SQL & PYSPARK EQUIVALENT

DML OPERATIONS

Concept	SQL	PySpark

SELECT	SELECT column(s) FROM table
	SELECT * FROM table
		df.select("column(s)")
		df.select("*")

DISTINCT	SELECT DISTINCT column(s) FROM table	df.select("column(s)").distinct()
WHERE	SELECT column(s) FROM table WHERE condition	df.filter(condition)\ .select("column(s)")
ORDER BY	SELECT column(s) FROM table ORDER BY column(s)	df.sort("column(s)")\ .select("column(s)")
LIMIT	SELECT column(s) FROM table LIMIT n	df.limit(n).select("column(s)")
COUNT	SELECT COUNT(*) FROM table	df.count()

Concept	SQL	PySpark

SUM	SELECT SUM(column) FROM table	from pyspark.sql.functions import sum;
		df.agg(sum("column"))

AVG	SELECT AVG(column) FROM table	from pyspark.sql.functions import avg;
		df.agg(avg("column"))

MAX / MIN	SELECT MAX(column)
	FROM table
		from pyspark.sql.functions import max;
		df.agg(max("column"))

String Length	SELECT LEN(string) FROM table	from pyspark.sql.functions import length;
		df.select(length(col("string")))

Convert to Uppercase	SELECT UPPER(string)
	FROM table
		from pyspark.sql.functions import upper;
		df.select(upper(col("string")))

Convert to Lowercase	SELECT LOWER(string)
	FROM table
		from pyspark.sql.functions import lower;
		df.select(lower(col("string")))

Concept	SQL	PySpark

Concatenate Strings	SELECT CONCAT(string1, string2) FROM table	from pyspark.sql.functions import concat;
		df.select(concat(col("string1"),
		col("string2")))

Trim String	SELECT TRIM(string)
	FROM table
		from pyspark.sql.functions import trim;
		df.select(trim(col("string")))

Substring	SELECT SUBSTRING(string, start, length) FROM table	from pyspark.sql.functions import substring;
		df.select(substring(col("string"),start, length))

CURDATE, NOW, CURTIME	SELECT CURDATE() FROM table	from pyspark.sql.functions import current_date;
		df.select(current_date())

CAST, CONVERT	SELECT CAST(column AS datatype) FROM table	df.select(col("column").cast("datatype"))
IF	SELECT IF(condition, value1, value2) FROM table	from pyspark.sql.functions import when, otherwise; df.select(when(condition,value1)\ .otherwise(value2))

Concept	SQL	PySpark

COALESCE	SELECT COALESCE(column1, column2, column3) FROM table	from pyspark.sql.functions import coalesce;
		df.select(coalesce("column1","column2", "column3"))

JOIN	JOIN table1 ON table1.column = table2.column	df1.join(df2, "column")
GROUP BY	GROUP BY column(s)	df.groupBy("column(s)")
PIVOT	PIVOT (agg_function(column) FOR pivot_column IN (values))	df.groupBy("pivot_column")\\ .pivot("column").agg(agg_function)

Logical Operators	SELECT column FROM table
	WHERE column1 = value
	AND column2 > value
		df.filter((col("column1") == value)
		& (col("column2") > value))

IS NULL, IS NOT NULL	SELECT column FROM table
	WHERE column IS NULL
		df.filter(col("column").isNull())\\
		.select("column")

IN	SELECT column FROM table
	WHERE column
	IN (value1,value2, value3)
		df.filter(col("column")\\
		.isin(value1,value2,value3))\\
		.select("column")

Concept	SQL	PySpark

LIKE	SELECT column FROM table
	WHERE column LIKE 'value%'
		df.filter(col("column").like("value%"))

BETWEEN	SELECT column FROM table
	WHERE column
	BETWEEN value1 AND value2
		df.filter((col("column") >= value1)
		& (col("column") <= value2))\
		.select("column")

UNION, UNION ALL	SELECT column FROM table1 UNION SELECT column FROM table2	df1.union(df2).select("column") or df1.unionAll(df2).select("column")

RANK, DENSERANK, ROWNUMBER	SELECT column, RANK() OVER
	(ORDER BY column) as
	rank FROM table
		from pyspark.sql import Window;
		from pyspark.sql.functions import rank;
		df.select("column",
		rank().over(Window.orderBy("column"))\
		.alias("rank"))

CTE	WITH cte1 AS (SELECT * FROM
	table1),
	SELECT * FROM cte1 WHERE
	condition
		df.createOrReplaceTempView("cte1");
		df_cte1 = spark.sql("SELECT * FROM
		cte1 WHERE condition");
		df_cte1.show() or
		df.filter(condition1).filter(condition2)

DDL operations

Concept	SQL	PySpark
Datatypes	INT: for integer values BIGINT: for large integer values FLOAT: for floating point values DOUBLE: for double precision floating point values CHAR: for fixed-length character strings VARCHAR: for variable-length character strings DATE: for date values TIMESTAMP: for timestamp values	In PySpark, the data types are similar, but are represented differently. IntegerType: for integer values LongType: for long integer values FloatType: for floating point values DoubleType: for double precision floating point values StringType: for character strings TimestampType: for timestamp values DateType: for date values

Create Table	CREATE TABLE table_name
	(column_name data_type
	constraint);
		df.write.format("parquet")\\
		.saveAsTable("table_name")

Concept	SQL	PySpark

Create Table with Columns definition	CREATE TABLE table_name(
	column_name data_type
	[constraints],
	column_name data_type
	[constraints],
	...);
		from pyspark.sql.types import StructType,
		StructField, IntegerType, StringType, DecimalType
		
		schema = StructType([
		StructField("id", IntegerType(), True),
		StructField("name", StringType(), False),
		StructField("age", IntegerType(), True),
		StructField("salary", DecimalType(10,2), True)])
		
		df = spark.createDataFrame([], schema)

Create Table with Primary Key	CREATE TABLE table_name(column_name data_type PRIMARY KEY, ...); If table already exists: ALTER TABLE table_name ADD PRIMARY KEY (column_name);	In PySpark or HiveQL, primary key constraints are not enforced directly. However, you can use the dropDuplicates() method to remove duplicate rows based on one or more columns. df = df.dropDuplicates(["id"])

Create Table with Auto Increment constraint	CREATE TABLE table_name(
	id INT AUTO_INCREMENT,
	name VARCHAR(255),
	PRIMARY KEY (id));
		not natively supported by the DataFrame API, but there are several ways to achieve the same functionality.
		
		from pyspark.sql.functions import
		monotonically_increasing_id
		df = df.withColumn("id",
		monotonically_increasing_id()+start_value)

Concept	SQL	PySpark

Adding a column	ALTER TABLE table_name
	ADD column_name datatype;
		from pyspark.sql.functions import lit
		df=df.withColumn("column_name", lit(None).cast("datatype"))

Modifying a column	ALTER TABLE table_name MODIFY column_name datatype;	df=df.withColumn("column_name", df["column_name"].cast("datatype"))
Dropping a column	ALTER TABLE table_name DROP COLUMN column_name;	df = df.drop("column_name")
Rename a column	ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name; In mysql, ALTER TABLE employees CHANGE COLUMN first_name first_name_new VARCHAR(255);	df =df.withColumnRenamed("existing_column", "new_column")