SQL vs PySpark vs Pandas

System Characteristics

Performance

- **PySpark**: Optimized for distributed processing
- **Pandas**: Limited by memory
- **SQL**: Dependent on database capabilities

Execution Mode

- **PySpark**: Lazy Evaluation
- **Pandas**: Eager Evaluation
- **SQL**: Eager Evaluation

Parallel Processing

- **PySpark**: Yes, across multiple nodes
- **Pandas**: No, single machine
- **SQL**: Typically single machine unless parallel query execution is supported

Handling Big Data

- **PySpark**: Efficiently handles large datasets
- **Pandas**: Not designed for big data
- **SQL**: Limited by database capabilities

File Format Support

- **PySpark**: CSV, JSON, Parquet, ORC, Avro, etc.
- **Pandas**: CSV, Excel, JSON, HDF5, etc.
- **SQL**: Varies; often requires external tools for non-tabular formats

Basic Data Operations

Data Loading

- **SQL**: SELECT * FROM table_name

- **PySpark**: df = spark.read.csv("file.csv")
- **Pandas**: df = pd.read_csv("file.csv")

Selecting Columns

- **SQL**: SELECT column1, column2 FROM table;
- **PySpark**: df.select("column1", "column2").show()
- **Pandas**: df[['column1', 'column2']]

Filtering Rows

- **SQL**: SELECT * FROM table WHERE column > 50
- **PySpark**: df.filter(df['column'] > 50) or df.filter(df.column > 50).show()
- **Pandas**: df[df['column'] > 50]

Multiple Conditions (AND, OR, NOT)

- **SQL**: SELECT * FROM table WHERE column1 > 10 AND column2 < 20;
- **PySpark**: df.filter((df.column1 > 10) & (df.column2 < 20)).show()
- **Pandas**: df[(df['column1'] > 10) & (df['column2'] < 20)]

IN Operator

- **SQL**: SELECT * FROM table WHERE column IN (val1, val2);
- **PySpark**: df.filter(df.column.isin(val1, val2)).show()
- **Pandas**: df[df['column'].isin([val1, val2])]

LIKE Operator

- **SQL**: SELECT * FROM table WHERE column LIKE 'pattern%';
- **PySpark**: df.filter(df.column.like('pattern%')).show()
- **Pandas**: df[df['column'].str.contains('pattern')]

Handling NULL Values

- **SQL**: SELECT * FROM table WHERE column IS NULL;
- **PySpark**: df.filter(df.column.isNull()).show()
- **Pandas**: df[df['column'].isnull()]
- **SQL**: UPDATE table_name SET column = value WHERE column IS NULL
- **PySpark**: df.fillna(value)

- **Pandas**: df.fillna(value)

Data Manipulation

Adding a New Column

- **SQL**: ALTER TABLE table_name ADD COLUMN new_col INT;
- **PySpark**: df = df.withColumn("new_col", df["column"] + 10)
- **Pandas**: df['new_col'] = df['column'] + 10

Removing a Column

- **SQL**: ALTER TABLE table_name DROP COLUMN column;
- **PySpark**: df = df.drop("column")
- **Pandas**: df = df.drop('column', axis=1)

Sorting Data

- **SQL**: SELECT * FROM table ORDER BY column ASC;
- **PySpark**: df.orderBy(df.column.asc()).show() or df.orderBy('column', ascending=True).show()
- **Pandas**: df.sort_values("column", ascending=True)
- **SQL**: SELECT * FROM table ORDER BY column DESC;
- **PySpark**: df.orderBy(df.column.desc()).show() or df.orderBy('column', ascending=False).show()
- **Pandas**: df.sort_values("column", ascending=False)

Limiting Results

- **SQL**: SELECT * FROM table LIMIT 10;
- **PySpark**: df.limit(10).show()
- **Pandas**: df.head(10)

Distinct Values

- **SQL**: SELECT DISTINCT column FROM table;
- **PySpark**: df.select('column').distinct().show()
- **Pandas**: df['column'].unique()

Data Modification

Insert

- **SQL**: INSERT INTO table VALUES (...);
- **PySpark**: df.write.insertInto('table')
- **Pandas**: df.append(...)

Update

- **SQL**: UPDATE table SET column=value WHERE condition;
- **PySpark**: df.withColumn('column', new_value)
- **Pandas**: df.loc[df.condition, 'column'] = value

Delete

- **SQL**: DELETE FROM table WHERE condition;
- **PySpark**: df.filter(~condition).show()
- **Pandas**: df = df[~df['column'].condition]

Aggregations

Count

- **SQL**: SELECT COUNT(*) FROM table;
- **PySpark**: df.count()
- **Pandas**: df.shape[0]

Sum

- **SQL**: SELECT SUM(column) FROM table;
- **PySpark**: df.selectExpr('SUM(column)').show()
- **Pandas**: df['column'].sum()

Average

- **SQL**: SELECT AVG(column) FROM table;
- **PySpark**: df.selectExpr('AVG(column)').show()
- **Pandas**: df['column'].mean()

Minimum

- **SQL**: SELECT MIN(column) FROM table;
- **PySpark**: df.selectExpr('MIN(column)').show()
- **Pandas**: df['column'].min()

Maximum

- **SQL**: SELECT MAX(column) FROM table;
- **PySpark**: df.selectExpr('MAX(column)').show()
- **Pandas**: df['column'].max()

Group By

- **SQL**: SELECT column, COUNT(*) FROM table GROUP BY column;
- **PySpark**: df.groupBy('column').count().show()
- **Pandas**: df.groupby('column').size()
- **SQL**: SELECT column, SUM(column) FROM table_name GROUP BY column
- **PySpark**: df.groupBy("column").agg({"column": "sum"})
- **Pandas**: df.groupby('column').sum()

