
Last Access	UNKNOWN	
Created By	Spark 3.3.0	
Type	MANAGED	
Provider	parquet	
Statistics	37700526 bytes	

only showing top 20 rows

What is Catalog?

-Catalog stores the metadata of SQL Objects.

Temp Views:

Temp views are only available until the session is active.

Spark SQL Functions:

PySpark DataFrames API functions are available for Spark SQL by default. No need for any import.

AZURE COSMOS DB

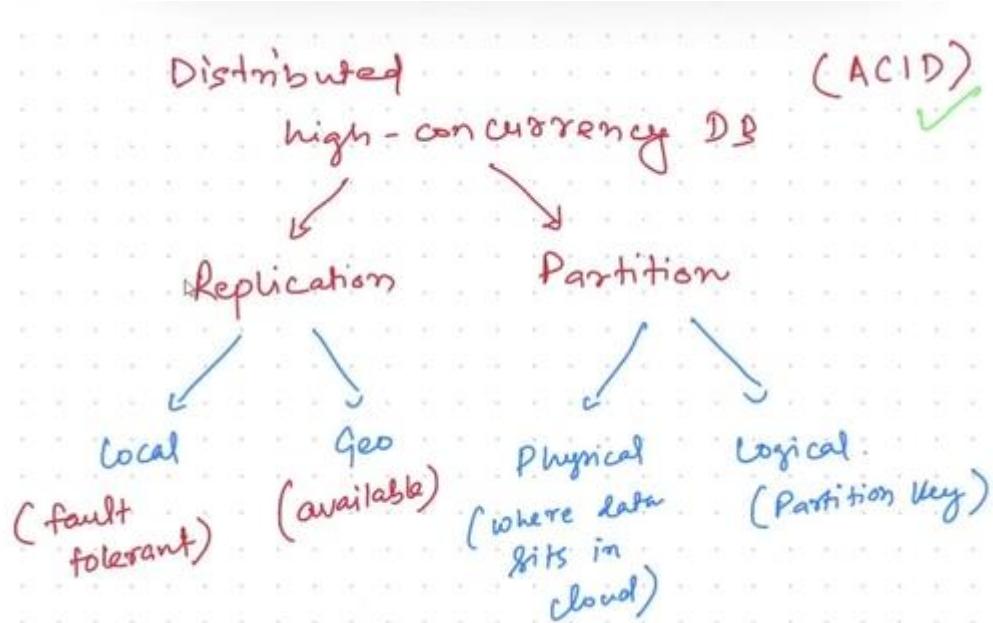
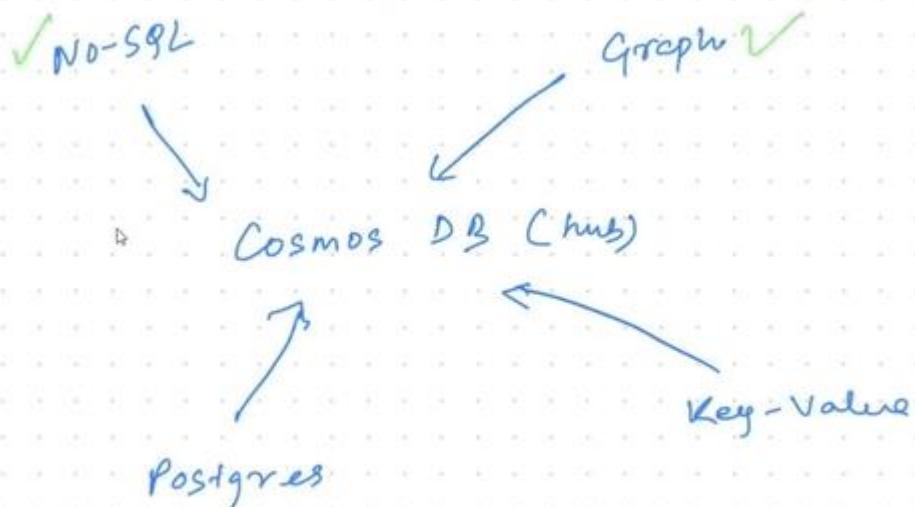
- WHAT IS NO-SQL DB?
- WHY IS COSMOS DB SO POPULAR?
- HOW TO SETUP COSMOS DB?
- HOW TO READ AND WRITE FROM COSMOS DB USING APACHE SPARK?

