Apache Spark 2 using SQL - Basic Transformations

1️⃣ Understanding Spark SQL Basic Transformations

Spark SQL transformations allow you to **modify, filter, and combine** data without changing the original dataset.
In SQL context, transformations can be:

- **Selection** (SELECT, .select())
- **Filtering** (WHERE, .filter() / .where())
- **Column aliasing** (AS)
- **Computed columns** (derived columns with expressions)
- **Sorting** (ORDER BY, .orderBy())
- **Aggregation** (GROUP BY)
- **Joins** (if needed at the basic level)
- **Limiting rows** (LIMIT)

💡 Spark SQL transformations are **lazy** — they don't execute until you perform an **action** like .show(), .count(), .collect().

2️⃣ Step-by-Step Scala Example

```
// 1. Import Spark SQL
import org.apache.spark.sql.{SparkSession, functions => F}
// 2. Create Spark Session
val spark = SparkSession.builder()
  .appName("SparkSQLBasicTransformations")
  .master("local[*]")
  .getOrCreate()
import spark.implicits._
// 3. Create sample DataFrame
val data = Seq(
  (1, "Alice", 25, "HR", 5000),
  (2, "Bob", 30, "IT", 7000),
  (3, "Charlie", 28, "IT", 6500),
  (4, "David", 35, "Finance", 8000)
)
val df = data.toDF("id", "name", "age", "dept", "salary")
// 4. Register as temporary view for SQL
df.createOrReplaceTempView("employees")
```

```
// 5. SQL: Basic Transformations
// a) Select specific columns
val selectedDF = spark.sql("SELECT name, dept, salary FROM employees")
selectedDF.show()
// b) Filter rows
val filteredDF = spark.sql("SELECT * FROM employees WHERE salary > 6500")
filteredDF.show()
// c) Add computed column
val bonusDF = spark.sql("SELECT name, salary, salary * 0.10 AS bonus FROM employees")
bonusDF.show()
// d) Sort data
val sortedDF = spark.sql("SELECT * FROM employees ORDER BY salary DESC")
sortedDF.show()
// e) Aggregation
val avgSalaryDept = spark.sql(
  """SELECT dept, AVG(salary) as avg_salary
     FROM employees
     GROUP BY dept"""
)
avgSalaryDept.show()
```

3 Explanation of Each Transformation

Transformation	SQL Equivalent	Purpose
.select() / SELECT	Pick specific columns	Reduce data to needed fields
.filter() / WHERE	Filter rows based on condition	Focus on relevant records
.withColumn() / Expressions	Add or modify columns	Derived metrics or new data
.orderBy() / ORDER BY	Sort results	Organize output
.groupBy() / GROUP BY	Aggregate data	Summarize

4 Practical Example

Business Case:
You have employee data and want:

1. Only IT employees
2. Show their name, salary, bonus (15% of salary)
3. Sorted by bonus (descending)

SQL Query in Spark:

```
val itBonusDF = spark.sql("""
   SELECT name, salary, salary * 0.15 AS bonus
   FROM employees
   WHERE dept = 'IT'
   ORDER BY bonus DESC
""")
itBonusDF.show()
```

5 Quizzes

Q1. Which Spark SQL command would you use to filter employees with salary > 7000?
a) FILTER salary > 7000

b) SELECT * FROM employees WHERE salary > 7000
c) WHERE salary > 7000 FROM employees
d) .show(salary > 7000)

Q2. In Spark SQL, what does .createOrReplaceTempView() do?
a) Creates a table in Hive permanently
b) Registers DataFrame as a temporary SQL view
c) Deletes an existing DataFrame
d) Saves DataFrame as a CSV

Q3. Which is a **transformation** and not an **action**?
a) .show()
b) .count()
c) .select()
d) .collect()

Q4. What is the default execution mode of Spark SQL transformations?
a) Immediate execution
b) Lazy evaluation
c) Parallel immediate execution
d) None of the above

Q5. Which Spark SQL function is used to sort data?
a) FILTER BY
b) SORT
c) ORDER BY
d) GROUP BY

6️⃣ GitHub Repositories to Explore

- Apache Spark SQL Examples in Scala – by spark-examples
- Spark SQL with Scala — Jaceklaskowski
- Spark SQL Basics — Dataflair
- Big Data Spark SQL Samples