Big Data Hadoop and Spark Developer

simplilearn

Spark SQL - Processing DataFrames

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

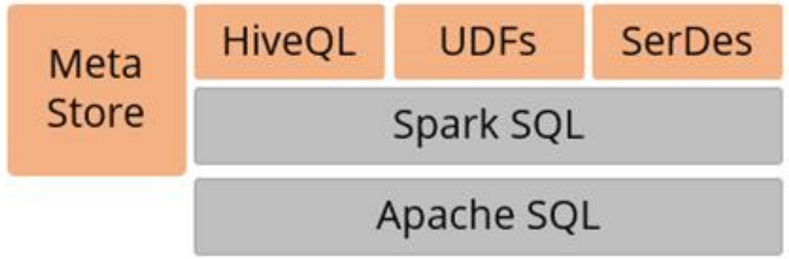
- Explain the importance and features of Spark SQL
- Describe the methods to convert RDDs to DataFrames
- Load existing data into a DataFrame

DATA AND
ARTIFICIAL INTELLIGENCE

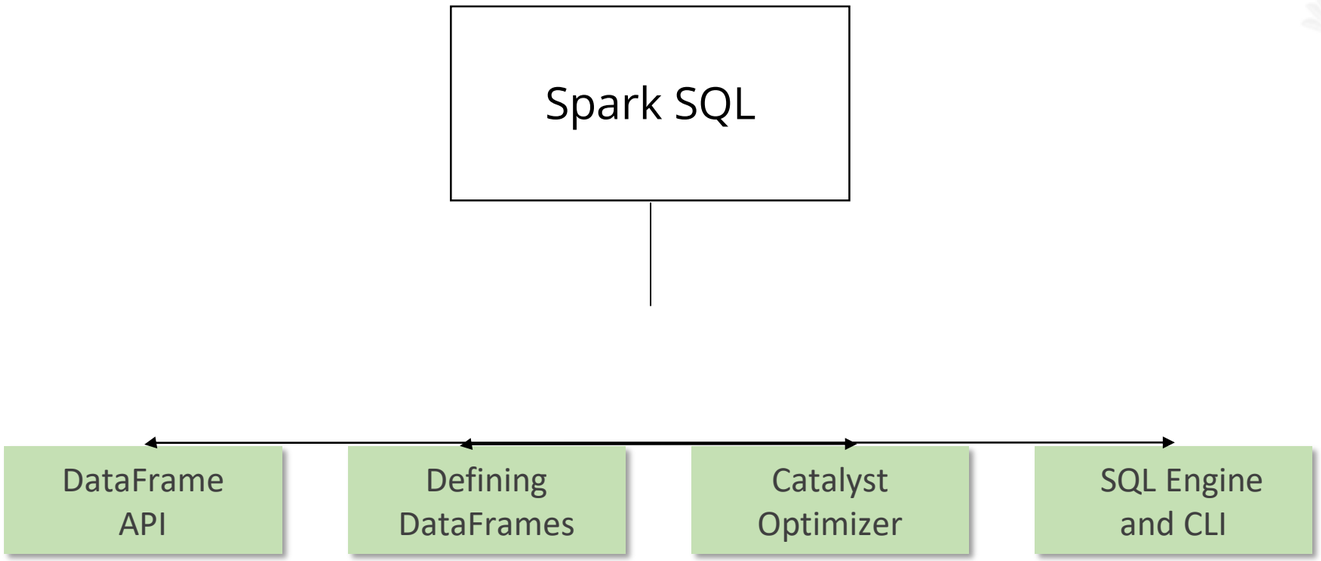
Spark SQL Introduction

What Is Spark SQL?

Spark SQL is a module for structured data processing that is built on top of core Spark.



Spark SQL provides the following:



Importance of Spark SQL

Spark SQL provides four main capabilities for using structured and semi-structured data.