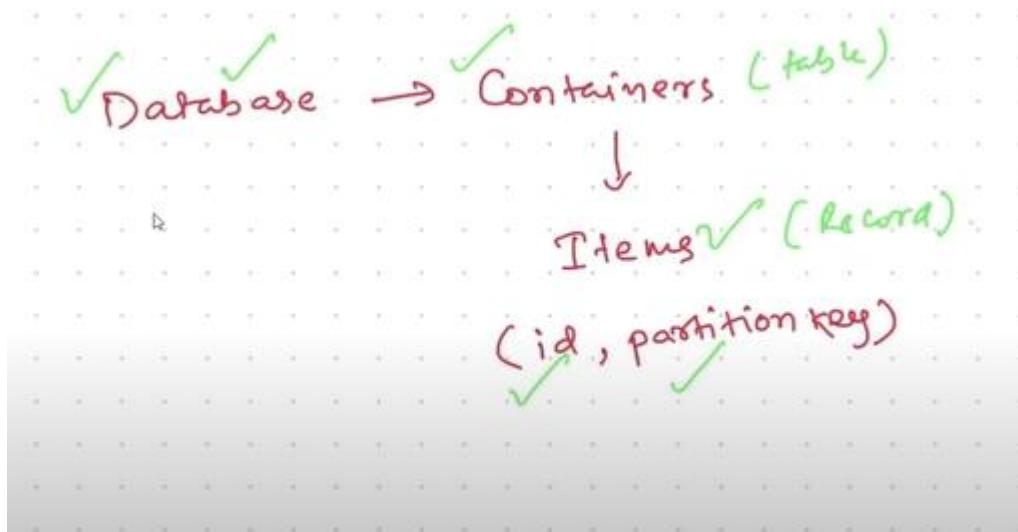
NoSQL

- No Fixed Schema ✓
- Majority focus on Scaling Out
- Supports variety of Data

SQL

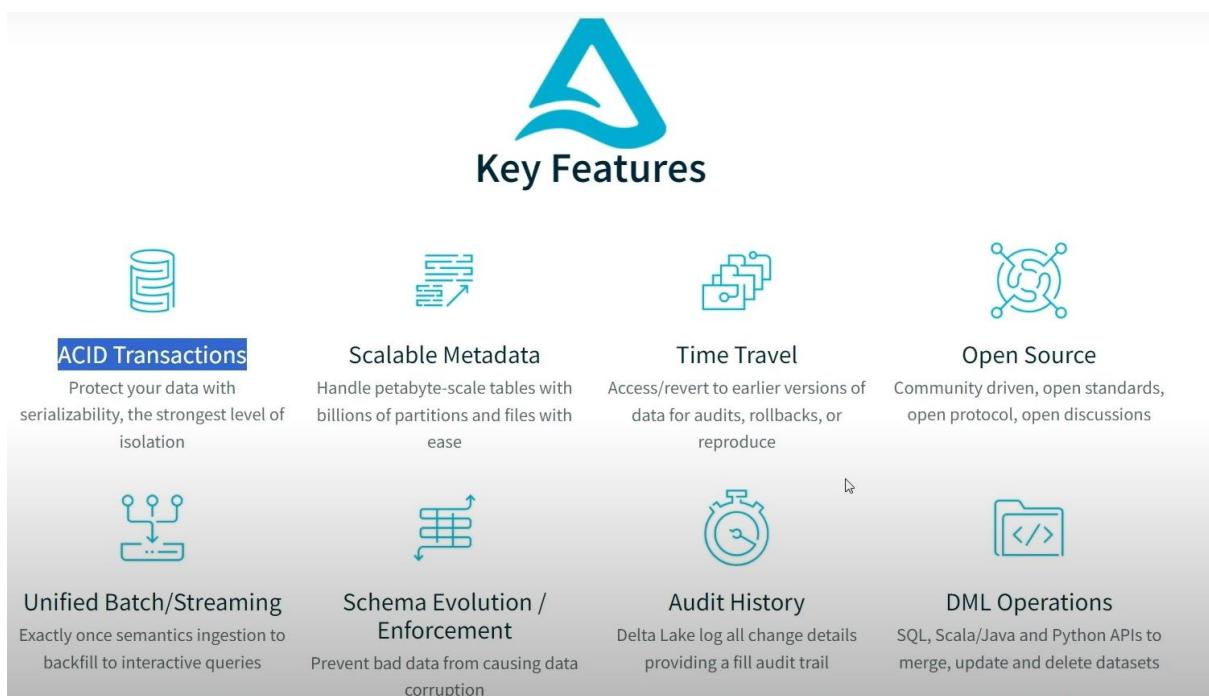
- Fixed Schema ✓
- Majority focus on Scaling Up
- Supports majority Structured data





Using Secrets Securely:

This strategy works for on-premise or standard clusters. If you are working with Databricks, make sure to save secrets in Azure Key Vault and import them while reading or writing data.



```
In [0]: # Spark Session
spark
```

SparkSession - hive

SparkContext

Spark UI

Version

v3.3.2

Master

local[8]

AppName

Databricks Shell

```
In [0]: # Default Catalog for Databricks
spark.conf.get("spark.sql.catalogImplementation")
```

