SQL & PySpark Equivalence: A Comprehensive Guide

Structured Query Language (SQL) and PySpark are powerful tools for large-scale data processing. SQL is widely used for querying and managing structured data in relational databases, while PySpark, built on Apache Spark, excels in distributed computing and big data analytics.

This guide provides a side-by-side comparison of key SQL operations and their PySpark equivalents, covering data types, database operations, table alterations, partitioning, views, schema management, file operations, queries, aggregations, string and date functions, conditional logic, joins, grouping, set operations, window functions, and CTEs. It aims to help data professionals transition seamlessly between SQL and PySpark in hybrid environments.

1. Data Types

SQL Data Type	PySpark Equivalent
INT	IntegerType ()
BIGINT	LongType ()
FLOAT	FloatType ()
DOUBLE	DoubleType ()
CHAR(n) / VARCHAR(n)	StringType ()
DATE	DateType ()
TIMESTAMP	TimestampType ()

2. Database & Table operations

Concept	SQL Query	PySpark Equivalent
Create Database	CREATE DATABASE db_name;	spark.sql ("CREATE DATABASE db_name")
Use Database	USE db_name;	spark.catalog.setCurrentDatabase("db_name")
Drop Database	DROP DATABASE db_name;	spark.sql ("DROP DATABASE db_name")
Show Databases	SHOW DATABASES;	spark.sql ("SHOW DATABASES").show()
Create Table	CREATE TABLE table_name (col1 INT, col2 STRING);	df.write.format("parquet"). saveAsTable("table_name")
Drop Table	DROP TABLE table_name;	spark.sql ("DROP TABLE IF EXISTS table_name")
Truncate Table	TRUNCATE TABLE table_name;	spark.sql ("TRUNCATE TABLE table_name")
Describe Table	DESCRIBE TABLE table_name;	df.printSchema()
Show Tables	SHOW TABLES;	spark.sql ("SHOW TABLES").show()

3. Table Alterations

Concept	SQL Query	PySpark Equivalent
Add Column	ALTER TABLE table_name ADD COLUMN col3 STRING;	df.withColumn("col3", lit(None).cast("string"))
Rename Column	ALTER TABLE table_name RENAME COLUMN old_name TO new_name;	df.withColumnRenamed("old_name", "new_name")
Drop Column	ALTER TABLE table_name DROP COLUMN col3;	df.drop("col3")

4. Partitioning & Bucketing

Concept	SQL Query	PySpark Equivalent
Create Partitioned Table	CREATE TABLE table_name (col1 INT, col2 STRING) PARTITIONED BY (col3 STRING);	df.write.partitionBy("col3").format("parquet").saveAsTable("table_name")
Insert into Partitioned Table	INSERT INTO table_name PARTITION (col3='value') SELECT col1, col2 FROM source_table;	df.write.mode("append").partitionBy("col3").saveAsTable("table_name")
Create Bucketed Table	CREATE TABLE table_name (col1 INT, col2 STRING) CLUSTERED BY (col1) INTO 10 BUCKETS;	df.write.bucketBy(10, "col1").saveAsTable("table_name")

5. Views (Temporary & Permanent)

Concept	SQL Query	PySpark Equivalent
Create View	CREATE VIEW view_name AS SELECT * FROM table_name;	df.createOrReplaceTempView("view_name")
Drop View	DROP VIEW view_name;	spark.sql("DROP VIEW IF EXISTS view_name")
Create Global View	CREATE GLOBAL VIEW view_name AS SELECT * FROM table_name;	df.createGlobalTempView("view_name")
Show Views	SHOW VIEWS;	spark.sql("SHOW VIEWS").show()

6. Schema Management

Concept	SQL Query	PySpark Equivalent
Define Schema Manually	CREATE TABLE table_name (col1 INT, col2 STRING, col3 DATE);	from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DateTypeschema = StructType([StructField("col1", IntegerType(), True), StructField("col2", StringType(), True), StructField("col3", DateType(), True)])
Check Schema	DESCRIBE TABLE table_name;	df.printSchema()
Change Column Data Type	ALTER TABLE table_name ALTER COLUMN col1 TYPE BIGINT;	df.withColumn("col1", col("col1").cast("bigint"))

7. File-Based Table Operations

Concept	SQL Query	PySpark Equivalent
Save as Parquet	N/A (Implicit in Hive)	df.write.format("parquet").save("path/to/parquet")
Save as Delta Table	CREATE TABLE table_name USING DELTA LOCATION 'path';	df.write.format("delta").save("path/to/delta")
Save as CSV	N/A	df.write.format("csv").option("header", "true").save("path/to/csv")
Save as JSON	N/A	df.write.format("json").save("path/to/json")
Save as ORC	N/A	df.write.format("orc").save("path/to/orc")