Acts as a distributed SQL query engine

Provides DataFrames for programming abstraction

Allows users to query structured data in Spark programs

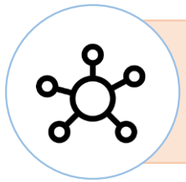
Can be used with platforms such as Scala, Java, R, and Python

Advantages of Spark SQL

Hive Compatibility: Compatible with the existing Hive queries, UDFs, and data

Integrated: Mixes SQL queries with Spark programs

Unified Data Access: Loads and queries data from different sources



Standard Connectivity: Connects through JDBC or ODBC

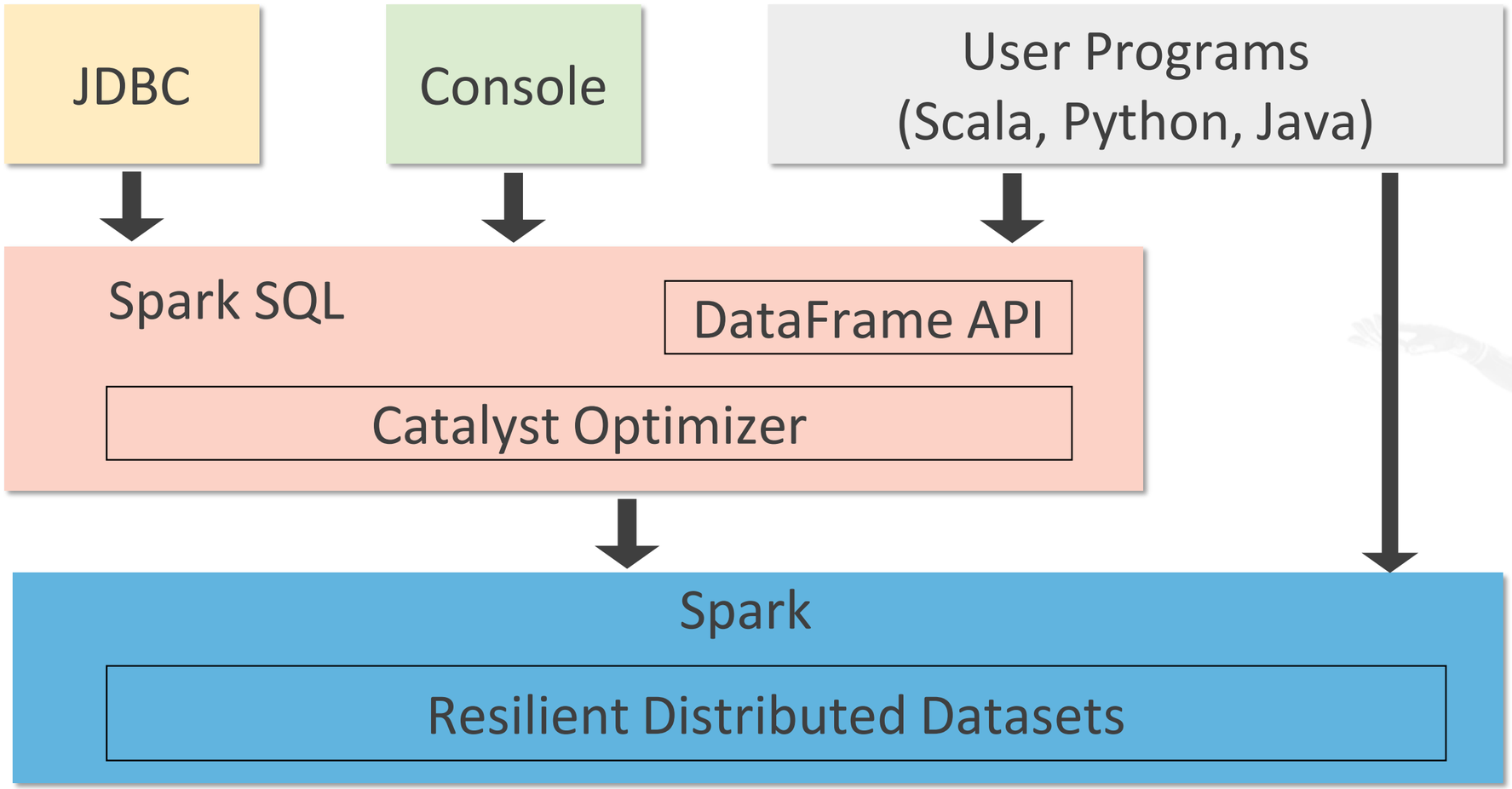
Performance: Executes the most optimal plans

DATA AND
ARTIFICIAL INTELLIGENCE

Spark SQL Architecture

Spark SQL Architecture

The below diagram shows the typical architecture and interfaces of Spark SQL.



