

PySpark All Query Topics

✓ 1. Reading and Writing Data

◆ Explanation:

Spark supports reading/writing data from **CSV, JSON, Parquet, Delta**, etc., using `spark.read` and `df.write`.

◆ Code Example:

```
# Read CSV
df_csv = spark.read.option("header",
True).csv("/path/data.csv")
```

```
# Read JSON
df_json =
spark.read.json("/path/data.json")
```

```
# Read Parquet
df_parquet =
```

```
spark.read.parquet("/path/data.parquet")
```

```
# Write Parquet  
df_csv.write.mode("overwrite").parquet("/o  
utput/parquet")
```

```
# Write CSV with header  
df_json.write.option("header",  
True).csv("/output/csv")
```

🔍 Why It Matters:

Efficient file handling is **core to ETL**. Use `.option()` for format control (like headers, delimiters, etc.).

✓ 2.Schema Handling

◆ Explanation:

Define schemas explicitly using `StructType` for **performance & stability** (avoids schema inference).

◆ Code Example:

```
from pyspark.sql.types import StructType,  
StructField, StringType, IntegerType  
  
schema = StructType([  
    StructField("id", IntegerType(),  
True),  
    StructField("name", StringType(),  
True)  
])  
df =  
spark.read.schema(schema).csv("/path/data.  
csv")
```

🔍 Why It Matters:

Faster loads

Avoids issues with **incorrect data types**

✓ 3.Filtering Rows (filter / where)

◆ Explanation:

Use `filter()` or `where()` to select rows matching a condition.

◆ Code Example:

```
df.filter(df["age"] > 25).show()  
df.where("salary > 50000").show()
```

🔍 Why It Matters:

Pushes filtering to the source (predicate pushdown), improving performance.

✓ 4.Selecting Columns (selectWithColumn)

◆

Explanation:

`select` picks columns.

`withColumn` adds or updates columns.

◆ **Code Example:**

```
df.select("name", "ageshow()  
").  
from pyspark.sql.functions import col  
df.withColumn("age_plus_5", col("age") +  
5).show()
```

🔍 **Why It Matters:**

Helps you **transform data** efficiently and prepare it for further processing.

✓ **5.Renaming & Dropping Columns**



Explanation:

`withColumnRenamed`: Rename columns
`drop`: Drop columns

◆ **Code Example:**

```
df = df.withColumnRenamed("dob",  
"date_of_birth")
```

```
df = df.drop("unwanted_column")
```

🔍 Why It Matters:

Maintains **clean schema** especially when joining or preparing final output.

✓ 6. Aggregations (groupBy, agg)

◆ Explanation:

Use `groupBy` with aggregation functions like `count`, `sum`, `avg`, etc.

◆ Code Example:

```
from pyspark.sql.functions import count,  
sum, avg  
  
df.groupBy("department").agg(  
    count("*").alias("total"),  
    sum("salary").alias("total_salary"),  
    avg("salary").alias("avg_salary"))
```

```
).show()
```

🔍 Why It Matters:

Core of **reporting**, **dashboarding**, and **KPI generation**.

✓ 7. Joins

◆ Explanation:

PySpark supports inner, left, right, full, semi, anti joins.

◆ Code Example:

```
df1.join(df2, on="id", how="inner").show()  
df1.join(df2, on="id", how="left").show()
```

🔍 Why It Matters:

Used in **data merging**, **relational ETL**, and **lookups**.

8.Window Functions

◆ Explanation:

Used for **row-level operations** like rank, lead, lag, row_number **without collapsing rows**.

◆ Code Example:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import
row_number
windowSpec =
Window.partitionBy("department").orderBy(
"salary")
df.withColumn("rank",
row_number().over(windowSpec)).show()
```

Why It Matters:

Crucial for **top-N queries, lag analysis, sessionization**, etc.

9. Sorting Data

◆ Explanation:

Use `orderBy` or `sort`.

◆ Code Example:

```
df.orderBy("age").show()  
df.orderBy(df["age"].desc()).show()
```

10. Null Handling

◆ Explanation:

Handle nulls with `fillna`, `dropna` or conditionally replace using `when/otherwise`.

◆ Code Example:

```
df.fillna({"salary": 0}).show()  
df.dropna().show()  
  
from pyspark.sql.functions import when  
df.withColumn("salary",
```

```
when(df.salary.isNull(),  
0).otherwise(df.salary)).show()
```

✓ 11. User Defined Functions (UDFs)

◆ Explanation:

Use when you can't express logic using existing Spark functions.

◆ Code Example:

```
from pyspark.sql.functions import udf  
from pyspark.sql.types import StringType  
  
def upper_case(name):  
    return name.upper()  
  
upper_udf = udf(upper_case, StringType())  
  
df.withColumn("name_upper",  
upper_udf("name")).show()
```

Warning:

Slower than native functions – **avoid unless necessary.**

12. Broadcast Join

◆ Explanation:

Broadcast smaller DataFrames to improve performance of joins.

Code Example:

