Big Data Hadoop and Spark Developer

Spark SQL and DataFrames

Learning Objectives

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

- Identify the importance and features of Spark SQL
- Convert RDDs to DataFrames
- Load data through different sources
- Illustrate user-defined functions
- Interoperate with RDDs

Spark SQL Introduction

Spark SQL

- It is a module for structured data processing that is built on top of Spark Core.
- The Spark SQL module provides an abstraction called DataFrame, which simplifies the process of working with structured datasets.
- It supports multiple programming languages, such as Java, R, Python, and Scala.
- It provides row-level updates and real-time online transaction processing.

Need for Spark SQL

- Spark SQL was created to overcome the limitations of Apache Hive.
- It was released in 2014.
- It is built on top of the Spark Core module.
- It can integrate with NoSQL databases, but it primarily works with RDBMS.

Limitations of Apache Hive

Apache Hive uses MapReduce as the execution engine which leads to lags in performance.

Real-time online transactional processing (OLTP) queries are not possible in Hive.

Row-level updates are not possible in Hive.

Hive cannot drop encrypted databases.

Limitations of Apache Hive

APACHE Spark™

- It executes faster as it has in-memory computation.
- It can run Hive queries as it has no migration issues.
- Real-time querying is possible with Spark SQL.
- It has the capability to interact with Hive metastore.

Limitations of Apache Hive

- Mixes SQL queries with Spark programs
- Provides standard connectivity with the help of JDBC or ODBC
- Executes the optimal plan for better performance
- Supports multiple file formats, such as CSV, Parquet, and Avro

Spark SQL Architecture

Spark SQL Architecture

Spark SQL consist of four libraries that deal with relational and procedural processing.