SQLContext

The SQLContext class or any of its descendants acts as the entry point into all functionalities.

Q

How to get the benefit of a superset of the basic SQLContext functionality?

Build a HiveContext to:

- Use the writing ability for queries
- Access Hive UDFs and read data from Hive tables

```
val sqlContext = new
org.apache.Spark.sql.SQLContext(sc)
```

SQLContext

Points to Remember:

- You can use the Spark.sql.dialect option to select the specific variant of SQL used for parsing queries
- On a SQLContext, the sql function allows applications to programmatically run SQL queries and then return a DataFrame as the result.

```
val df = sqlContext.sql("SELECT * FROM table")
```

DATA AND
ARTIFICIAL INTELLIGENCE

DataFrames

DataFrames

DataFrames represent a distributed collection of data in which data is organized into columns that are named.

Construct a DataFrame

Use sources such as tables in Hive, structured data files, existing RDDs, and external databases.

Convert Them to RDDs

Call the rdd method, that returns the DataFrame content, as an RDD of rows.

In the prior versions of Spark SQL API, SchemaRDD has been renamed as DataFrame.

Creating DataFrames

DataFrames can be created:

- From an existing structured data source
- From an existing RDD
- By performing an operation or query on another DataFrame
- By programmatically defining a schema

Creating a DataFrame

```
val sc: SparkContext          // An existing SparkContext.
val sqlContext = new org.apache.Spark.sql.SQLContext(sc)

val df = sqlContext.read.json("examples/src/main/resources/customers.json")
// Displays the content of the DataFrame to stdout

df.show()
```

Assisted Practice

Handling Various Data Formats

Duration: 10 mins

Problem Statement: In this demonstration, you will learn how to handle various data formats.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Using DataFrame Operations

DataFrames provide a domain-specific language that can be used for structured data manipulation in Java, Scala, and Python.

```
val sc: SparkContext
    val sqlContext = new org.apache.Spark.sql.SQLContext(sc)

    // Create the DataFrame
    val df = sqlContext.read.json("examples/src/main/resources/customers.json")

// Show the content of the DataFrame
df.show()

    // Print the schema in a tree format
    df.printSchema()

    // Select only the "name" column
    df.select("name").show()
```

Using DataFrame Operations

```
// Select everybody, but increment the age by 1
df.select(df("name"), df("age") + 1).show()

// Select people older than 21
df.filter(df("age") > 21).show()

// Count people by age
df.groupBy("age").count().show()
```

Assisted Practice

Implement Various DataFrame Operations

Duration: 10 mins

Problem Statement: In this demonstration, you will learn how to implement various DataFrame operations like filter, aggregates, joins, count, and sort.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

UDFs: User Defined Functions

UDFs are used to define new column-based functions that help Spark SQL in transforming datasets.

The example below defines a UDF to convert a given text to upper case.

```
-> val dataset = Seq((0, "hello"),(1, "world")).toDF("id","text")

-> val upper: String =&amp;amp;amp;amp;gt; String =_.toUpperCase

-> import org.apache.Spark.sql.functions.udf

-> val upperUDF = udf(upper)

-> dataset.withColumn("upper", upperUDF('text)).show
```

UDAF: User-Defined Aggregate Functions

Spark SQL allows users to define custom aggregation functions called User-Defined Aggregate Functions or UDAFs.

Type	Value	Type Total
A	3	15
A	12	15
B	7	9
B	2	9
C	9	20
C	11	20

UDAFs are very useful when performing aggregations across groups or columns.

UDAF: User-Defined Aggregate Functions

In order to write a custom UDAF, you need to extend UserDefinedAggregateFunction and define four methods.

```
class GeometricMean extends UserDefinedAggregateFunction
```

Initialize	On a given node, this method is called once for each group.
Update	For a given group, Spark will call “update” for each input record of that group.
Merge	If the function supports partial aggregates, Spark computes partial result and combines them together.
Evaluate	Once all the entries for a group are exhausted, Spark will call “evaluate” to get the final result.