Out[2]: 'hive'

```
In [0]: %sql
show databases
```

databaseName

default

```
In [0]: # Read Sales parquet data
df_sales = spark.read.parquet("/data/input/sales_data.parquet")
```

```
In [0]: # View data
display(df_sales)
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-11-24T19:00:00.000+0000	1995601912	2077350195	Walgreen	11-25	197.23
2017-11-24T19:00:00.000+0000	1734117021	644879053	unkn ppd id: 768641	11-26	8.58
2017-11-24T19:00:00.000+0000	1734117022	847200066	Wal-Mart ppd id: 555914	Algiers	11-26
2017-11-24T19:00:00.000+0000	1734117030	1953761884	Home Depot ppd id: 265293	11-25	384.5
2017-11-24T19:00:00.000+0000	1734117089	1898522855	Target	11-25	66.33
2017-11					

```
In [0]: # Read a particular version - pyspark api
df_sales_delta = spark.read.table("sales_delta@v1")

display(df_sales_delta.where("trx_id = '1734117021'"))
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-11-24T19:00:00.000+0000	1734117021	644879053	unkn ppd id: 768641	11-26	0.0

```
In [0]: %sql

select * from sales_delta@v0 where trx_id = '1734117021'
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-11-24T19:00:00.000+0000	1734117021	644879053	unkn ppd id: 768641	11-26	8.58

```
In [0]: df_new = spark.sql("select *, current_timestamp() as time_now from sales_delta@v0 where trx_id = '1734117021'")

display(df_new)
```

transacted_at	trx_id	retailer_id	description	amount	city_id	time_now
2017-11-24T19:00:00.000+0000	1734117021	644879053	unkn ppd id: 768641	11-26	8.58	930259917 2024-05-26T07:30:03.636+0000

```
In [0]: # Append data to existing delta table
df_new.write.format("delta").mode("append").option("mergeSchema", True).option("path", "/data/output/sales_delta_1").saveAsTable("sales_delta")

In [0]: 
```

```
In [0]: # View data
```

```
In [0]: # Reading Delta Table using Delta Libraries
from delta import DeltaTable

dt = DeltaTable.forName(spark, "sales_delta")

display(dt.history())
```

version	timestamp	userId	userName	operation	operationParameters	job	notebook	clu
2	2024-05-26T07:31:26.000+0000	5635101767402432	easewithdata@gmail.com	WRITE	Map(mode -> Append, partitionBy -> [])	null List(954438656804733)	0	p7fi
1	2024-05-26T07:16:09.000+0000	5635101767402432	easewithdata@gmail.com	UPDATE	Map(predicate -> [(trx_id#2208 = null 1734117021)])	null List(954438656804733)	0	p7fi

```
In [0]: # Converting a Parquet to Delta - Check and Convert
DeltaTable.isDeltaTable(spark, "/data/output/sales_delta_1")
DeltaTable.convertToDelta(spark, "parquet./data/output/sales_parquet_1")

Out[67]: <delta.tables.DeltaTable at 0x7fe25dd57820>
```

```
In [0]: # Validate
display(dbutils.fs.ls("/data/output/sales_parquet_1"))
```

path	name	size	modificationTime
dbfs:/data/output/sales_parquet_1/_SUCCESS	_SUCCESS	0	1716706626000
dbfs:/data/output/sales_parquet_1/_committed_6541294181759099722	_committed_6541294181759099722	122	1716706626000
dbfs:/data/output/sales_parquet_1/_delta_log/	_delta_log/	0	0
dbfs:/data/output/sales_parquet_1/_started_6541294181759099722	_started_6541294181759099722	0	1716706616000
dbfs:/data/output/sales_parquet_1/part-00000-tid-6541294181759099722-63ab469e-b892-46da-81bc-6924b0bb28c4-3-1-c000.snappy.parquet	part-00000-tid-6541294181759099722-63ab469e-b892-46da-81bc-6924b0bb28c4-3-1-c000.snappy.parquet	27539240	1716706625000

```
In [0]: %sql
```

```
describe extended sales_parquet
```

col_name	data_type	comment
transacted_at	timestamp	null
trx_id	int	null
retailer_id	int	null
description	string	null
amount	double	null
city_id	int	null

```
# Detailed Table Information
```

Catalog	spark_catalog
Database	default
Table	sales_parquet
Created Time	Sun May 26 06:57:07 UTC 2024
Last Access	UNKNOWN
Created By	Spark 3.3.2
Type	EXTERNAL
Location	dbfs:/data/output/sales_parquet_1
Provider	delta
Owner	root

```
In [0]: %sql
```

```
CONVERT TO DELTA sales_parquet
```

```
In [0]: %sql
```

```
RESTORE TABLE sales_delta TO VERSION AS OF 1
```

table_size_after_restore	num_of_files_after_restore	num_removed_files	num_restored_files	removed_files_size	restored_files_size
27539244	1	1	0	2362	0

