01 Intro to Spark-SQL -KirkYagami 👨‍💻🕵️

1. Introduction to Spark SQL

Spark SQL is a component of Apache Spark that allows for querying structured data using SQL queries. It integrates relational data processing with Spark's functional programming API, providing a unified interface for data manipulation. It supports a wide range of data formats and sources, including JSON, Parquet, Avro, and JDBC.

2. Setting Up Spark SQL

Before using Spark SQL, you need to initialize a `SparkSession`, which is the entry point for Spark SQL operations.

- **Example: Creating a SparkSession**

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Spark SQL Example") \
    .getOrCreate()
```

3. Loading Data with Spark SQL

You can load data into Spark SQL DataFrames from various sources. For this example, we'll use the `disney_plus_shows.csv` dataset.

- **Example: Loading CSV Data**

```
# Load CSV data into DataFrame
disney_df = spark.read \
  .format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .option("multiline", "true") \
  .option("quote", "\"") \
  .option("escape", "\"") \
  .load("disney_plus_shows.csv")

# Show schema and first few rows
disney_df.printSchema()
disney_df.show(truncate=False)
```

4. Registering DataFrames as SQL Tables

Once the DataFrame is loaded, you can register it as a temporary SQL table to run SQL queries.

- **Example: Registering a DataFrame**

```
# Register DataFrame as a SQL table
disney_df.createOrReplaceTempView("disney_shows")
```

5. Basic SQL Queries

With the DataFrame registered as a SQL table, you can now run SQL queries on it.

- **Example: Basic SQL Query**

```
# Run a SQL query
result = spark.sql("""
    SELECT title, genre, imdb_rating
    FROM disney_shows
    WHERE imdb_rating > 7.0
""")

result.show(truncate=False)
```

This query selects shows with an IMDb rating greater than 7.0.

6. Aggregation and Grouping

Spark SQL supports standard SQL operations such as aggregation and grouping.

- **Example: *Aggregating* Data**

```
# Aggregate and group data
genre_counts = spark.sql("""
    SELECT genre, COUNT(*) as count
    FROM disney_shows
    GROUP BY genre
    ORDER BY count DESC
""")

genre_counts.show(truncate=False)
```

This query counts the number of shows in each genre and orders the results by count in descending order.

7. Handling Missing Data

You can handle missing data in SQL queries using functions like `COALESCE` and `IS NULL`.

- **Example: Handling Null Values**

```
# Replace null values with a default value
cleaned_data = spark.sql("""
    SELECT
        title,
        COALESCE(genre, 'Unknown') AS genre,
```

```
        COALESCE(imdb_rating, 0) AS imdb_rating
    FROM disney_shows
""")

cleaned_data.show(truncate=False)
```

This query replaces null values in the `genre` and `imdb_rating` columns with 'Unknown' and 0, respectively.

8. Joining DataFrames

You can join multiple DataFrames using SQL.

- **Example: Self-Join**

```
# Self-join to find shows with the same genre
genre_joins = spark.sql("""
    SELECT a.title AS show1, b.title AS show2, a.genre
    FROM disney_shows a
    JOIN disney_shows b
    ON a.genre = b.genre
    WHERE a.title <> b.title
""")

genre_joins.show(truncate=False)
```

This query finds pairs of shows with the same genre.

9. Subqueries and Nested Queries

Spark SQL supports subqueries and nested queries, allowing for complex data retrieval.

- **Example: Subquery**

```
# Find the shows with the highest IMDb rating in each genre
top_rated_shows = spark.sql("""
    SELECT title, genre, imdb_rating
    FROM disney_shows
    WHERE imdb_rating = (
        SELECT MAX(imdb_rating)
        FROM disney_shows
        WHERE genre = disney_shows.genre
    )
""")

top_rated_shows.show(truncate=False)
```

This query selects the show with the highest IMDb rating in each genre.

10. Working with Dates and Times

Spark SQL provides functions to handle date and time operations.

- **Example: Extract Year from Date**

```
# Extract year from the released_at column
shows_with_year = spark.sql("""
    SELECT title, released_at, YEAR(released_at) AS release_year
    FROM disney_shows
""")

shows_with_year.show(truncate=False)
```

This query extracts the year from the `released_at` column.

- **Example: Filtering by Date Range**

```
# Filter shows released after January 1, 2020
recent_shows = spark.sql("""
    SELECT title, released_at
    FROM disney_shows
    WHERE released_at >= '2020-01-01'
""")

recent_shows.show(truncate=False)
```

This query selects shows released after January 1, 2020.

11. Advanced SQL Functions

Spark SQL includes advanced functions for data manipulation.

- **Example: String Functions**

```
# Extract the first 10 characters of the plot
short_plots = spark.sql("""
    SELECT title, SUBSTRING(plot, 1, 10) AS short_plot
    FROM disney_shows
""")

short_plots.show(truncate=False)
```

This query extracts the first 10 characters from the `plot` column.

- **Example: *Aggregating with Window Functions***

```
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number

# Use window function to rank shows within each genre by IMDb rating
window_spec = Window.partitionBy("genre").orderBy("imdb_rating DESC")

ranked_shows = disney_df.withColumn("rank", row_number().over(window_spec))
ranked_shows.createOrReplaceTempView("ranked_shows")

top_ranked = spark.sql("""
    SELECT title, genre, imdb_rating, rank
```

```
    FROM ranked_shows
    WHERE rank = 1
""")

top_ranked.show(truncate=False)
```

This query ranks shows within each genre by IMDb rating and selects the top-ranked show per genre.

12. Real Use Cases

1. **Business Intelligence:**
   - Spark SQL can be used to aggregate sales data, generate reports, and perform complex queries to gain insights into business operations.
2. **Data Warehousing:**
   - In data warehousing scenarios, Spark SQL can query large datasets, perform ETL operations, and create aggregated tables for analytics.
3. **Data Cleaning:**
   - Spark SQL can be used for cleaning and transforming data, handling missing values, and applying business rules to prepare data for analysis.
4. **Real-Time Analytics:**
   - Spark SQL supports streaming data sources, allowing real-time querying and analysis of live data streams.

13. Conclusion

Spark SQL provides a powerful and flexible interface for querying and manipulating structured data. By leveraging SQL queries within Spark, you can perform complex data transformations, aggregations, and analyses efficiently. The integration of SQL with Spark's distributed processing capabilities allows for handling large-scale data processing tasks with ease.