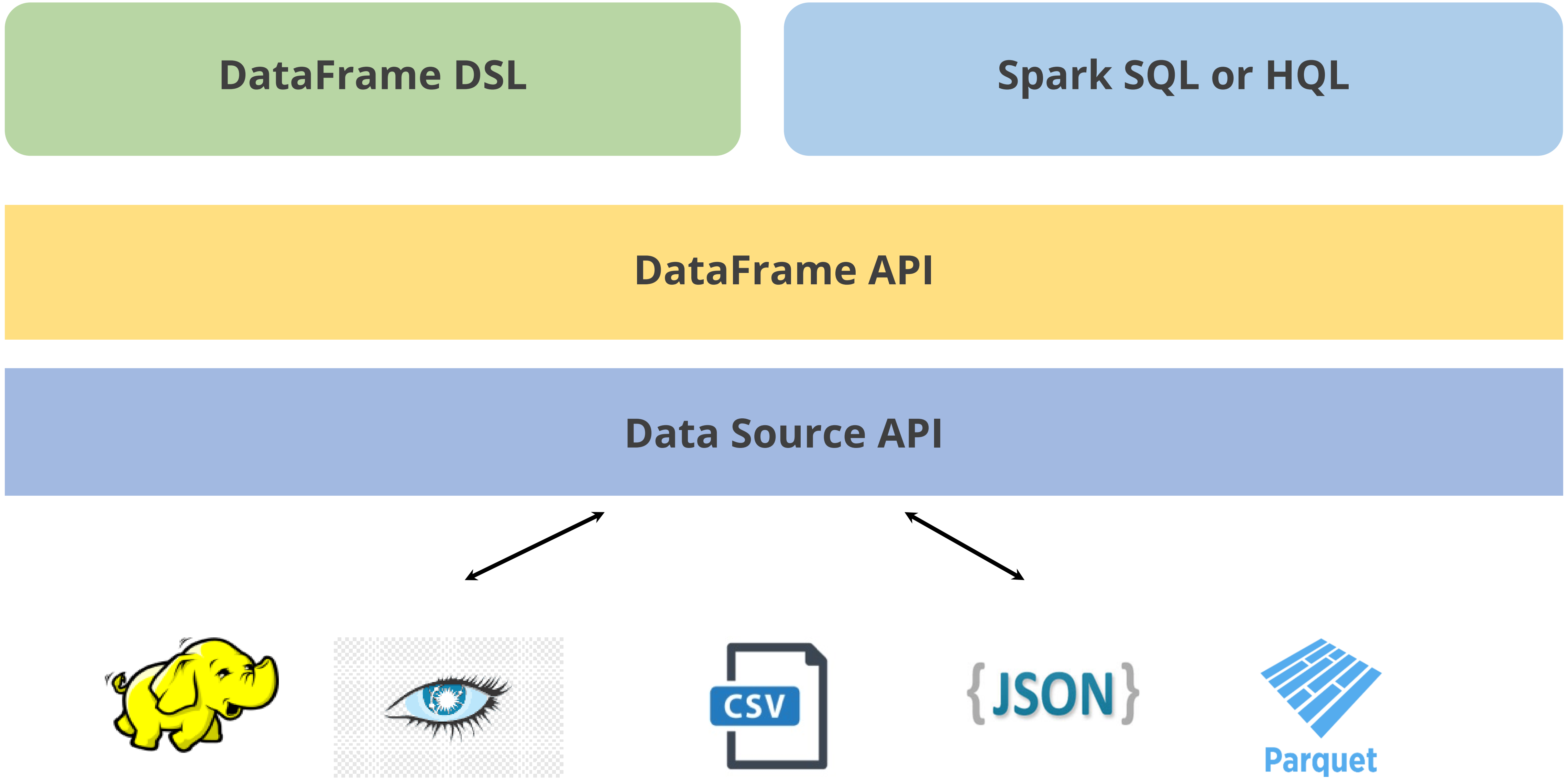
Data source API

DataFrame API

SQL service

DataFrame DSL

Spark SQL Architecture



DataFrame DSL

Spark SQL or HQL

DataFrame API

Data Source API

Spark SQL Architecture

Data Source API

It helps one read and store structured and semi-structured data in Spark SQL. Data Source API can load files from multiple file formats.

DataFrame API

