Ingeniería para el Procesado Masivo de Datos

Dra. Ana Beatriz Medina Ruiz

SPARK II

UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Tema 4: **SPARK I**

Comparación: Vagrant vs Docker vs VirtualBox

Característica	Vagrant	Docker	VirtualBox
Tipo de Virtualización	Virtualización completa	Contenedores ligeros	Virtualización completa
Uso principal	Entornos de desarrollo replicables	Despliegue y gestión de microservicios	Virtualización general
Requiere SO completo en VM?	Sí	No (usa contenedores sobre el kernel del host)	Sí
Aislamiento	Total (cada VM tiene su SO)	Parcial (comparten kernel del host)	Total
Rendimiento	Medio (depende de la VM)	Alto (contenedores son más ligeros)	Bajo (mayor consumo de recursos)
Facilidad de configuración	Fácil con `Vagrantfile`	Fácil con `Dockerfile`	Requiere configuración manual
Casos de uso	Desarrollo y pruebas	Microservicios, CI/CD	Sistemas operativos completos

Tema 4: **SPARK I**

¿Cuándo elegir Vagrant?

✓ Cuando necesites un **entorno de desarrollo replicable** en varios equipos.
✓ Cuando trabajes con múltiples sistemas operativos en desarrollo.
✓ Cuando necesites **automatizar la configuración** de una VM (en lugar de hacerlo manualmente en VirtualBox).
✓ Cuando quieras simular servidores en local antes de desplegar en producción.

Tema 4: **SPARK II**

- 4.1. Identificar Ventajas de Usar DataFrames en lugar de RDD
- 4.2. Diferencia y Similitudes de SparkSQL con la API estructurada
- 4.3. Funciones típicas manipulación dataframes con la API estructurada y SpakSql

Tema 4: Ventajas de Usar DataFrames en lugar de RDD

Característica	DataFrame	RDD
Esquema	Columnas con nombres y tipos	Sin esquema, datos genéricos
Optimización	Sí (Catalyst + Tungsten)	No automática
API	Alto nivel, SQL-like, conciso	Bajo nivel, map/filter, más código necesario
Serialización/Memoria	Binario optimizado, off-heap, eficiente	Java/Kryo, GC, fragmentación de memoria
Integración	BI y varias fuentes (JSON, Parquet, JDBC...)	Soporte limitado
Uso recomendado	Casi siempre para datos estructurados	Sólo casos especiales no estructurados

Tema 4: Diferencia y Similitudes de SparkSQL con la API estructurada

1. Forma de expresar lógica

Aspecto	Spark SQL	API DataFrame / Dataset (Scala/Java/Python)
Estilo	Declarativo (SQL)	Imperativo / funcional
Legibilidad	Familiar para usuarios SQL	Modular, más control y legible en codebases
Composición del código	Menos reusable, fragmentos SQL	Código reutilizable con funciones, loops
Testing	SQL difícil de testear unitariamente	APIs permiten pruebas unitarias y mejores mensajes de error

▸ Tema 4: Funciones típicas manipulación dataframes con la API estructurada y SpakSQL

1. Crear DataFrame desde SQL
Ejecutar SQL sobre una tabla temporal:

```
df.createOrReplaceTempView("table_name") result = spark.sql("SELECT * FROM table_name WHERE col1 > 50")
```

2. Selección de columnas (similar a SQL)
Seleccionar columnas en SQL:

```
result = spark.sql("SELECT col1, col2 FROM table_name")
```

3. Filtrado de datos
Filtrar con condiciones en SQL:

```
result = spark.sql("SELECT * FROM table_name WHERE col1 > 50")
```

4. Agregar columnas
Crear una nueva columna calculada

```
result = spark.sql("SELECT col1, col2, col1 * 2 AS new_col FROM table_name")
```

▸ Tema 4: Funciones típicas manipulación dataframes con la API estructurada y SpakSQL

5. Operaciones agregadas (GROUP BY)
Agrupar y aplicar funciones agregadas:

```
result = spark.sql("SELECT col1, AVG(col2) AS avg_col2, SUM(col3) AS sum_col3
FROM table_name GROUP BY
```

Tema 4: Operaciones comunes con DataFrames (API estructurada)

1. Creación de DataFrame
Desde una colección (RDD o lista):

```
df = spark.createDataFrame(data, ["col1", "col2", "col3"])
```