Assisted Practice

UDF and UDAF

Duration: 10 mins

Problem Statement: In this demonstration, you will learn how to create UDF and UDAF.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Interoperating with RDDs

To convert existing RDDs into DataFrames, Spark SQL supports two methods:

Reflection-Based

- Infers an RDD schema containing specific types of objects
- Works well when the schema is already known while writing the Spark application

Programmatic

- Lets you build a schema and apply to an already existing RDD
- Allows you to build DataFrames when you do not know the columns and their types until runtime

Using the Reflection-Based Approach

For Spark SQL, the Scala interface allows users to convert an RDD with case classes to a DataFrame automatically.

1. The case class has the table schema, where the argument names to the case class are read using the reflection method.
2. The case class can be nested and used to contain complex types like sequence of arrays
3. Scala Interface implicitly convert the resultant RDD to a DataFrame and register it as a table.

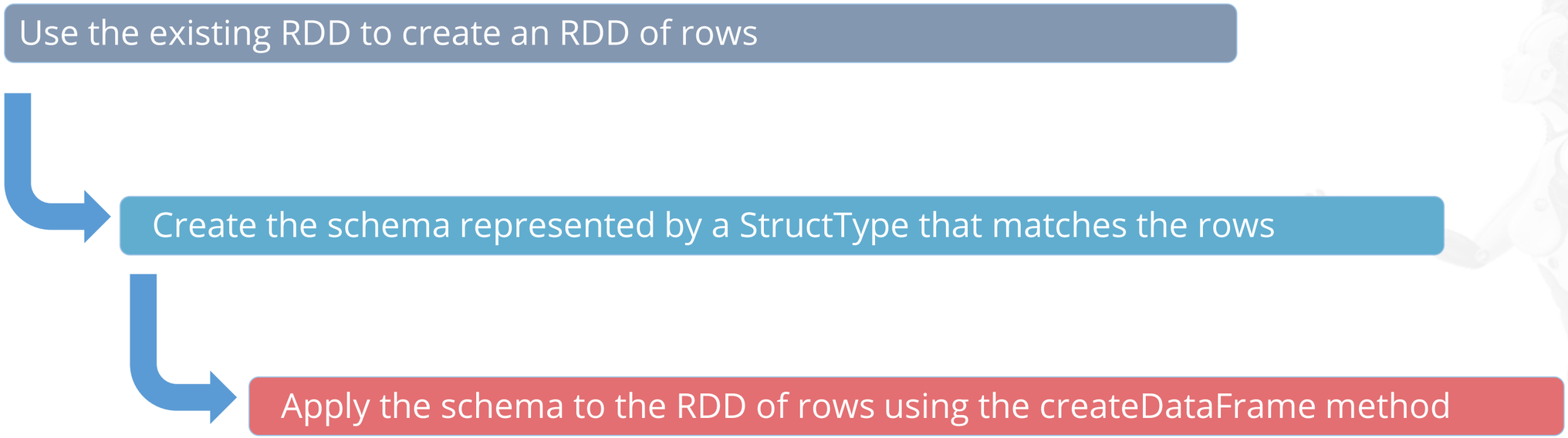
Using the Reflection-Based Approach

```
val sqlContext = new org.apache.Spark.sql.SQLContext(sc)
import sqlContext.implicits._
case class Person(name: String, age: Int)
// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("examples/src/main/resources/people.txt").map(_.split(",")).map(p => Person(p(0),
p(1).trim.toInt)).toDF()
people.registerTempTable("people")
// SQL statements can be run by using the sql methods provided by sqlContext.
val teenagers = sqlContext.sql("SELECT name, age FROM people WHERE age >= 13 AND age <= 19")
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
// or by field name:
teenagers.map(t => "Name: " + t.getAs[String]("name")).collect().foreach(println)
// row.getValuesMap[T] retrieves multiple columns at once into a Map[String, T]
teenagers.map(_.getValuesMap[Any](List("name", "age"))).collect().foreach(println)
```

Using the Programmatic Approach

This method is used when you cannot define case classes ahead of time.
For example, when the records structure is encoded in a text dataset or a string.