It converts the data into tabular columns that can be read using Data Source API. DataFrame is a distributed collection of data organized into named columns. Tabular columns helps perform SQL operations.

SQL service

SQL service is the entry point for working with structured data in Apache Spark. It is also used to fetch the result from interpreted and optimized data.

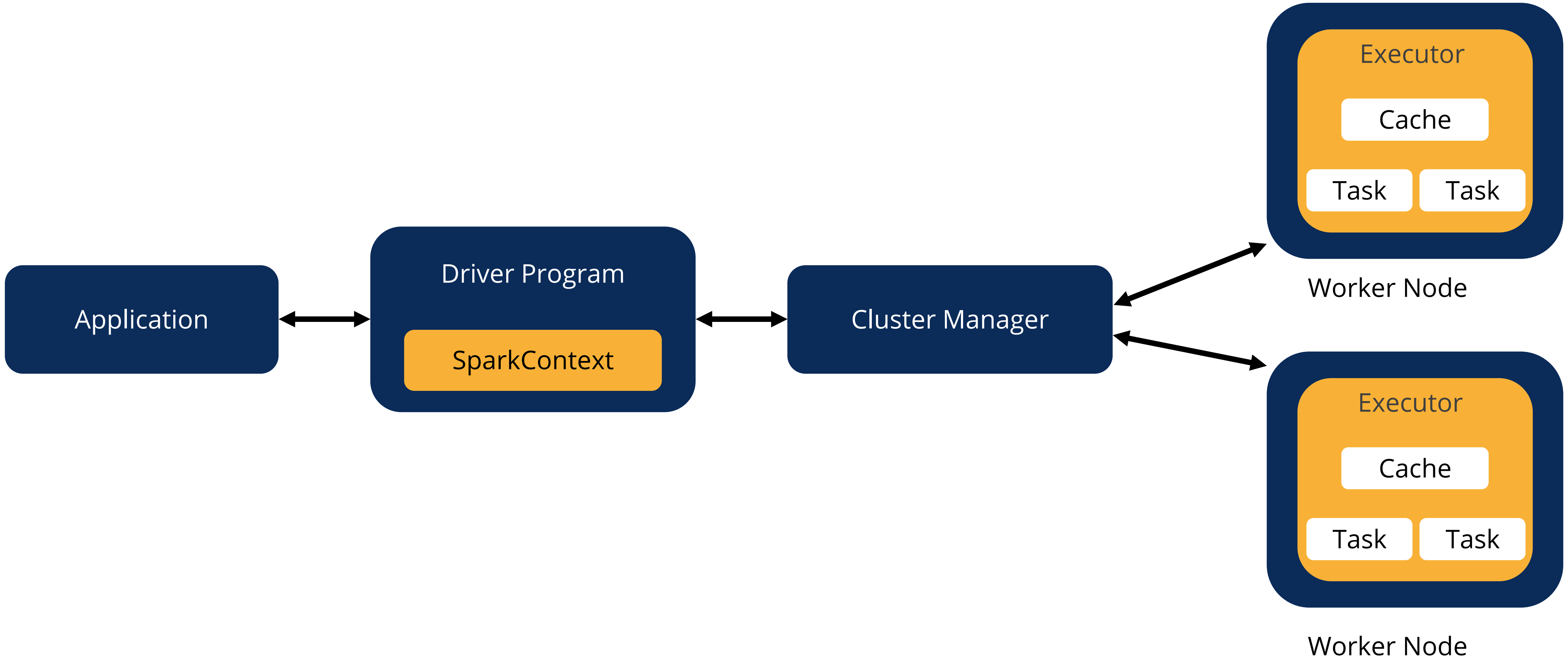
DataFrame DSL (domain specific language)