```
In [0]: %sql
```

```
select * from sales_delta where trx_id = '1734117021'
```

transacted_at	trx_id	retailer_id	description	amount	city_id
2017-11-24T19:00:00.000+0000	1734117021	644879053	unkn ppd id: 768641 11-26	0.0	930259917

```
In [0]:
```

```
spark.conf.set("spark.databricks.delta.retentionDurationCheck.enabled","false")
```

```
dt = DeltaTable.forName(spark, "sales_delta")
```

```
dt.vacuum(0)
```

```
Out[81]: DataFrame[]
```

```
In [0]:
```

```
display(dbutils.fs.ls("/data/output/sales_delta_1"))
```

```
In [0]: display(dbutils.fs.ls("/data/output/sales_delta_1"))
```

path	name	size	modificationTime
dbfs:/data/output/sales_delta_1/_delta_log/	_delta_log/	0	0
dbfs:/data/output/sales_delta_1/part-00000-a5c1cf7a-0fab-4b45-a914-8d5c3dfb1ea5-c000.snappy.parquet	part-00000-a5c1cf7a-0fab-4b45-a914-8d5c3dfb1ea5-c000.snappy.parquet	27539244	1716707768000

```
In [0]: %sql  
select * from sales_delta@v0 where trx_id = '1734117021'
```

org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 262.0 failed 1 times, most recent failure: Lost task 0.0 in stage 262.0 (TID 1616) (ip-10-172-218-81.us-west-2.compute.internal executor driver): com.databricks.sql.io.FileReadException: Error while reading file dbfs:/data/output/sales_delta_1/part-00000-52d6e56f-5f92-4280-b5f1-db1153aba3b8-c000.snappy.parquet. at org.apache.spark.sql.execution.datasources.FileScanRDD

Spark Hands On

1. Data Scanning in Big Data
2. Impact of Partitioning
3. Why its Important ??

```
In [0]: # Demo dataset from Databricks  
display(dbutils.fs.ls("/databricks-datasets/nyctaxi/tables/nyctaxi_yellow/"))
```

```
In [0]: display(dbutils.fs.ls("/data/input/nyctaxi/parquet/"))
```

path	name	size	modificationTime
dbfs:/data/input/nyctaxi/parquet/part-00000-7dcc09ea-2fc1-491d-a234-3b0ce1db9336-c002.snappy.parquet	part-00000-7dcc09ea-2fc1-491d-a234-3b0ce1db9336-c002.snappy.parquet	374549044	1720944538000
dbfs:/data/input/nyctaxi/parquet/part-00001-158e21e1-9d7b-44e9-ad15-3ba7e1797de8-c002.snappy.parquet	part-00001-158e21e1-9d7b-44e9-ad15-3ba7e1797de8-c002.snappy.parquet	364876497	1720944572000

```
In [0]: # Check the count and verify data scanning  
# data copied at location - "/data/input/nyctaxi/parquet/"  
df_parquet = spark.read.parquet("/data/input/nyctaxi/parquet/")  
display(df_parquet)
```

```
In [0]: # Check the count and verify data scanning
# data copied at location - "/data/input/nyctaxi/parquet/"
df_parquet = spark.read.parquet("/data/input/nyctaxi/parquet/")
display(df_parquet)
```

vendor_id	pickup_datetime	dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	rate_code_id	store_and_fwd旗
CMT	2010-02-12T08:13:37.000+0000	2010-02-12T08:23:22.000+0000	1	1.0	-73.993828	40.732739	1	
CMT	2010-02-12T20:48:01.000+0000	2010-02-12T21:01:40.000+0000	1	2.3	-74.007883	40.716298	1	
CMT	2010-02-12T16:25:23.000+0000	2010-02-12T16:47:31.000+0000	2	4.2	-74.001943	40.706967	1	
CMT	2010-02-13T02:02:16.000+0000	2010-02-13T02:06:49.000+0000	1	0.7	-74.003708	40.729451	1	
CMT	2010-02-12T20:49:29.000+0000	2010-02-12T21:04:18.000+0000	1	5.8	-73.993709	40.681982	1	
CMT	2010-02-13T07:51:56.000+0000	2010-02-13T07:56:42.000+0000	1	1.8	-74.000636	40.727351	1	
CMT	2010-02-12T11:12:37.000+0000	2010-02-12T11:22:08.000+0000	1	1.8	-74.004539	40.707313	1	
CMT	2010-02-13T06:59:54.000+0000	2010-02-13T07:03:14.000+0000	1	1.1	-74.012405	40.709634	1	
CMT	2010-02-12T04:20:26.000+0000	2010-02-12T04:24:21.000+0000	1	1.2	-74.005623	40.726305	1	

```
In [0]: # Check filter data
# select count(1) from nyctaxi where vendor_id = 'VTS' and trip_distance > 1.8
df_parquet.where("vendor_id = 'VTS' and trip_distance > 1.8").count()
```

Out[8]: 5466328