2. Selección de columnas
Seleccionar columnas específicas:

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

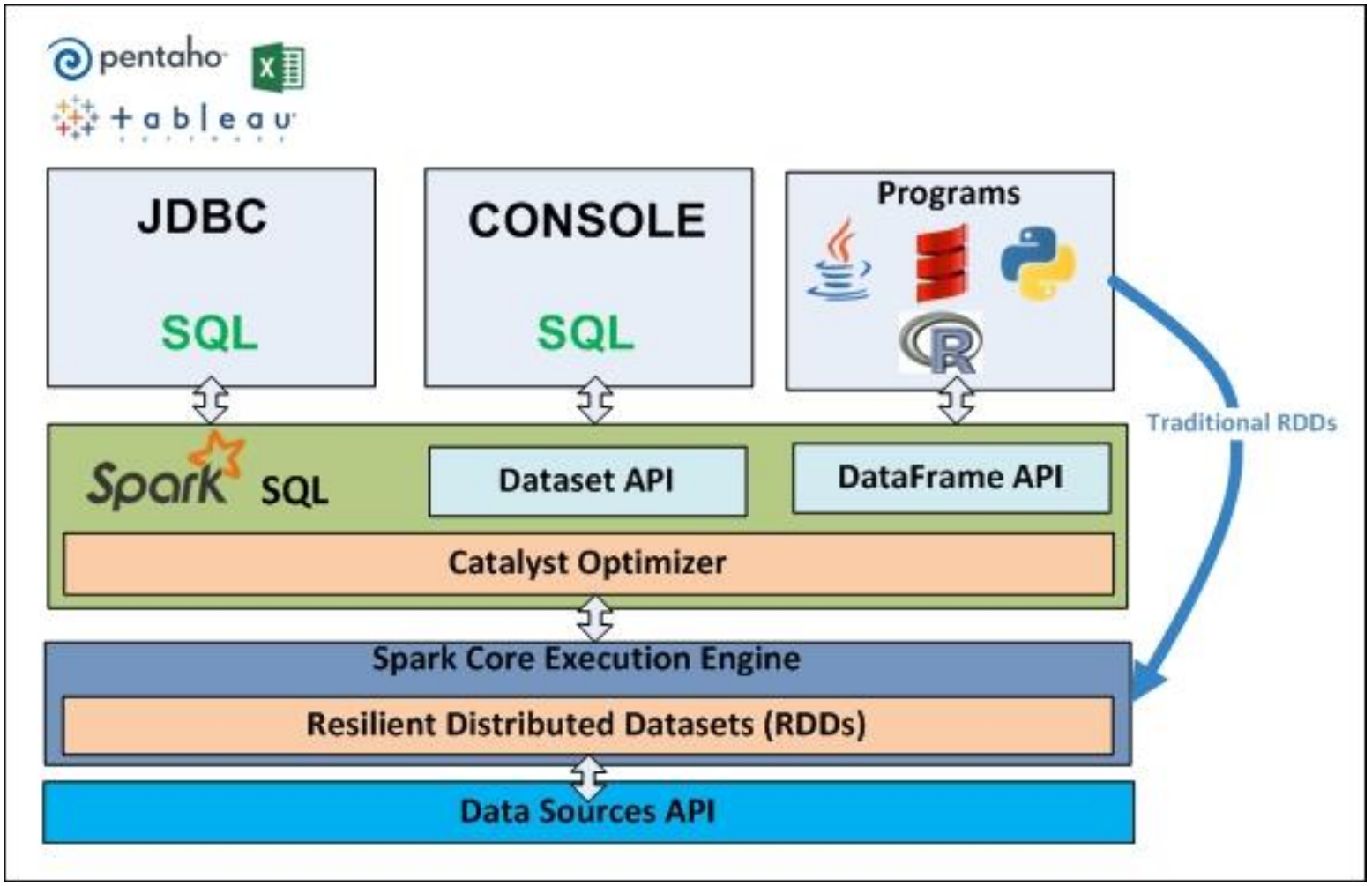
Renombrar columna:

```
df.withColumnRenamed("old_name", "new_name")
```

3. Operaciones agregadas (GROUP BY)
Agrupar y aplicar funciones agregadas:

```
result = spark.sql("SELECT col1, AVG(col2) AS avg_col2, SUM(col3) AS sum_col3
FROM table_name GROUP BY col1")
```

▸ Tema 4: Arquitectura de DataFrame en Spark



Tema 4: JOB en Spark

Un **Job es el trabajo que Spark hace cuando tú le dices: "Dame un resultado"**.

Ese Job se encarga de:

✅ Leer los datos reales del archivo
✅ Filtrar las edades mayores a 30
✅ Mostrar el resultado en pantalla

Un **Job** en Spark es como decir:
“Haz todo lo necesario para darme este resultado ahora.”

Tema 4: ¿Qué hace Spark?

1. **Primero lee el archivo** → No hace nada aún (solo se prepara).
2. **Aplica el filtro** → Tampoco hace nada aún.
3. **Cuando haces** `.show()` → **lanza un Job**.

NOMBRE	EDAD
ANA	28
CARLOS	35
GLORIA	42

```
df_filtrado.show()     # Acción  Transformación perezosa
df_filtrado.count()    # Acción
df_filtrado.write.csv("salida.csv")  # Acción
```

```
df = spark.read.csv("personas.csv",
header=True)
df_mayores = df.filter(df["edad"] > 30)
df_mayores.show()
```

www.unir.net