The below steps are used for defining case classes:



Using the Programmatic Approach

```
val sqlContext = new org.apache.Spark.sql.SQLContext(sc)
val people = sc.textFile("examples/src/main/resources/people.txt")

// The schema is encoded in a string
val schemaString = "name age"
import org.apache.Spark.sql.Row;
import org.apache.Spark.sql.types.{StructType,StructField,StringType};

// Generate the schema based on the string of schema, Convert records of the RDD (people)
   to Rows and Apply the schema to the RDD.
val schema =   StructType( schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))
val rowRDD = people.map(_.split(",")).map(p => Row(p(0), p(1).trim))
val peopleDataFrame = sqlContext.createDataFrame(rowRDD, schema)
peopleDataFrame.registerTempTable("people")
```

Assisted Practice

Process DataFrame Using SQL Query

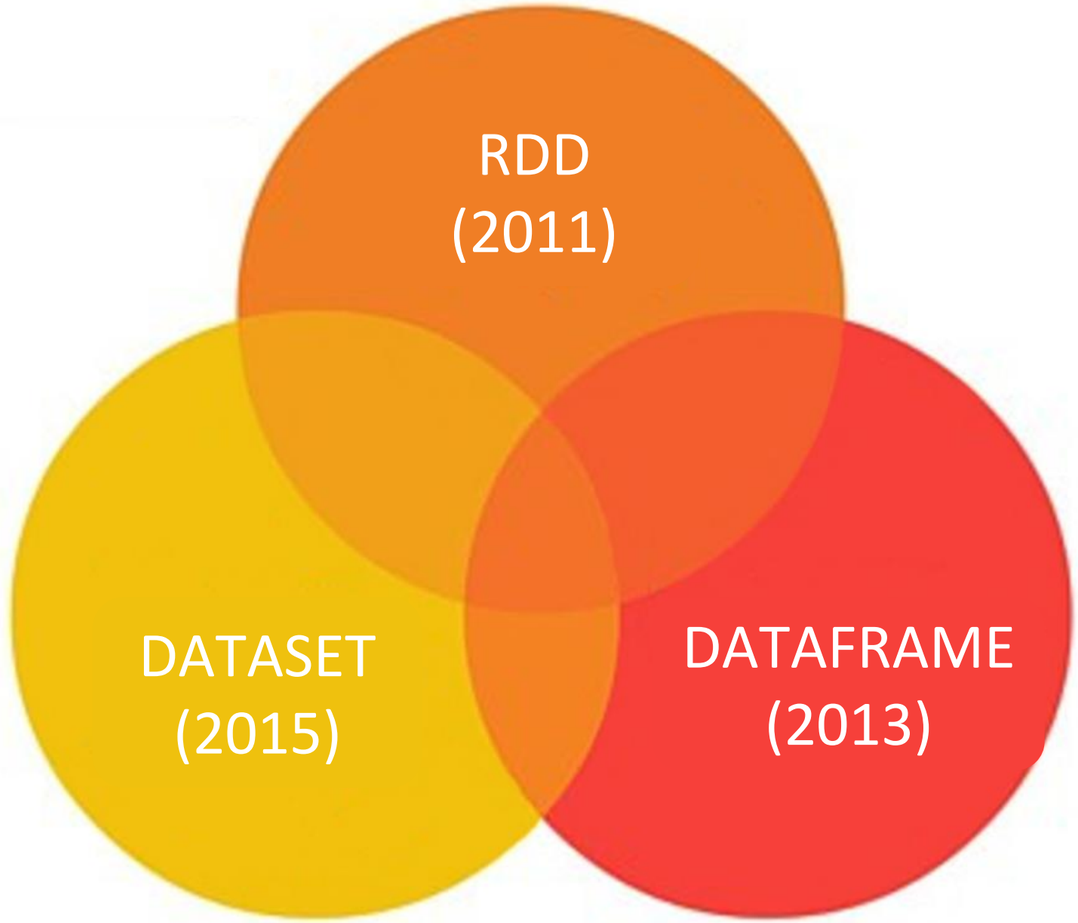
Duration: 10 mins

Problem Statement: In this demonstration, you will learn how to process DataFrame(s) using SQL queries.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

RDD vs. DataFrame vs. Dataset

Apache Spark provides three types of APIs: RDD, DataFrame, and Dataset.