```
In [0]: # Write the data in partitioned format
df_parquet.write.format("parquet").mode("overwrite").partitionBy("vendor_id").option("path", "/data/input/nyctaxi/partitioned")
```

```
In [0]: display(dbutils.fs.ls("/data/input/nyctaxi/partitioned/"))
```

path	name	size	modificationTime
dbfs:/data/input/nyctaxi/partitioned/_SUCCESS	_SUCCESS	0	1720952782000
dbfs:/data/input/nyctaxi/partitioned/_started_1872870739494801713	_started_1872870739494801713	0	1720952067000
dbfs:/data/input/nyctaxi/partitioned/vendor_id=CMT/	vendor_id=CMT/	0	0
dbfs:/data/input/nyctaxi/partitioned/vendor_id=DDS/	vendor_id=DDS/	0	0
dbfs:/data/input/nyctaxi/partitioned/vendor_id=VTS/	vendor_id=VTS/	0	0

```
In [0]: df_partitioned = spark.read.parquet("/data/input/nyctaxi/partitioned/")
df_partitioned.where("vendor_id = 'VTS' and trip_distance > 1.8").count()
```

Out[14]: 5466328

```
In [0]: %sql
select count(1) from nyctaxi_partitioned where vendor_id = 'VTS' and trip_distance > 1.8
```

count(1)

5466328

Partitioning breaks data in folders:

Folder names are in

format:<partitioning><column>=<key>e.g.country=IN

Note: Partitioning column should be part of query in-order to improve performance

Impact of High Cardinality Column:

Always avoid partitioning on High Cardinality or Unique value columns to avoid creating too many partitions.

Spark Hands On

1. Importance of Data Skipping
2. Optimization in Delta Lake
3. Importance of Z-Ordering

```
In [0]: # Dataset
display(dbutils.fs.ls("/databricks-datasets/definitive-guide/data/retail-data/all/"))

In [0]: %fs head dbfs:/data/input/sales/sales.csv

[Truncated to first 65536 bytes] InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country
536365,85123A,WHITE HANGING HEART T-LIGHT HOLDER,6,12/1/2010 8:26,2.55,17850,United Kingdom
536365,71053,WHITE METAL LANTERN,6,12/1/2010 8:26,3.39,17850,United Kingdom
536365,84406B,CREAM CUPID HEARTS COAT HANGER,8,12/1/2010 8:26,2.75,17850,United Kingdom
536365,84029G,KNITTED UNION FLAG HOT WATER BOTTLE,6,12/1/2010 8:26,3.39,17850,United Kingdom
536365,84029E,RED WOOLLY HOTTIE WHITE HEART,,6,12/1/2010 8:26,3.39,17850,United Kingdom
536365,22752,SET 7 BABUSHKA NESTING BOXES,2,12/1/2010 8:26,7.65,17850,United Kingdom
536365,21730,GLASS STAR FROSTED T-LIGHT HOLDER,6,12/1/2010 8:26,4.25,17850,United Kingdom
536366,22633,HAND WARMER UNION JACK,6,12/1/2010 8:28,1.85,17850,United Kingdom
536366,22632,HAND WARMER RED POLKA DOT,6,12/1/2010 8:28,1.85,17850,United Kingdom
536367,84879,ASSORTED COLOUR BIRD ORNAMENT,32,12/1/2010 8:34,1.69,13047,United Kingdom
536367,22745,POPPY'S PLAYHOUSE BEDROOM,,6,12/1/2010 8:34,2.1,13047,United Kingdom
536367,22748,POPPY'S PLAYHOUSE KITCHEN,6,12/1/2010 8:34,2.1,13047,United Kingdom
536367,22749,FELTCRAFT PRINCESS CHARLOTTE DOLL,8,12/1/2010 8:34,3.75,13047,United Kingdom
536367,22310,IVORY KNITTED MUG COSY,,6,12/1/2010 8:34,1.65,13047,United Kingdom
536367,84969,BOX OF VINTAGE JIGSAW BLOCKS,3,12/1/2010 8:34,4.95,13047,United Kingdom
536367,22622,BOX OF VINTAGE ALPHABET BLOCKS,2,12/1/2010 8:34,9.95,13047,United Kingdom
536367,21754,HOME BUILDING BLOCK WORD,3,12/1/2010 8:34,5.95,13047,United Kingdom
536367,21777,LOVE BUILDING BLOCK WORD,3,12/1/2010 8:34,5.95,13047,United Kingdom
536367,48187,DOORMAT NEW ENGLAND,4,12/1/2010 8:34,7.95,13047,United Kingdom
536368,22960,JAM MAKING SET WITH

In [0]: # Write the data in form of delta table
df = spark.read.csv(path="/data/input/sales/sales.csv", inferSchema=True, header=True)
df.repartition(16).write.format("delta").mode("overwrite").partitionBy("country").option("path", "/data/output/sales_delta_")