DataFrame DSL is a collection of rows with a schema, which is a result of executing structured query.

Spark Session and SparkContext

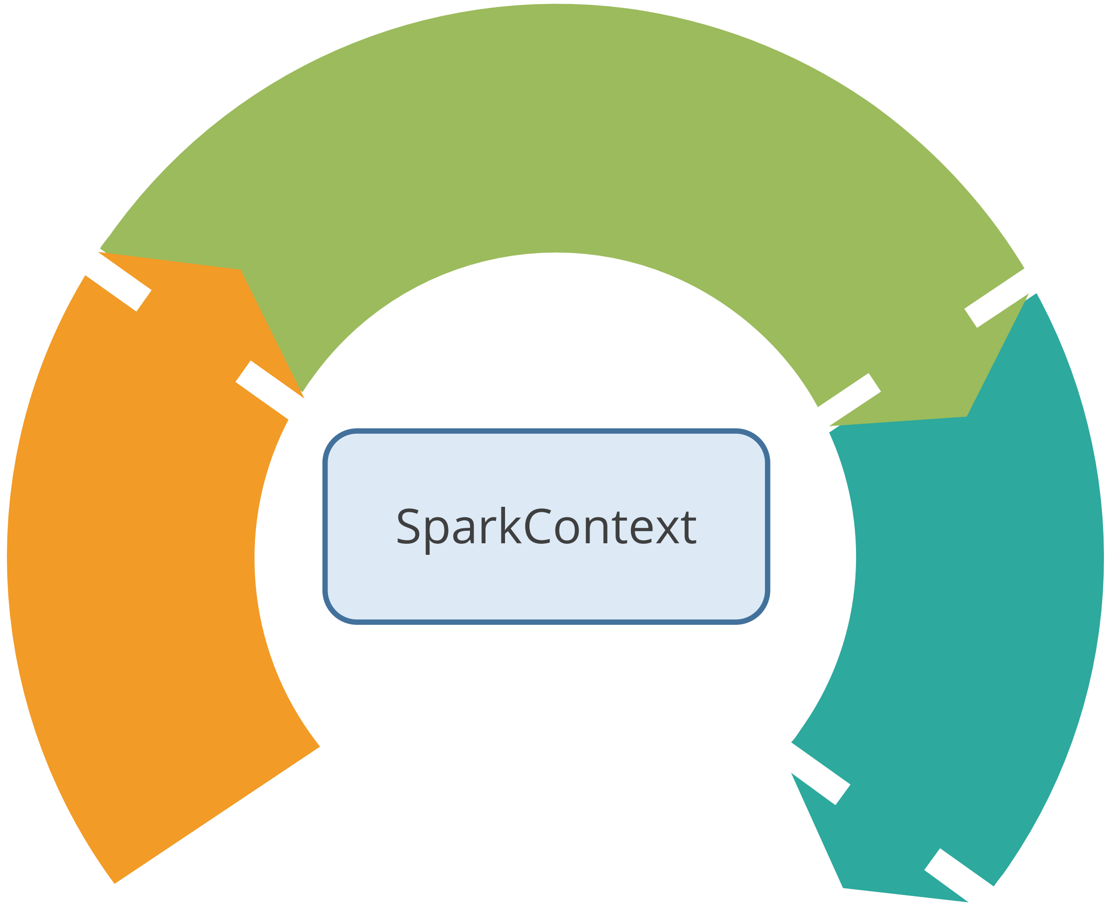
SparkContext

It is the main entry point to Spark functionality, which provides internal services and establishes a connection to a Spark execution environment.



SparkContext

It can be used to create RDDs, accumulators, and broadcast variables to access Spark services and to run jobs till SparkContext is terminated.



It needs separate contexts to be created in order to use APIs of SQL, Hive, and Streaming.

It deals with the versions prior to 2.0.0.

SQLContext in Spark SQL

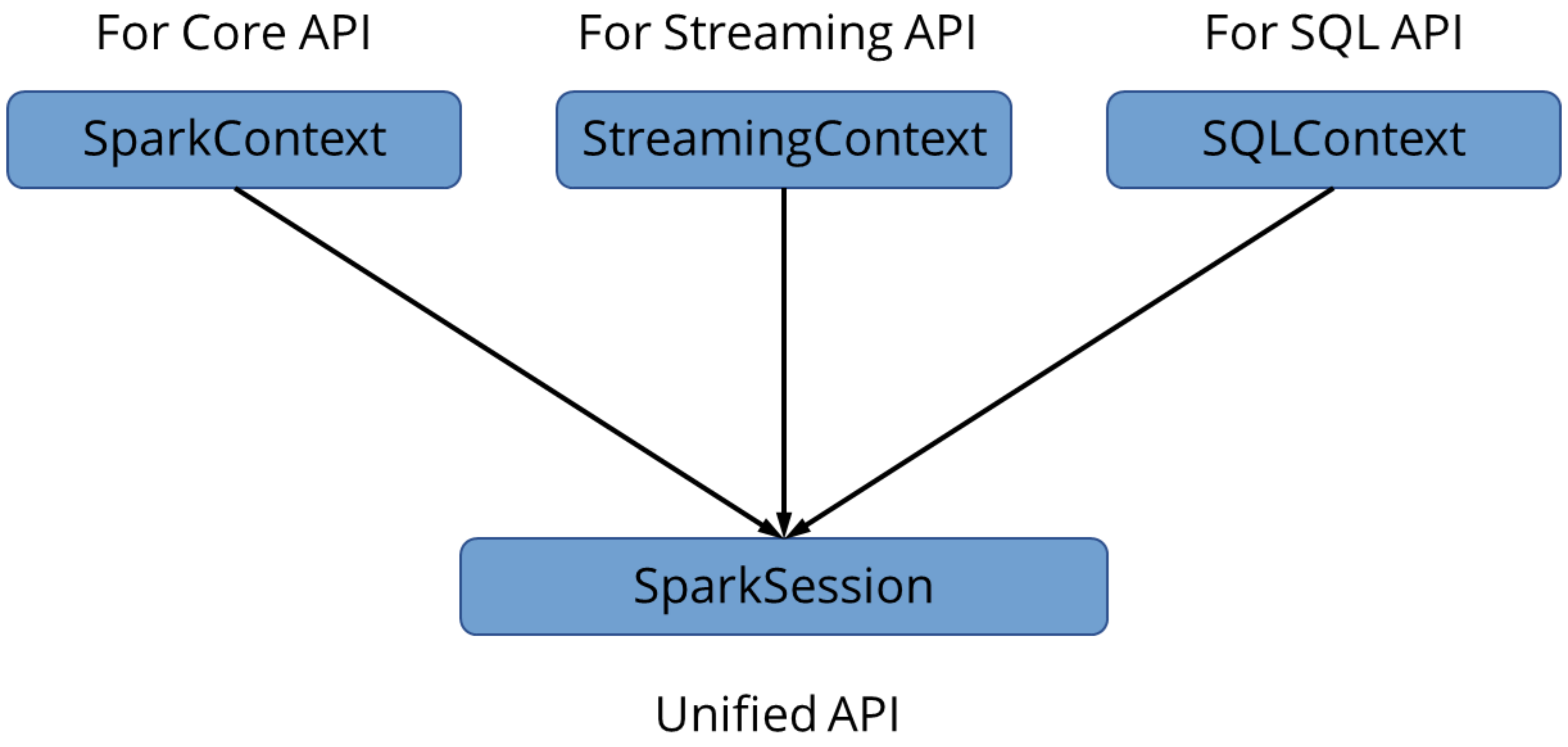
It is the main entry point for DataFrame and SQL functionality. It is used to create DataFrames, register DataFrames as tables, execute SQL over tables, cache tables, and read Parquet files.