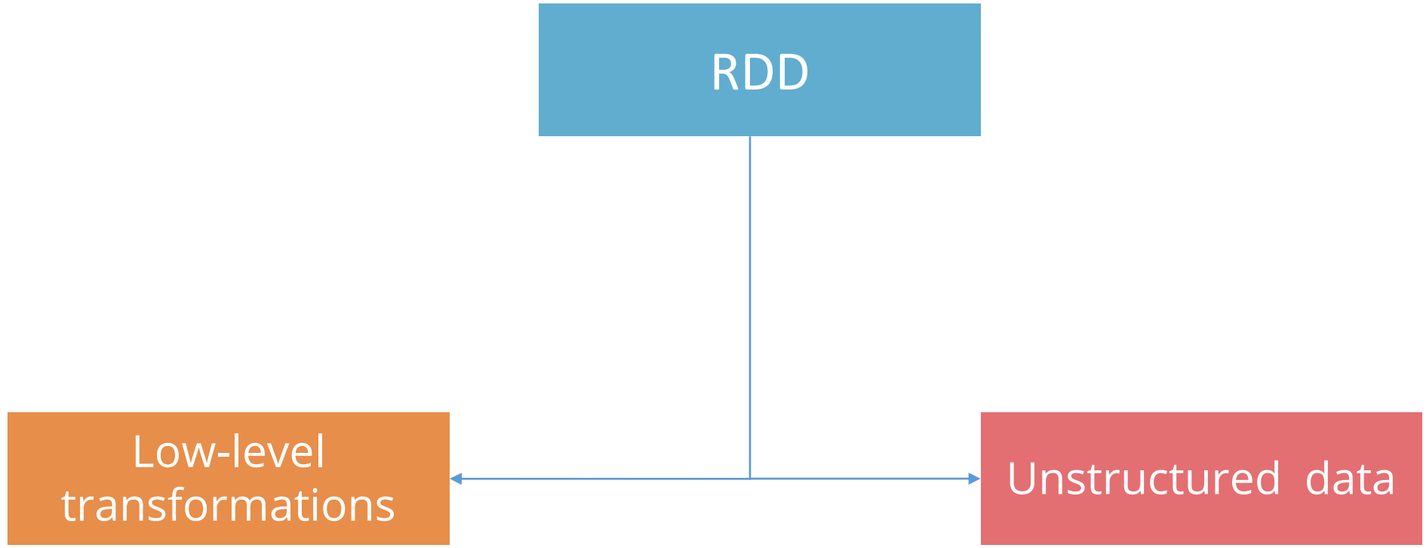
Example: Filter By Attribute

Given below are the various ways to filter an attribute using the three APIs.

RDD	`rdd.filter(_.marks > 75)`
DataFrame	`df.filter("marks > 75");`
Dataset	`dataset.filter(_.marks > 75);`