In [0]: %sql
select * from sales_delta

org.apache.spark.sql.catalyst.ExtendedAnalysisException: [TABLE_OR_VIEW_NOT_FOUND] The table or view `sales_delta_partitioned` cannot be found. Verify the spelling and correctness of the schema and catalog.
If you did not qualify the name with a schema, verify the current_schema() output, or qualify the name with the correct schema and catalog.
To tolerate the error on drop use DROP VIEW IF EXISTS or DROP TABLE IF EXISTS. SQLSTATE: 42P01; line 1 pos 14;
'Project [*]
+- 'UnresolvedRelation [sales_delta_partitioned], [], false

at org.apache.spark.sql.catalyst.analysis.package$AnalysisErrorAt.tableNotFound(package.scala:90)
at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis$0$2(CheckAnalysis.scala:259)
at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis$0$2$adapted(CheckAnalysis.scala:232)

In [0]: # Data at delta location
display(dbutils.fs.ls("/data/output/sales_delta/"))


```

path	name	size	modificationTime
dbfs:/data/output/sales_delta/_delta_log/	_delta_log/	0	0
dbfs:/data/output/sales_delta/part-00000-4c2446ef-409c-4392-8ad4-75a228f0627b-c000.snappy.parquet	part-00000-4c2446ef-409c-4392-8ad4-75a228f0627b-c000.snappy.parquet	556272	1721394879000
dbfs:/data/output/sales_delta/part-00001-1c209617-a48d-4c08-824c-4b6dbfc77c00-c000.snappy.parquet	part-00001-1c209617-a48d-4c08-824c-4b6dbfc77c00-c000.snappy.parquet	555198	1721394879000
dbfs:/data/output/sales_delta/part-00002-d49dc9e9-e26a-4e05-9b6c-42547c7a0616-c000.snappy.parquet	part-00002-d49dc9e9-e26a-4e05-9b6c-42547c7a0616-c000.snappy.parquet	555484	1721394880000
dbfs:/data/output/sales_delta/part-00003-255380dc-2bfd-452c-a507-3ed2021ce192-c000.snappy.parquet	part-00003-255380dc-2bfd-452c-a507-3ed2021ce192-c000.snappy.parquet	555210	1721394880000
dbfs:/data/output/sales_delta/part-00004-36d34035-61c9-4f4d-a1ec-035daf0f1651-c000.snappy.parquet	part-00004-36d34035-61c9-4f4d-a1ec-035daf0f1651-c000.snappy.parquet	555365	1721394880000
dbfs:/data/output/sales_delta/part-00005-5305094e-320b-4e4b-a4f7-b8c319f5e9c9-c000.snappy.parquet	part-00005-5305094e-320b-4e4b-a4f7-b8c319f5e9c9-c000.snappy.parquet	556829	1721394880000
dbfs:/data/output/sales_delta/part-00006-4fc7d66f-98fb-42c2-8a6e-cff0a9253ca2-c000.snappy.parquet	part-00006-4fc7d66f-98fb-42c2-8a6e-cff0a9253ca2-c000.snappy.parquet	554588	1721394880000
dbfs:/data/output/sales_delta/part-00007-384ded64-d5ff-42d8-89d3-ba5eabc87150-c000.snappy.parquet	part-00007-384ded64-d5ff-42d8-89d3-ba5eabc87150-c000.snappy.parquet	554633	1721394880000
dbfs:/data/output/sales_delta/part-00008-eff136d6-6a2d-4bc7-ac27-1753481853c0-c000.snappy.parquet	part-00008-eff136d6-6a2d-4bc7-ac27-1753481853c0-c000.snappy.parquet	554983	1721394882000

```
In [0]: display(dbutils.fs.ls("/data/output/sales_delta_partitioned/Country=Australia"))
```

	path	name	size	modificationTime
	dbfs:/data/output/sales_delta_partitioned/Country=Australia/part-00000-52f07fc-dfc4-43a5-895d-6031814ef8c9.c000.snappy.parquet	part-00000-52f07fc-dfc4-43a5-895d-6031814ef8c9.c000.snappy.parquet	25140	1721398404000
	dbfs:/data/output/sales_delta_partitioned/Country=Australia/part-00000-c60956fa-a5d7-4d67-9f4b-acebcf042b2f.c000.snappy.parquet	part-00000-c60956fa-a5d7-4d67-9f4b-acebcf042b2f.c000.snappy.parquet	5322	1721398182000
	dbfs:/data/output/sales_delta_partitioned/Country=Australia/part-00001-6656eab6-f5a1-4da4-be51-ccd4f6c5aeb6.c000.snappy.parquet	part-00001-6656eab6-f5a1-4da4-be51-ccd4f6c5aeb6.c000.snappy.parquet	5483	1721398182000
	dbfs:/data/output/sales_delta_partitioned/Country=Australia/part-00002-8b0cf503-8be7-4c5a-be56-381391834518.c000.snappy.parquet	part-00002-8b0cf503-8be7-4c5a-be56-381391834518.c000.snappy.parquet	5498	1721398182000