Class pyspark.sql.SQLContext(sparkContext, sqlContext=None)

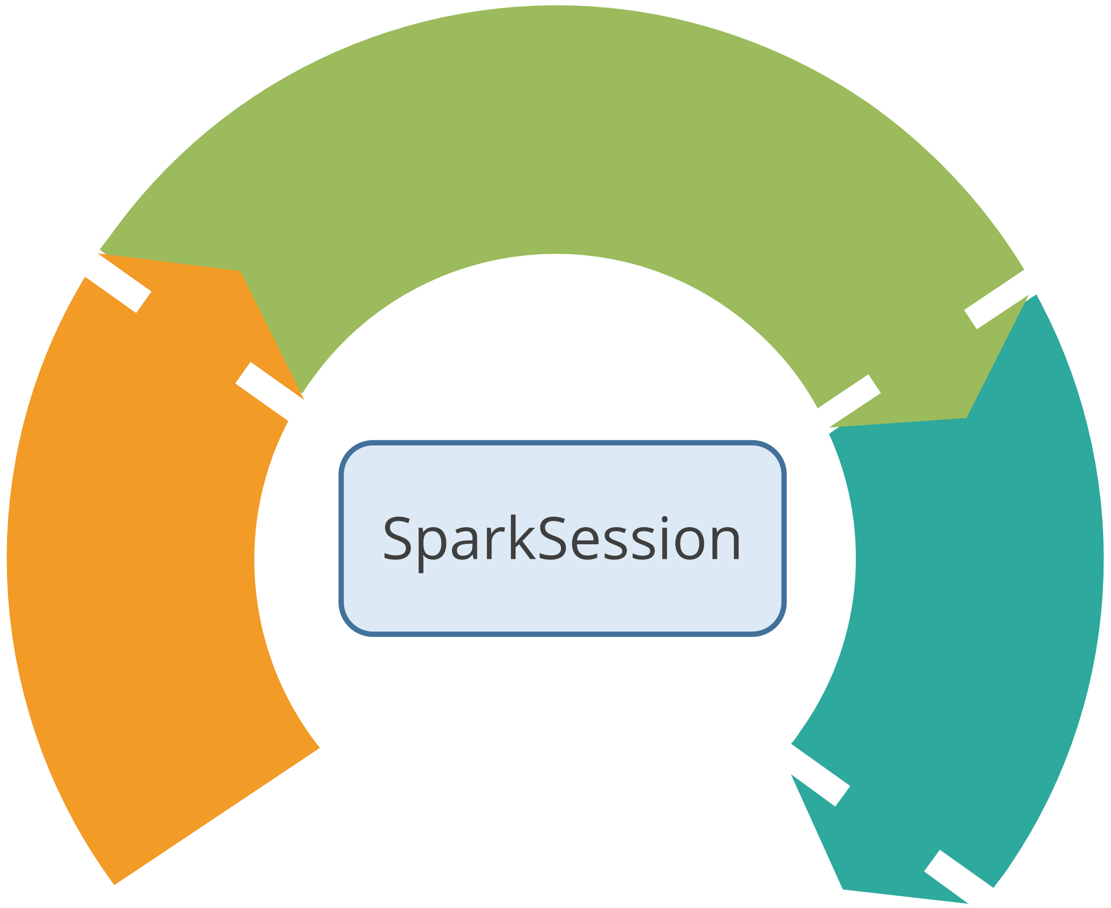
```
Sample Code:
>>> l = [('Alice', 1)]
>>> sqlContext.createDataFrame(l).collect()
Output:[Row(_1=u'Alice', _2=1)]

>>> sqlContext.createDataFrame(l, ['name', 'age']).collect()
Output:[Row(name=u'Alice', age=1)]
```