```
from pyspark.sql.functions import  
broadcast  
  
df1.join(broadcast(df2), "id").show()
```

Why It Matters:

Prevents **shuffle**, making join faster.

13. Caching & Persistence



Explanation:

`cache()`: Stores data in memory

`persist()`: Stores in memory/disk (configurable)

Code Example:

```
df.cache()  
df.count() # triggers caching
```

14. Repartitioning & Coalescing



Explanation:

`repartition(n)`: Increases partitions (shuffle involved)

`coalesce(n)`: Decreases partitions (no shuffle)

◆ **Code Example:**

```
df = df.repartition(10)  
df = df.coalesce(1)
```

✓ **15. Saving as Table / View**

◆ **Explanation:**

Create temporary/permanent views or tables from DataFrames.

◆ **Code Example:**

```
df.createOrReplaceTempView("emp_view")  
  
# Now you can use SQL  
spark.sql("SELECT * FROM emp_view WHERE  
salary > 50000").show()
```

Great! Let's continue with **Advanced PySpark Query Topics**, following the same format:

🔥 Advanced PySpark Query Topics



16. Pivot and Unpivot

◆ Explanation:

- pivot(): Converts **rows to columns**
- melt() (unpivoting): Not natively supported, but can be simulated using stack()

◆ Code Example:

```
# Pivot example: Average salary per
department and gender
df.groupBy("department").pivot("gender").a
vg("salary").show()
```

```
# Unpivot using stack()
df.selectExpr("id", "stack(2, 'math',
math_score, 'english', eng_score) as
(subject, score)").show()
```

🔍 Why It Matters:

Useful for **reporting**, **reshaping data**, and **machine learning feature transformation**.

✓ 17. Exploding Arrays and Maps

◆ Explanation:

Use `explode()` to **flatten** nested arrays or map fields.

◆ Code Example:

```
from pyspark.sql.functions import explode
```

```
# Sample DataFrame
df = spark.createDataFrame([
    (1, ["apple", "banana"]),
    (2, ["orange", "grapes"]))
```

```
], ["id", "fruits"])

df.select("id",
explode("fruits").alias("fruit")).show()
```

🔍 Why It Matters:

Used when working with **JSON**, **API data**, or **nested columns** from Kafka/NoSQL.

18. Handling Nested JSON

Explanation:

Used dot notation or `from_json()` to parse deeply nested structures.

◆ Code Example:

```
from pyspark.sql.functions import
from_json
from pyspark.sql.types import StructType,
StructField, StringType
```

```
schema = StructType([
    StructField("name", StringType(), True),
    StructField("info", StructType([
        StructField("city", StringType(), True),
        StructField("phone", StringType(), True)
    ])))
])

df = spark.read.json("/path/nested.json",
schema=schema)
df.select("name", "info.city",
"info.phone").show()
```

🔍 Why It Matters:

Important for **ingesting logs, Kafka streams, API payloads**, etc.

✓ 19. Working with Delta Lake

◆ Explanation:

Delta Lake adds **ACID transactions, schema evolution**, and **time travel** to Spark.

Code Example:

```
#Save as Delta
df.write.format("delta").mode("overwrite")
    .save("/delta/path")

# Read Delta
df =
spark.read.format("delta").load("/delta/pa
th")

# Time Travel (e.g., load old version)
df =
spark.read.format("delta").option("version
AsOf", 2).load("/delta/path")
```

Why It Matters:

Crucial for **data lakes**, **CDC pipelines**, **recovery**, and **data auditing**.

20. Performance Optimizations

◆ Explanation:

Speed up Spark jobs with config tuning, partitioning, broadcast joins, caching, and Catalyst-aware transformations.

◆ Key Techniques:

```
# Caching
df.cache()

# Avoid UDFs, use built-in functions
from pyspark.sql.functions import upper
df.withColumn("upper_name", upper("name"))

# Use broadcast joins wisely
from pyspark.sql.functions import broadcast
```

```
df1.join(broadcast(df2), "id")  
  
# Partition pruning  
df.write.partitionBy("state").parquet("/pa  
rtitioned")
```

🔍 Why It Matters:

Better performance = **lower cost, faster pipelines, and happy stakeholders**

✓ 21. Reading from JDBC

◆ Explanation:

Use Spark to read from relational databases (MySQL, PostgreSQL, SQL Server, etc.)

◆ Code Example:

```
jdbc_url =  
"jdbc:mysql://localhost:3306/mydb"  
props = {"user": "root", "password":  
"root123"}
```

```
df = spark.read.jdbc(url=jdbc_url,  
table="employees", properties=props)
```

✓ 22. Writing to Hive Tables

◆ Explanation:

Savedata into Hive-managed or external tables.

◆ Code Example:

```
spark.sql("CREATE DATABASE IF NOT EXISTS  
sales")
```

```
df.write.mode("overwrite").saveAsTable("sa  
les.emp_data")
```