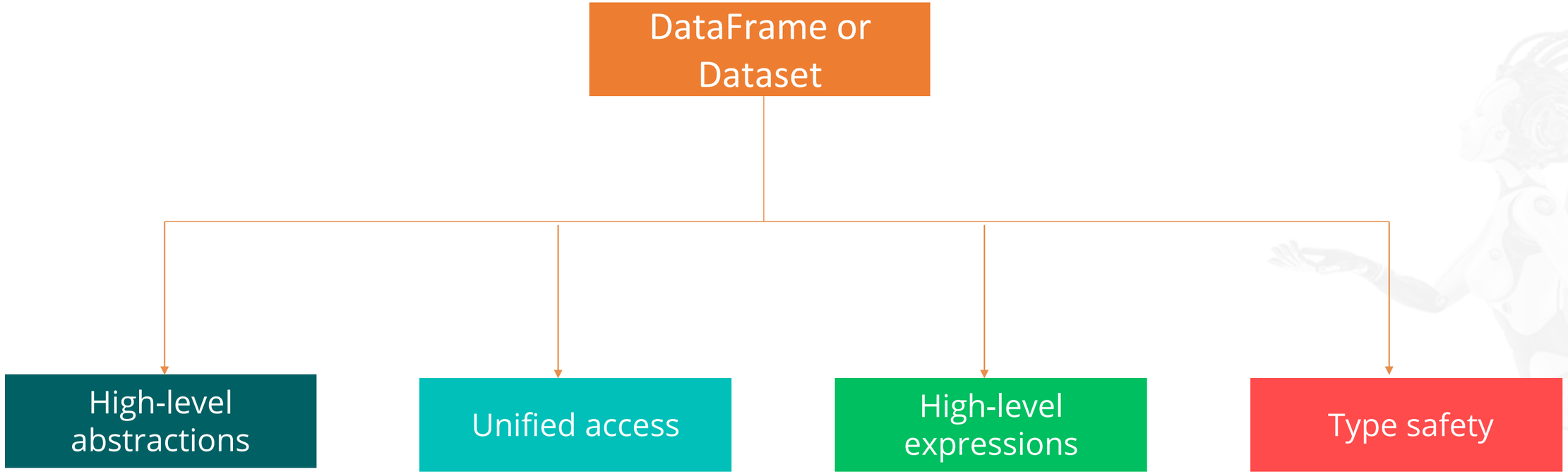
Use Case: RDD API

We can use any of the three APIs based on the requirements.



Use Case: DataFrame or Dataset API

In Spark 2.0, DataFrame APIs and Datasets APIs were merged, unifying data processing capabilities across libraries.



Unassisted Practice

Processing DataFrames

Duration: 15 mins

Problem Statement: Using Scala, perform the below tasks on Spark DataFrames.

- Create the case classes with the following fields: Department, Employee, and DepartmentWithEmployees.
  Note: Create the DepartmentWithEmployees instances from Departments and Employees.
  Insert at least four values.

- Create two DataFrames from the list of the above case classes.

- Combine the two DataFrames and write the combined DataFrame to a parquet file.

- Use filter() or where() clause to return the rows whose first name is either “Alice” or “John”.
  Note: Use first name filter as per your entries.

- Retrieve rows with missing firstName or lastName.

- Find the distinct (firstName, lastName) combinations.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Unassisted Practice

Steps to Perform

// Create the case classes

```
case class Department(id: String, name: String)
case class Employee(firstName: String, lastName: String, email: String, salary: Int)
case class DepartmentWithEmployees(department: Department, employees: Seq[Employee])
```

// Create the Departments

```
val department1 = new Department("123456", "Computer Science")
val department2 = new Department("789012", "Mechanical Engineering")
val department3 = new Department("345678", "Theater and Drama")
val department4 = new Department("901234", "Indoor Recreation")
```

// Create the Employees

```
val employee1 = new Employee("Alice", "Mathew", "no-reply@harvard.edu", 100000)
val employee2 = new Employee("John", "Singleton", "no-reply@stanford.edu", 120000)
val employee3 = new Employee("matei", null, "no-reply@oxford.edu", 140000)
val employee4 = new Employee(null, "wendell", "no-reply@princeton.edu", 160000)
```

Unassisted Practice

Steps to Perform

```
// Create the DepartmentWithEmployees instances from Departments and Employees

val departmentWithEmployees1 = new DepartmentWithEmployees(department1, Seq(employee1, employee2))
val departmentWithEmployees2 = new DepartmentWithEmployees(department2, Seq(employee3, employee4))
val departmentWithEmployees3 = new DepartmentWithEmployees(department3, Seq(employee1, employee4))
val departmentWithEmployees4 = new DepartmentWithEmployees(department4, Seq(employee2, employee3))

// Create DataFrames from a list of the case classes

val departmentsWithEmployeesSeq1 = Seq(departmentWithEmployees1, departmentWithEmployees2)
val df1 = departmentsWithEmployeesSeq1.toDF()
display(df1)

val departmentsWithEmployeesSeq2 = Seq(departmentWithEmployees3, departmentWithEmployees4)
val df2 = departmentsWithEmployeesSeq2.toDF()
display(df2)
```

Steps to Perform

```
// combining the two DataFrames

val unionDF = df1.unionAll(df2)
display(unionDF)

// Write the combined DataFrame to a Parquet file

unionDF.write.parquet("/user/simpli_learn/simplitest")

// Explode the employees column

val flattenDF = parquetDF.select(functions.explode($"employees")).flattenSchema
val columnsRenamed = Seq("firstName", "lastName", "email", "salary")
val explodeDF = flattenDF.toDF(columnsRenamed: _*)
explodeDF.show()
```