SparkSession

It provides a single point of entry to interact with the underlying Spark functionality and allows one to program Spark with DataFrame and Dataset APIs. All the functionalities available in SparkContext are also available in SparkSession.



SparkSession



Deals with all versions after 2.0.0

Configures Spark run-time configuration properties once the SparkSession is instantiated

Includes all the APIs eliminating the need to create separate contexts

Unified SparkSession

The SparkSession class is the starting point for all Spark functionality. To create a basic SparkSession, SparkSession.builder is used:

class pyspark.sql.SparkSession(sparkContext,jsparkSession=None)

```
//SparkSession uses the following builder pattern.

Code Snippet:
from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName("Demo")\
    .config("config.name", "config.value")\
    .getOrCreate()
```

SchemaRDDs

SchemaRDD is a collection of row objects along with a schema. A schema defines the data types of each column in the row. A SchemaRDD can be created from:

- Existing RDD
- JSON dataset
- Parquet File
- Running HiveQL on Hive Database

SchemaRDDs Instance Methods

saveAsParquetFile(self, path)

Code Snippet:

```
>>> import tempfile, shutil
>>> parquetFile = tempfile.mkdtemp()
>>> shutil.rmtree(parquetFile)
>>> srdd = sqlCtx.inferSchema(rdd)
>>> srdd.saveAsParquetFile(parquetFile)
```

registerTempTable(self, name)

Code Snippet:

```
>>> srdd = sqlCtx.inferSchema(rdd)
>>> srdd.registerTempTable("
```

User-Defined Functions

User-Defined Functions (UDFs)

User-Defined Functions

Allows to register a custom function and use the same within SQL

Well-created UDFs avoid performance issues

Provides advanced functionality that is not present in the in-built functions

Supports existing Apache Hive User-Defined functions

Registering a function as UDF

Create a function to convert a string to uppercase

Register function as UDF

Code Snippet:

```
def to_uppercase(str):
    resStr=""
    arr = str.split(" ")
    for x in arr:
       resStr= resStr + x[0:1].upper() +
x[1:len(x)] + " "
    return resStr
```

spark.udf.register(“to_upper", uppercase, StringType())

UDF with DataFrame

Create a function to convert a string to uppercase

Register function using UDF with DataFrame

Code Snippet:

```
def to_uppercase(str):
    return resStr.upper()
```

Code Snippet:

```
df.select(col(“col1"), \

convertUDF(col(“col2")).alias(“uppercase_value
"))\
   .show(truncate=False)
```

User-Defined Aggregate Functions

User-Defined Aggregate Functions

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

- Spark SQL allows users to define custom aggregation functions called User-Defined Aggregate Functions or UDAFs.
- UDAFs are very useful when performing aggregations across groups or columns.

User-Defined Aggregate Functions

To create a custom UDAF, it is required to extend UserDefinedAggregateFunction.

Syntax:

```
class GeometricMean extends UserDefinedAggregateFunction
```

Methods

Initialize

On a given node, this method is called once for each group.

Update

This method is called once for each group on a given node.
Spark will call update for each input record in a given group.