```
In [0]: %sql  
select * from sales_delta_partitioned where InvoiceNo = '576394' and country = 'Australia'
```

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
576394	22326	ROUND SNACK BOXES SET OF4 WOODLAND	96	11/15/2011 10:32	2.55	12415	Australia
576394	22384	LUNCH BAG PINK POLKADOT	100	11/15/2011 10:32	1.45	12415	Australia
576394	22131	FOOD CONTAINER SET 3 LOVE HEART	72	11/15/2011 10:32	1.65	12415	Australia
576394	22466	FAIRY TALE COTTAGE NIGHT LIGHT	96	11/15/2011 10:32	1.65	12415	Australia
576394	22631	CIRCUS PARADE LUNCH BOX	64	11/15/2011 10:32	1.65	12415	Australia
576394	23084	RABBIT NIGHT LIGHT	960	11/15/2011 10:32	1.79	12415	Australia
576394	20725	LUNCH BAG RED RETROSPOT	100	11/15/2011 10:32	1.45	12415	Australia
576394	23497	CLASSIC CHROME BICYCLE BELL	48	11/15/2011 10:32	1.25	12415	Australia
576394	21770	OPEN CLOSED METAL SIGN	36	11/15/2011 10:32	4.25	12415	Australia

```
In [0]: %sql  
select min(invoiceno), max(invoiceno), _metadata.file_name from sales_delta  
group by _metadata.file_name  
order by min(invoiceno)
```

min(invoiceno)	max(invoiceno)	file_name
536365	539259	part-00000-410bc7f1-2563-4a5c-ae69-a78e3e07de41-c000.snappy.parquet
539259	541784	part-00001-54d9343c-6d45-4472-850b-7cfdc712106d-c000.snappy.parquet
541784	544839	part-00002-ccf52c99-ecb0-464a-8722-969c8fb773c-c000.snappy.parquet
544839	547940	part-00003-7d645b59-d0af-437f-a568-5fb3e030eb6d-c000.snappy.parquet
547940	550921	part-00004-f91da6ea-945d-40b5-aff7-4938bfea50a0-c000.snappy.parquet
550921	554083	part-00005-b66f1513-bac3-46b2-b263-1156cf64ae5b-c000.snappy.parquet
554083	557636	part-00006-ddb91e45-fa35-4862-a7ca-a8efebd4257c-c000.snappy.parquet
557636	560688	part-00007-a8bcbae2-51bf-4977-afe4-dd865e8050f7-c000.snappy.parquet
560688	563744	part-00008-3f5d5eaa-61f2-4bbf-9bf4-394c7a61026b-c000.snappy.parquet

```
In [0]: spark.conf.set("spark.databricks.delta.optimize.maxFileSize", 64*1024*8)

In [0]: %sql
OPTIMIZE sales_delta_partitioned where country = 'Australia' ZORDER BY (InvoiceNo)
```

path	metrics
dbfs:/data/output/sales_delta_partitioned	List(1, 16, List(25140, 25140, 25140.0, 1, 25140), List(5322, 6174, 5606.8125, 16, 89709), 1, List(minCubeSize(107374182400), List(0, 0), List(16, 89709), 0, List(16, 89709), 1, null), 1, 16, 0, false, 0, 0, 1721398399355, 1721398405941, 8, 1, null, List(0, 0), 8, 8, 327, 0, null))

```
In [0]: %sql
select min(invoiceno), max(invoiceno), _metadata.file_name from sales_delta
group by _metadata.file_name
order by min(invoiceno)
```

min(invoiceno)	max(invoiceno)	file_name
536365	539259	part-00000-410bc7f1-2563-4a5c-ae69-a78e3e07de41-c000.snappy.parquet
539259	541784	part-00001-54d9343c-6d45-4472-850b-7cfcd712106d-c000.snappy.parquet
541784	544839	part-00002-ccf52c99-ecb0-464a-8722-969c8fdb773c-c000.snappy.parquet
544839	547940	part-00003-7d645b59-d0af-437f-a568-5fb3e030eb6d-c000.snappy.parquet