8. Basic SELECT Queries

Concept	SQL Query	PySpark Equivalent
Select Specific Columns	SELECT column1, column2 FROM table;	df.select("column1", "column2")
Select All Columns	SELECT * FROM table;	df.select("*")
Distinct Values	SELECT DISTINCT column FROM table;	df.select("column").distinct()
WHERE Condition	SELECT * FROM table WHERE column = 'value';	df.filter(col("column") == 'value')
ORDER BY	SELECT * FROM table ORDER BY column;	df.sort("column")
LIMIT Rows	SELECT * FROM table LIMIT n;	df.limit(n)
COUNT Rows	SELECT COUNT(*) FROM table;	df.count()

9. Aggregate Functions

Concept	SQL Query	PySpark Equivalent
SUM	SELECT SUM(column) FROM table;	df.agg({"column": "sum"})
AVG	SELECT AVG(column) FROM table;	df.agg({"column": "avg"})
MAX	SELECT MAX(column) FROM table;	df.agg({"column": "max"})
MIN	SELECT MIN(column) FROM table;	df.agg({"column": "min"})

10. String Functions

Concept	SQL Query	PySpark Equivalent
String Length	SELECT LEN(column) FROM table;	df.select(length(col("column")))
Convert to Uppercase	SELECT UPPER(column) FROM table;	df.select(upper(col("column")))
Convert to Lowercase	SELECT LOWER(column) FROM table;	df.select(lower(col("column")))
Concatenate Strings	SELECT CONCAT(string1, string2) FROM table;	df.select(concat(col("string1"), col("string2")))
Trim String	SELECT TRIM(column) FROM table;	df.select(trim(col("column")))
Substring	SELECT SUBSTRING(column, start, length) FROM table;	df.select(substring(col("column"), start, length))

11. Date & Time Functions

Concept	SQL Query	PySpark Equivalent
Current Date	SELECT CURDATE();	df.select(current_date())
Current Timestamp	SELECT NOW();	df.select(current_timestamp())
CAST / CONVERT	SELECT CAST(column AS datatype) FROM table;	df.select(col("column").cast("datatype"))

12. Conditional Logic

Concept	SQL Query	PySpark Equivalent
IF (Conditional Logic)	SELECT IF(condition, value1, value2) FROM table;	df.select(when(condition, value1).otherwise(value2))
COALESCE	SELECT COALESCE(column1, column2, column3) FROM table;	df.select(coalesce(col("column1"), col("column2"), col("column3")))

13. Join, Grouping & Pivoting

Concept	SQL Query	PySpark Equivalent
JOIN	SELECT * FROM table1 JOIN table2 ON table1.column = table2.column;	df1.join(df2, "column")
GROUP BY	SELECT column, agg_function(column) FROM table GROUP BY column;	df.groupBy("column").agg({"column": "agg_function"})
PIVOT	PIVOT (agg_function(column) FOR pivot_column IN (values));	df.groupBy("pivot_column").pivot("column").agg({"column": "agg_function"})

14. Logical Operators

Concept	SQL Query	PySpark Equivalent
AND / OR	SELECT * FROM table WHERE column1 = value AND column2 > value;	df.filter((col("column1") == value) & (col("column2") > value))
IS NULL / IS NOT NULL	SELECT * FROM table WHERE column IS NULL;	df.filter(col("column").isNull())
LIKE	SELECT * FROM table WHERE column LIKE 'value%';	df.filter(col("column").like("value%"))
BETWEEN	SELECT * FROM table WHERE column BETWEEN value1 AND value2;	df.filter((col("column") >= value1) & (col("column") <= value2))

15. Set Operations

Concept	SQL Query	PySpark Equivalent
UNION	SELECT column FROM table1 UNION SELECT column FROM table2;	df1.union(df2).select("column")
UNION ALL	SELECT column FROM table1 UNION ALL SELECT column FROM table2;	df1.unionAll(df2).select("column")

16. Window Functions

Concept	SQL Query	PySpark Equivalent
RANK()	SELECT column, RANK() OVER (PARTITION BY col2 ORDER BY column) FROM table;	df.withColumn("rank", rank().over(Window.partitionBy("col2").orderBy("column")))
DENSE_RANK()	SELECT column, DENSE_RANK() OVER (PARTITION BY col2 ORDER BY column) FROM table;	df.withColumn("dense_rank", dense_rank().over(Window.partitionBy("col2").orderBy("column")))
ROW_NUMBER()	SELECT column, ROW_NUMBER() OVER (PARTITION BY col2 ORDER BY column) FROM table;	df.withColumn("row_number", row_number().over(Window.partitionBy("col2").orderBy("column")))
LEAD()	SELECT column, LEAD(column, 1) OVER (PARTITION BY col2 ORDER BY column) FROM table;	df.withColumn("lead_value", lead("column", 1).over(Window.partitionBy("col2").orderBy("column")))
LAG()	SELECT column, LAG(column, 1) OVER (PARTITION BY col2 ORDER BY column) FROM table;	df.withColumn("lag_value", lag("column", 1).over(Window.partitionBy("col2").orderBy("column")))

17. Common Table Expressions (CTEs)

Concept	SQL Query	PySpark Equivalent
CTE	WITH cte1 AS (SELECT * FROM table1) SELECT * FROM cte1 WHERE condition;	df.createOrReplaceTempView("cte1")df_cte1 = spark.sql("SELECT * FROM cte1 WHERE condition")