Merge

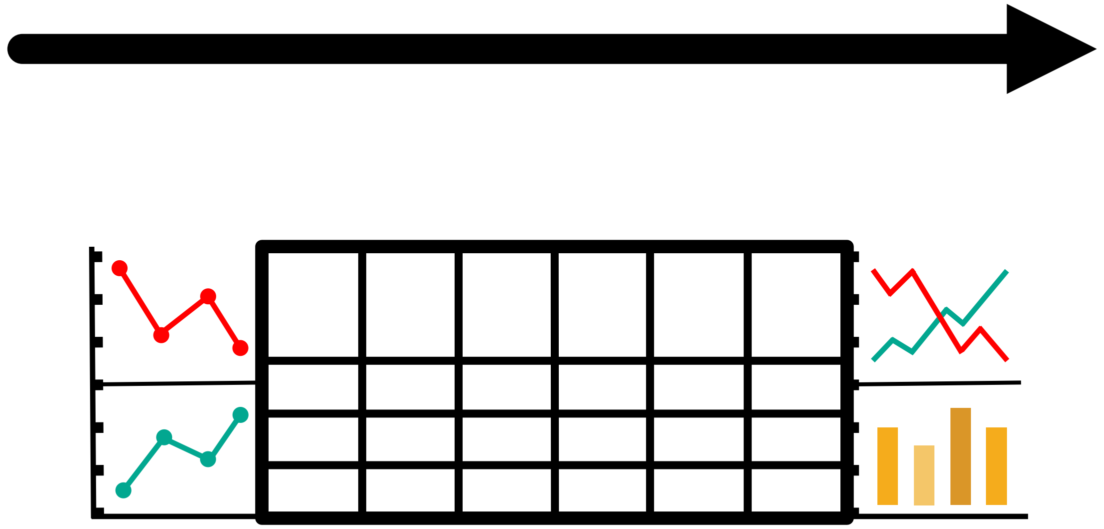
Spark computes partial aggregate results and combines them if the function supports partial aggregates

Evaluate

Spark will call evaluate to get the final result when all the entries for a group are exhausted.

