

Data Engineering 101 SQL and PySpark



Shwetank Singh
GritSetGrow - GSGLearn.com

SELECT ALL COLUMNS

SQL

```
SELECT * FROM table;
```

PYSPARK

```
df.select("*")
```



SELECT SPECIFIC COLUMNS

SQL

```
SELECT col1, col2 FROM table;
```

PYSPARK

```
df.select("col1", "col2")
```



FILTERING ROWS (WHERE CLAUSE)

SQL

```
SELECT * FROM table WHERE condition;
```

PYSPARK

```
df.filter("condition")
```



ORDERING ROWS

SQL

```
SELECT * FROM table ORDER BY col1;
```

PYSPARK

```
df.orderBy("col1")
```



ORDERING ROWS DESCENDING

SQL

```
SELECT * FROM table ORDER BY col1 DESC;
```

PYSPARK

```
df.orderBy(df.col1.desc())
```



LIMITING ROWS

SQL

```
SELECT * FROM table LIMIT 10;
```

PYSPARK

```
df.limit(10)
```



SELECTING DISTINCT VALUES

SQL

```
SELECT DISTINCT col1 FROM table;
```

PYSPARK

```
df.select("col1").distinct()
```



GROUP BY AND AGGREGATE (COUNT) SQL

```
SELECT col1, COUNT(*) FROM table GROUP  
BY col1;
```

PYSPARK

```
df.groupBy("col1").count()
```



GROUP BY AND AGGREGATE (AVG)

SQL

```
SELECT col1, AVG(col2) FROM table GROUP  
BY col1;
```

PYSPARK

```
df.groupBy("col1").avg("col2")
```



INNER JOIN

SQL

```
SELECT *  
FROM table1  
INNER JOIN table2  
ON table1.id = table2.id;
```

PYSPARK

```
df1.join(df2, df1.id == df2.id, "inner")
```



LEFT JOIN

SQL

```
SELECT * FROM table1  
LEFT JOIN table2 ON table1.id = table2.id;
```

PYSPARK

```
df1.join(df2, df1.id == df2.id, "left")
```



RIGHT JOIN

SQL

```
SELECT * FROM table1  
RIGHT JOIN table2  
ON table1.id = table2.id;
```

PYSPARK

```
df1.join(df2, df1.id == df2.id, "right")
```



FULL OUTER JOIN

SQL

```
SELECT * FROM table1  
FULL JOIN table2  
ON table1.id = table2.id;
```

PYSPARK

```
df1.join(df2, df1.id == df2.id, "outer")
```



SUBQUERIES

SQL

```
SELECT * FROM (SELECT col1, col2 FROM  
table) sub_table;
```

PYSPARK

```
sub_df = df.select("col1", "col2")
```



CASE STATEMENTS

SQL

```
SELECT col1,  
CASE WHEN condition  
THEN result ELSE result2 END  
FROM table;
```

PYSPARK

```
df.select("col1", when(condition,  
result).otherwise(result2).alias("new_col"))
```



WINDOW FUNCTIONS (ROW NUMBER)

SQL

```
SELECT col1,  
ROW_NUMBER() OVER (ORDER BY col2) AS row_num  
FROM table;
```

PYSPARK

```
df.withColumn("row_num",  
row_number().over(Window.orderBy("col2")))
```



WINDOW FUNCTIONS (AGGREGATIONS)

SQL

```
SELECT col1,  
SUM(col2) OVER (PARTITION BY col3) AS sum_col2  
FROM table;
```

PYSPARK

```
df.withColumn("sum_col2", sum("col2") \  
    .over(Window.partitionBy("col3")))
```



LAG FUNCTION

SQL

```
SELECT col1,  
LAG(col2, 1) OVER (ORDER BY col3)  
AS lag_col2 FROM table;
```

PYSPARK

```
df.withColumn("lag_col2", lag("col2", 1) \  
    .over(Window.orderBy("col3")))
```



LEAD FUNCTION

SQL

```
SELECT col1,  
LEAD(col2, 1) OVER (ORDER BY col3) AS lead_col2  
FROM table;
```

PYSPARK

```
df.withColumn("lead_col2", lead("col2", 1) \  
    .over(Window.orderBy("col3")))
```