Unassisted Practice

Steps to Perform

```
// Using filter()  to return the rows where first name is either 'Alice' or 'John'

val filterDF = explodeDF
  .filter($"firstName" === "Alice" || $"firstName" === "John")
  .sort($"lastName".asc)
display(filterDF)

// Retrieve rows with missing firstName or lastName

val filterNonNullDF = nonNullDF.filter($"firstName" === "" || $"lastName" === "").sort($"email".asc)
display(filterNonNullDF)
```

Unassisted Practice

Steps to Perform

```
// aggregations using agg() and countDistinct()

import org.apache.spark.sql.functions._

val countDistinctDF = nonNullDF.select($"firstName", $"lastName")
  .groupBy($"firstName", $"lastName")
  .agg(countDistinct($"firstName") as "distinct_first_names")
display(countDistinctDF)
```

Key Takeaways

You are now able to:

- Explain the importance and features of Spark SQL
- Describe the methods to convert RDDs to DataFrames
- Load existing data into a DataFrame

DATA AND
ARTIFICIAL INTELLIGENCE

Knowledge Check

Knowledge Check 1

Which of the following is built on Spark for general purpose processing?

a. Hive

b. Spark SQL

c. MapReduce

d. None of the above

Knowledge Check 1

Which of the following is built on Spark for general purpose processing?

a. Hive

b. Spark SQL

c. MapReduce

d. None of the above

The correct answer is **b.**

Spark SQL is built on Spark for general purpose processing.

Knowledge Check 2

What are the functions of Spark SQL?

a. It provides the DataFrame API.

b. It defines DataFrames containing rows and columns.

c. It provides the Catalyst Optimizer along with SQL engine and CLI.

d. All of the above

Knowledge Check 2

What are the functions of Spark SQL?

a. It provides the DataFrame API.

b. It defines DataFrames containing rows and columns

c. It provides the Catalyst Optimizer along with SQL engine and CLI

d. All of the above

The correct answer is **d.**

Spark SQL provides the DataFrame API, defines DataFrames containing rows and columns, and provides the Catalyst Optimizer along with SQL engine and CLI.

Knowledge Check 3

Which of the following represents a distributed collection of data in which data is organized into columns that are named?

a. Spark SQL

b. SparkContext

c. DataFrames

d. Data Organizer

Knowledge Check 3

Which of the following represents a distributed collection of data in which data is organized into columns that are named?

a. Spark SQL

b. SparkContext

c. DataFrames

d. Data Organizer

The correct answer is **b.**

SQLContext represents a distributed collection of data in which data is organized into columns that are named.

Knowledge Check 4

Which of the following methods is utilized to convert RDDs to DataFrames?

a. Programmatic

b. Reflective-Based

c. Both a and b

d. None of the above

Knowledge Check 4

Which of the following methods is utilized to convert RDDs to DataFrames?

a. Programmatic

b. Reflective-Based

c. Both a and b

d. None of the above

The correct answer is **c.**

To convert existing RDDs into DataFrames, Spark SQL supports two methods: Programmatic and Reflective-Based.

Lesson-End Project

Problem Statement:

Every country has data for each of the companies that are operating in that country. Registering a company is mandatory as it has to provide information about its profit/loss and other details. “People data Labs” is one of the biggest data companies which collects and provides data. They recently open-sourced the datasets of “Global companies”, which operate in various countries. Data can be used to find a company in any specific industry, their employee count, and website details.

all_companies_details.csv contains the below fields:

1. Name
2. Domain
3. Year founded in
4. Industry
5. Size range
6. Country
7. LinkedIn URL
8. Current employee estimate
9. Total employee estimate

Lesson-End Project

Your task is to find:

1. The companies which were registered before 1980 in each country
2. The companies which were founded between 1990 and 2000 and has a size range of 10001+
3. The top industries having the maximum number of companies
4. The top 5 countries having the maximum number of companies that belong to the “facilities services” industry

DATA AND
ARTIFICIAL INTELLIGENCE

Thank You