SPARK SQL Y DATAFRAMES

INDICE

INTRODUCCIÓN A SPARK SQL

- Contexto unificado e integrado
- Aprovecha el propio motor de computación de la máquina

Existen varias formas de trabajar con Spark SQL:

- Dataframe
- Dataset
- SQL

OPERACIONES SENCILLAS EN SPARK SQL Y PANDAS

Spark SQL	Pandas Dataframe
`df.select("columna1", "columna2")`	`df"columna1","columna2"`
`df.filter(df["edad"] > 18)`	`df[df["edad"] > 18]`
`df.sort(spark_df['cases'].desc())`	`df.sort_values(by='cases',ascending=False)`
`df.groupBy("country")`	`df.groupBy(by="country")`
`df.agg(F.sum("cases").alias("total_cases"))`	`df.agg('cases' : 'sum')`
`df1.join(df2, df1["clave"] == df2["clave"], "inner")`	`df1.join(df2, on='clave', how='inner')`

DATAFRAME API

- Abstracción de datos distribuidos
- Procesamiento de conjuntos de datos de manera distribuida
- Permite escribir código en Scala, Java, Python o R
- Contiene un esquema de información sobre el tipo de datos de cada columna

COMPARATIVA SPARK DATAFRAMES VS PANDAS DATAFRAMES

Comparación	Spark DataFrames	Pandas DataFrames
Distribución de datos	Entorno distribuido	Opera en una única máquina.
Optimizaciones de rendimiento	Utliza Catalyst para realiza optimizaciones a nivel de ejecución.	Se realizan a nivel de ejecución y dependen de la eficiencia de las operaciones individuales en el nodo único.
Escalabilidad	Escalable horizontalmente.	Puede enfrentar limitaciones en el manejo de grandes volúmenes de datos.

INTEGRACIÓN CON OTRAS FUENTES DE DATOS

MYSQL

- *Configurar las opciones de conexión*

opciones = { "url": "jdbc:mysql://localhost:3306/mi_base_de_datos", "driver": "com.mysql.jdbc.Driver", "dbtable": "mi_tabla", "user": "usuario", "password": "contraseña" }

- *Leer datos desde la base de datos en un DataFrame*

df_mysql = spark.read.format("jdbc").options(opciones).load()

- *Crear vista temporal*

df_mysql.createOrReplaceTempView("mi_vista_mysql")

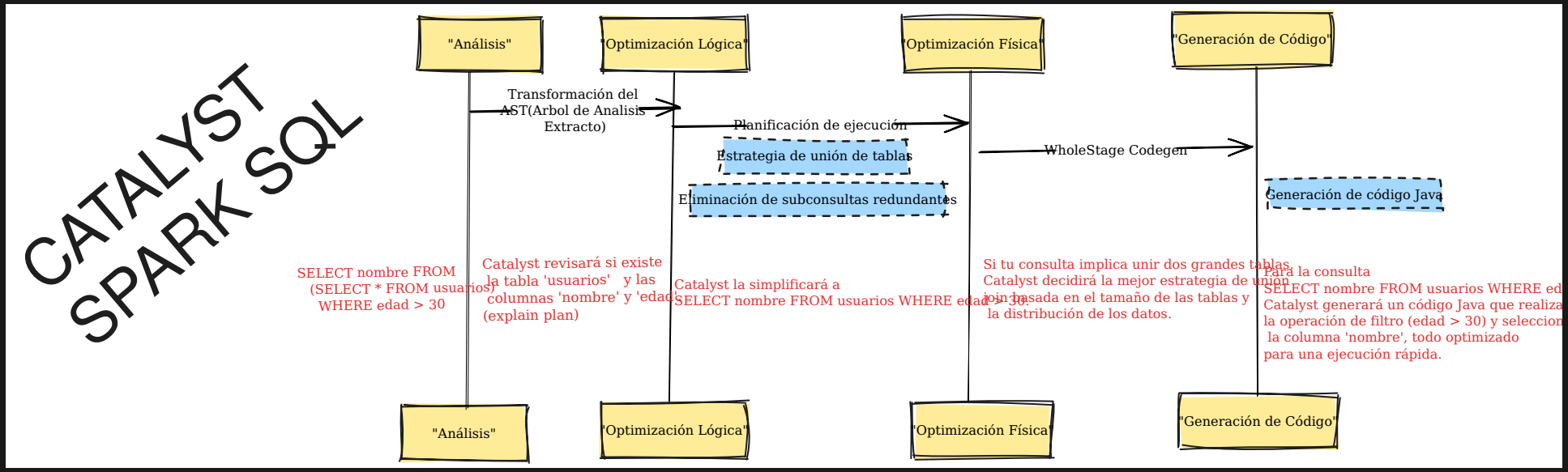
- *Ejecutar consultas SQL*

resultado_mysql = spark.sql("SELECT * FROM mi_vista_mysql") resultado_mysql.show()

OPTIMIZACIÓN Y TUNNING DE CONSULTA

OPTIMIZACIÓN DE CONSULTAS

CATALYST
SPARK SQL



TUNNING DE CONSULTA

Configuración/Práctica	*Descripción*
Particiones de datos	Usa la propiedad "spark.sql.shuffle.partitions" para ajustar el número de particiones para operaciones de shuffle. El valor predeterminado es 200.
Coalesce y Repartition en queries SQL	Usa coalesce(número_de_particiones) para reducir el número de particiones en un DataFrame o RDD. Usa repartition(número_de_particiones) para aumentar o disminuir el número de particiones y puede ser útil para optimizar el rendimiento en ciertas operaciones de Spark.
Formatos de Datos Serializados (Parquet, Avro, etc.)	Prefiere formatos de archivo optimizados para operaciones de lectura/escritura intermedias.
Evitar UDFs	Evita funciones definidas por el usuario cuando sea posible, ya que impiden la optimización.
Reducción de Operaciones de Shuffle	Minimiza las operaciones de shuffle, que son costosas en términos de I/O de disco y red.
Desactivar Registros de Depuración y de Información	Usa setLogLevel() para cambiar el nivel de registro.

FUNCIONALIDADES AVANZADAS

Código de ejemplo de funcionalidades avanzadas

CASOS DE USO Y APLICACIONES

Casos de uso reales :

- Yahoo : Transformación del algoritmo de aprendizaje automático
- ClearStory Data : Manejo y análisis de conjuntos de datos en tiempo real