HANDLING NULLS (IS NULL)

SQL

```
SELECT * FROM table WHERE col1 IS NULL;
```

PYSPARK

```
df.filter(df.col1.isNull())
```



HANDLING NULLS (IS NOT NULL)

SQL

```
SELECT * FROM table WHERE col1  
IS NOT NULL;
```

PYSPARK

```
df.filter(df.col1.isNotNull())
```



UNION OF TWO TABLES

SQL

```
SELECT * FROM table1  
UNION  
SELECT * FROM table2;
```

PYSPARK

```
df1.union(df2)
```



INTERSECT OF TWO TABLES

SQL

```
SELECT * FROM table1  
INTERSECT  
SELECT * FROM table2;
```

PYSPARK

```
df1.intersect(df2)
```



EXCEPT (DIFFERENCE) OF TWO TABLES

SQL

```
SELECT * FROM table1  
EXCEPT  
SELECT * FROM table2;
```

PYSPARK

```
df1.subtract(df2)
```



CREATING TEMPORARY VIEW

SQL

```
CREATE TEMP VIEW temp_table  
AS SELECT * FROM table;
```

PYSPARK

```
df.createOrReplaceTempView("temp_table")
```



USING SQL QUERIES ON DATAFRAMES

SQL

```
SELECT * FROM temp_table;
```

PYSPARK

```
spark.sql("SELECT * FROM temp_table")
```



PIVOTING DATA

SQL

```
SELECT * FROM  
(SELECT col1, col2 FROM table)  
PIVOT (SUM(col2) FOR col1 IN ('value1', 'value2'));
```

PYSPARK

```
df.groupBy()\n.pivot("col1", ['value1', 'value2']).sum("col2")
```



UPDATING ROWS

SQL

```
UPDATE table SET col1 = value WHERE  
condition;
```

PYSPARK

```
df = df.withColumn("col1", when(condition,  
value).otherwise(df.col1))
```



DELETING ROWS

SQL

```
DELETE FROM table WHERE condition;
```

PYSPARK

```
df = df.filter(~condition)
```



HANDLING DUPLICATES

SQL

```
SELECT col1, COUNT(*) FROM table  
GROUP BY col1 HAVING COUNT(*) > 1;
```

PYSPARK

```
df.groupBy("col1").count().filter("count > 1")
```



CALCULATING PERCENTAGE

SQL

```
SELECT col1, (col2 / col3) * 100 AS percentage  
FROM table;
```

PYSPARK

```
df.withColumn("percentage", (df.col2 /  
df.col3) * 100)
```



STRING FUNCTIONS (CONCATENATION)

SQL

```
SELECT CONCAT(col1, col2) AS new_col  
FROM table;
```

PYSPARK

```
df.select(concat("col1", "col2") \\  
          .alias("new_col"))
```



DATE FUNCTIONS (CURRENT DATE)

SQL

```
SELECT CURRENT_DATE AS today;
```

PYSPARK

```
df.select(current_date().alias("today"))
```



EXTRACTING YEAR FROM DATE

SQL

```
SELECT EXTRACT(YEAR FROM date_col) AS year  
FROM table;
```

PYSPARK

```
df.select(year("date_col").alias("year"))
```



CONDITIONAL AGGREGATION

SQL

```
SELECT SUM(CASE WHEN condition THEN col1 ELSE 0 END)  
FROM table;
```

PYSPARK

```
df.select(sum(when(condition, df.col1) \\\n              .otherwise(0)))
```



RENAMING COLUMNS

SQL

```
SELECT col1 AS new_col1 FROM table;
```

PYSPARK

```
df.withColumnRenamed("col1", "new_col1")
```



DROPPING COLUMNS

SQL

```
ALTER TABLE table DROP COLUMN col1;
```

PYSPARK

```
df.drop("col1")
```



ADDING NEW COLUMNS

SQL

```
ALTER TABLE table  
ADD COLUMN new_col data_type;
```

PYSPARK

```
df.withColumn("new_col", expression)
```



REPLACING VALUES

SQL

```
UPDATE table SET col1 = new_value  
WHERE condition;
```

PYSPARK

```
df.withColumn("col1", when(condition,  
new_value).otherwise(df.col1))
```