```
In [0]: %sql
select country, min(invoiceno), max(invoiceno), _metadata.file_name from sales_delta
group by country, _metadata.file_name
order by country, min(invoiceno)
```

country	min(invoiceno)	max(invoiceno)	file_name
Australia	536389	539419	part-00000-c1f0b791-02a7-46c8-8d78-629ddd1e6a19-c000.snappy.parquet
Australia	540267	543376	part-00001-a32bfba6-703e-4211-845d-07d7b99350b5-c000.snappy.parquet
Australia	543989	547659	part-00002-049fc3a8-40ad-4a69-a364-c32ea6e5e1f3-c000.snappy.parquet
Australia	548661	553546	part-00003-b980260c-2329-4040-b79e-85f01e7fe504-c000.snappy.parquet
Australia	554037	558537	part-00004-56e69511-f3be-486e-bb64-d77d63d7bec4-c000.snappy.parquet
Australia	559919	563179	part-00005-bea09919-82e2-4bfe-b514-d2620783b724-c000.snappy.parquet
Australia	563614	567085	part-00006-77e59bd3-cd5e-4d80-a29f-3879f96a586b-c000.snappy.parquet
Australia	568145	569723	part-00007-61c9bac0-9944-4094-837c-f9b221704e7b-c000.snappy.parquet
Australia	574014	574469	part-00008-52278b78-d510-4da1-b7df-b354e1bf00b7-c000.snappy.parquet
Australia	576394	578459	part-00009-e8ad7bb7-3f5d-4c73-aac1-e7ac0a085224-c000.snappy.parquet
Australia	C538723	C574344	part-00010-9f07be7f-058d-487d-91b0-749ed32e61aa-c000.snappy.parquet
Austria	539330	539330	part-00000-c1f0b791-02a7-46c8-8d78-629ddd1e6a19-c000.snappy.parquet

High Cardinality Column:

Columns with more unique values (less repetition)

z-order can be done in more than one column.

_metadata is a hidden column that can be used to get the source metadata.

Configuration:

This configuration is used to control the file compaction max size using OPTIMIZE command.

Selective Z-Ordering with Partition filters:

Only Optimizes and Z-orders the data for the specific partition.

The screenshot displays two pages from the Delta Lake documentation side-by-side.

Left Page: What are deletion vectors?

- Header:** DELTA LAKE 3.2.1
- Search Bar:** Search
- Breadcrumbs:** Documentation > Apache Spark connector > What are deletion vectors?
- GitHub Repo:** Delta Lake GitHub repo
- Section:** Delta Lake
- Table of Contents:**
 - Introduction
 - Apache Spark connector
 - Quickstart
 - Table batch reads and writes
 - Table streaming reads and writes
 - Table deletes, updates, and merges
 - Change data feed
 - Table utility commands
 - Constraints
 - How does Delta Lake manage feature compatibility?
 - Delta default column values
 - Delta column mapping
 - Use liquid clustering for Delta tables
- Note:** This feature is available in Delta Lake 2.3.0 and above. This feature is in experimental support mode with ...
- Description:** Deletion vectors are a storage optimization feature that can be enabled on Delta Lake tables. By default, when a single row in a data file is deleted, the entire Parquet file containing the record must be rewritten. With deletion vectors enabled for the table, some Delta operations use deletion vectors to mark existing rows as removed without rewriting the Parquet file. Subsequent reads on the table resolve current table state by applying the deletions noted by deletion vectors to the most recent table version.
- Support:** Support for deletion vectors was incrementally added with each Delta Lake version. The table below depicts the supported operations for each Delta Lake version.
- Table:** Operation | First available Delta Lake version | Enabled by default since Delta Lake version
SCAN | 2.3.0 | 2.3.0

The screenshot displays two pages from the Delta Lake documentation side-by-side.

Left Page: Use liquid clustering for Delta tables

- Header:** DELTA LAKE 3.2.1
- Search Bar:** Search
- Breadcrumbs:** Documentation > Apache Spark connector > Use liquid clustering for Delta tables
- GitHub Repo:** Delta Lake GitHub repo
- Section:** Delta Lake
- Table of Contents:**
 - Introduction
 - Apache Spark connector
 - Quickstart
 - Table batch reads and writes
 - Table streaming reads and writes
 - Table deletes, updates, and merges
 - Change data feed
 - Table utility commands
 - Constraints
 - How does Delta Lake manage feature compatibility?
 - Delta default column values
 - Delta column mapping
 - Use liquid clustering for Delta tables
 - What is liquid clustering used for?
- Note:** This feature is available in Delta Lake 3.1.0 and above. See Limitations.
- Description:** Liquid clustering improves the existing partitioning and ZORDER techniques by simplifying data layout decisions in order to optimize query performance. Liquid clustering provides flexibility to redefine clustering columns without rewriting existing data, allowing data layout to evolve alongside analytic needs over time.

What is liquid clustering used for?

The following are examples of scenarios that benefit from clustering:

- Tables often filtered by high cardinality columns.
- Tables with significant skew in data distribution.
- Tables that grow quickly and require maintenance and tuning effort.
- Tables with access patterns that change over time.
- Tables where a typical partition column could leave the table with too many or too few partitions.

Enable liquid clustering

You must enable liquid clustering when creating a table. Clustering is not compatible with partitioning or ZORDER. Once enabled, run `OPTIMIZE` jobs as normal to incrementally cluster data. See [How to trigger clustering](#).