Complex Aggregations

- **SQL**: SELECT column, SUM(col1), MAX(col2) FROM table GROUP BY column;
- **PySpark**: df.groupBy("column").agg({"col1": "sum", "col2": "max"})
- **Pandas**: df.groupby("column").agg({"col1": "sum", "col2": "max"})

Joins

Inner Join

- **SQL**: SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.id;
- **PySpark**: df1.join(df2, df1.id == df2.id, 'inner').show()
- **Pandas**: df1.merge(df2, on='id', how='inner')

Left Join

- **SQL**: SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id;
- **PySpark**: df1.join(df2, df1.id == df2.id, 'left').show()
- **Pandas**: df1.merge(df2, on='id', how='left')

Right Join

- **SQL**: SELECT * FROM table1 RIGHT JOIN table2 ON table1.id = table2.id;
- **PySpark**: df1.join(df2, df1.id == df2.id, 'right').show()
- **Pandas**: df1.merge(df2, on='id', how='right')

Full Outer Join

- **SQL**: SELECT * FROM table1 FULL OUTER JOIN table2 ON table1.id = table2.id;
- **PySpark**: df1.join(df2, df1.id == df2.id, 'outer').show()
- **Pandas**: df1.merge(df2, on='id', how='outer')

Window Functions

Row Number

- **SQL**: SELECT ROW_NUMBER() OVER (PARTITION BY col ORDER BY col2) FROM table;
- **PySpark**: df.withColumn('row_number', F.row_number().over(Window.partitionBy('col').orderBy('col2'))).show()
- **Pandas**: df['row_number'] = df.groupby('col').cumcount() + 1

Rank

- **SQL**: SELECT RANK() OVER (PARTITION BY col ORDER BY col2) FROM table;
- **PySpark**: df.withColumn('rank', F.rank().over(Window.partitionBy('col').orderBy('col2'))).show()
- **Pandas**: df['rank'] = df.groupby('col')['col2'].rank(method='dense', ascending=True)

Dense Rank

- **SQL**: SELECT DENSE_RANK() OVER (PARTITION BY col ORDER BY col2) FROM table;
- **PySpark**: df.withColumn('dense_rank', F.dense_rank().over(Window.partitionBy('col').orderBy('col2'))).show()
- **Pandas**: df['dense_rank'] = df.groupby('col')['col2'].rank(method='dense')

Sum Over

- **SQL**: SELECT SUM(column) OVER (PARTITION BY col) FROM table;
- **PySpark**: df.withColumn('sum_over', F.sum('column').over(Window.partitionBy('col'))).show()
- **Pandas**: df['sum_over'] = df.groupby('col')['column'].transform('sum')

Average Over

- **SQL**: SELECT AVG(column) OVER (PARTITION BY col) FROM table;

- **PySpark**: df.withColumn('avg_over', F.avg('column').over(Window.partitionBy('col'))).show()
- **Pandas**: df['avg_over'] = df.groupby('col')['column'].transform('mean')

String Functions

Upper Case

- **SQL**: SELECT UPPER(column) FROM table;
- **PySpark**: df.selectExpr('UPPER(column)').show()
- **Pandas**: df['column'].str.upper()

Lower Case

- **SQL**: SELECT LOWER(column) FROM table;
- **PySpark**: df.selectExpr('LOWER(column)').show()
- **Pandas**: df['column'].str.lower()

Trim

- **SQL**: SELECT TRIM(column) FROM table;
- **PySpark**: df.selectExpr('TRIM(column)').show()
- **Pandas**: df['column'].str.strip()

Replace

- **SQL**: SELECT REPLACE(column, 'old', 'new') FROM table;
- **PySpark**: df.selectExpr("REPLACE(column, 'old', 'new')").show()
- **Pandas**: df['column'].str.replace('old', 'new')

Date Functions

Current Date

- **SQL**: SELECT CURRENT_DATE;
- **PySpark**: df.selectExpr('CURRENT_DATE').show()
- **Pandas**: pd.to_datetime('today').date()

Add Days to Date

- **SQL**: SELECT DATE_ADD(column, INTERVAL 10 DAY) FROM table;
- **PySpark**: df.selectExpr('DATE_ADD(column, 10)').show()

- **Pandas**: df['column'] + pd.Timedelta(days=10)

Subtract Days from Date

- **SQL**: SELECT DATE_SUB(column, INTERVAL 10 DAY) FROM table;
- **PySpark**: df.selectExpr('DATE_SUB(column, 10)').show()
- **Pandas**: df['column'] - pd.Timedelta(days=10)

Date Difference

- **SQL**: SELECT DATEDIFF(end_date, start_date) FROM table;
- **PySpark**: df.selectExpr('DATEDIFF(end_date, start_date)').show()
- **Pandas**: df['end_date'] - df['start_date']

Pivoting/Unpivoting Data

Pivot

- **SQL**: PIVOT (SUM(value) FOR column_to_pivot IN (...)) (DBMS dependent)
- **PySpark**: df.groupBy("column").pivot("column_to_pivot").sum("value")
- **Pandas**: df.pivot_table(values='value', index='column', columns='column_to_pivot', aggfunc='sum')

Other Features

Creating Temporary Views

- **SQL**: CREATE OR REPLACE TEMP VIEW view_name AS ...
- **PySpark**: df.createOrReplaceTempView("view_name")
- **Pandas**: Not Applicable

Running SQL Queries

- **SQL**: SELECT * FROM view_name WHERE condition;
- **PySpark**: spark.sql("SELECT * FROM view_name WHERE ...").show()
- **Pandas**: Not Applicable