USING UDFS (USER-DEFINED FUNCTIONS)

SQL

```
CREATE FUNCTION my_udf AS ...;  
SELECT my_udf(col1) FROM table;
```

PYSPARK

```
df.select(my_udf("col1"))
```



EXPLODING ARRAYS

SQL

```
SELECT col1, EXPLODE(array_col) AS exploded_col  
FROM table;
```

PYSPARK

```
df.select("col1", explode("array_col") \\  
        .alias("exploded_col"))
```



FLATTENING NESTED DATA

SQL

```
SELECT col1, nested_col.* FROM table;
```

PYSPARK

```
df.select("col1", "nested_col.*")
```



CASTING DATA TYPES

SQL

```
SELECT CAST(col1 AS data_type)  
FROM table;
```

PYSPARK

```
df.select(df.col1.cast("data_type"))
```



HANDLING JSON DATA

SQL

```
SELECT JSON_VALUE(json_col, '$.key')  
FROM table;
```

PYSPARK

```
df.select(get_json_object("json_col", "$.key"))
```



GROUPING SETS

SQL

```
SELECT col1, col2, SUM(col3)  
FROM table  
GROUP BY GROUPING SETS ((col1), (col2));
```

PYSPARK

```
df.groupBy("col1",  
"col2").agg(sum("col3"))
```



ROLLUP

SQL

```
SELECT col1, col2, SUM(col3)  
FROM table  
GROUP BY ROLLUP(col1, col2);
```

PYSPARK

```
df.rollup("col1", "col2") \  
    .agg(sum("col3"))
```



CUBE

SQL

```
SELECT col1, col2, SUM(col3)  
FROM table GROUP BY CUBE(col1, col2);
```

PYSPARK

```
df.cube("col1", "col2") \  
    .agg(sum("col3"))
```



RANK FUNCTION

SQL

```
SELECT col1,  
RANK() OVER (ORDER BY col2) AS rank  
FROM table;
```

PYSPARK

```
df.withColumn("rank", rank() \  
.over(Window.orderBy("col2")))
```



DENSE RANK FUNCTION

SQL

```
SELECT col1,  
DENSE_RANK() OVER (ORDER BY col2) AS dense_rank  
FROM table;
```

PYSPARK

```
df.withColumn("dense_rank",  
dense_rank().over(Window.orderBy("col2")))
```



CUMULATIVE SUM (RUNNING TOTAL)

SQL

```
SELECT col1,  
SUM(col2) OVER (ORDER BY col1) AS running_total  
FROM table;
```

PYSPARK

```
df.withColumn("running_total", sum("col2")\  
    .over(Window.orderBy("col1") \  
    .rowsBetween(Window.unboundedPreceding,  
        Window.currentRow)))
```



HANDLING DATES (DATE DIFFERENCE)

SQL

```
SELECT DATEDIFF(date1, date2) FROM table;
```

PYSPARK

```
df.select(datediff("date1", "date2"))
```



STRING FUNCTIONS (SUBSTRING)

SQL

```
SELECT SUBSTRING(col1, start, length)  
FROM table;
```

PYSPARK

```
df.select(substring("col1", start, length))
```



UPPER AND LOWER CASE CONVERSION

SQL

```
SELECT UPPER(col1), LOWER(col2)  
FROM table;
```

PYSPARK

```
df.select(upper("col1"), lower("col2"))
```



FILTER WITH IN CLAUSE

SQL

```
SELECT * FROM table  
WHERE col1 IN (value1, value2);
```

PYSPARK

```
df.filter(df.col1.isin(value1, value2))
```



FILTER WITH BETWEEN CLAUSE

SQL

```
SELECT * FROM table  
WHERE col1 BETWEEN value1 AND value2;
```

PYSPARK

```
df.filter(df.col1.between(value1, value2))
```



ORDER BY MULTIPLE COLUMNS

SQL

```
SELECT * FROM table  
ORDER BY col1, col2 DESC;
```

PYSPARK

```
df.orderBy("col1", df.col2.desc())
```



THANK
you