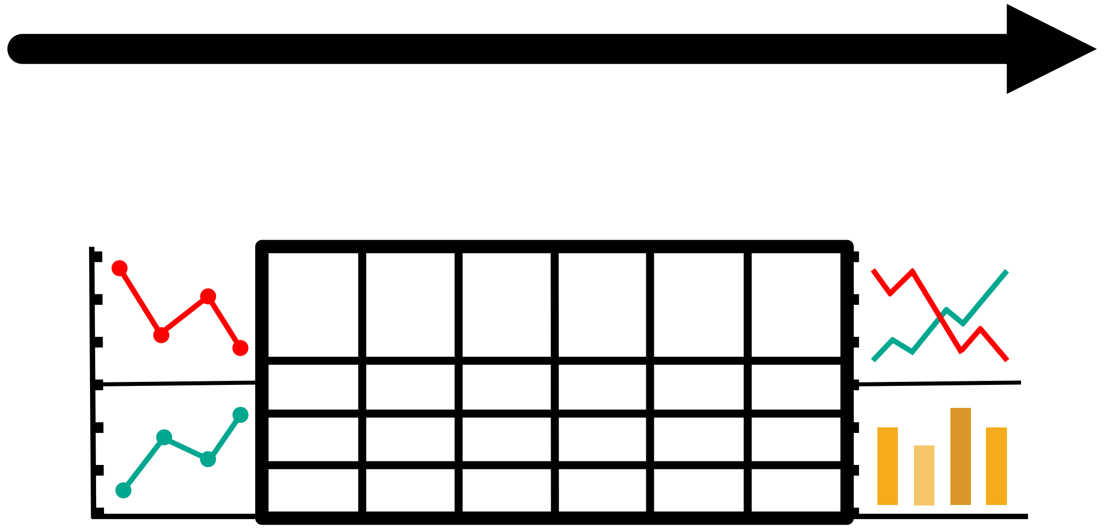
Apache Spark DataFrames

DataFrames



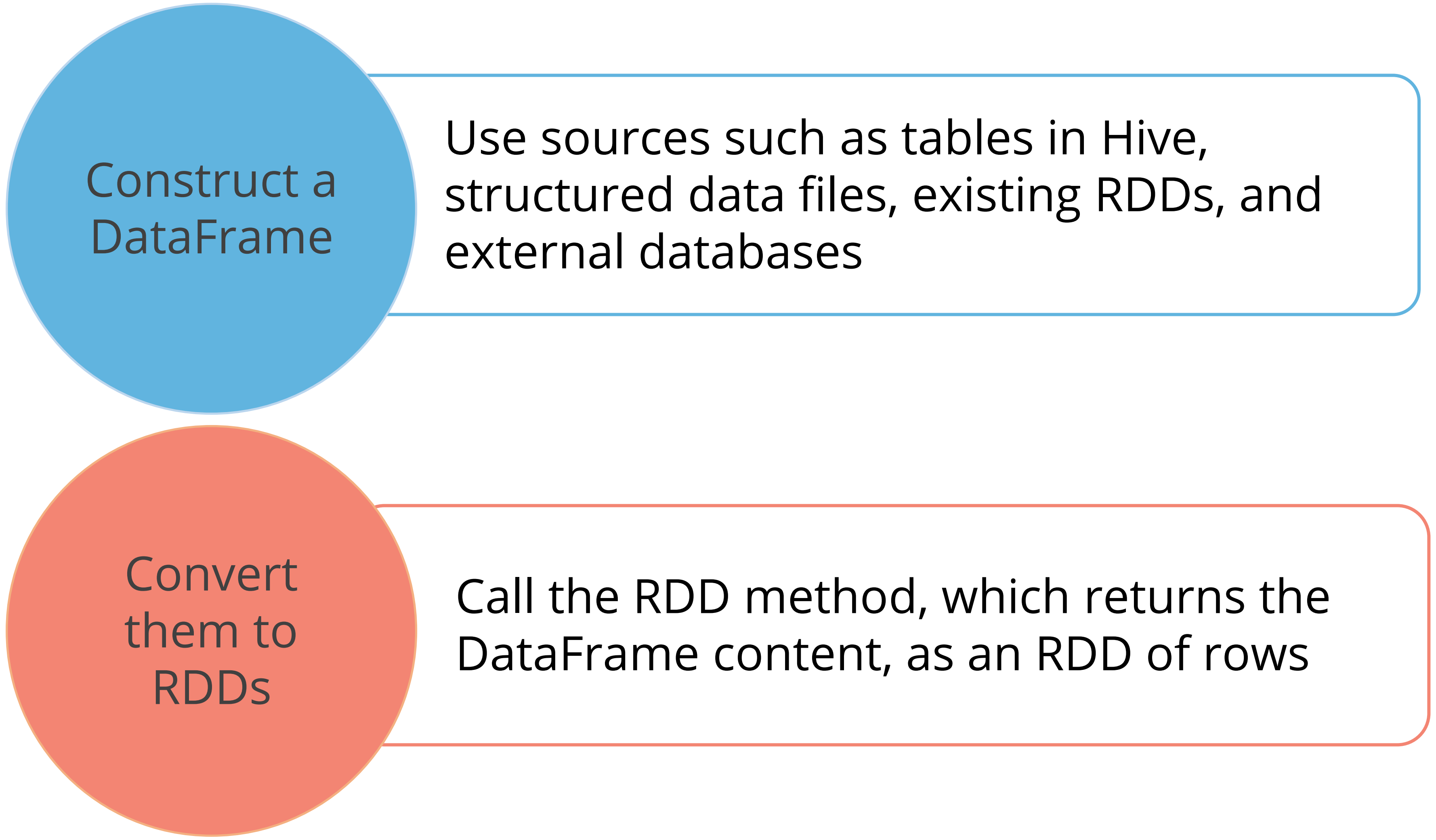
- DataFrames represent a distributed collection of data in which data is organized into named columns.
- It was introduced in Spark 1.3 as an extension to RDDs.
- It is a used for representing structured data in Spark.

DataFrames



- It seems like a relational table in RDBMS but with richer optimizations under the hood.
- Unlike RDDs, DataFrame keeps track of its schema.
- DataFrame = RDD + Schema (SchemaRDD)
- It supports various relational operations that lead to more optimized execution like Catalyst Optimizer.

DataFrames



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, which returns the DataFrame content, as an RDD of rows

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

Creating DataFrames

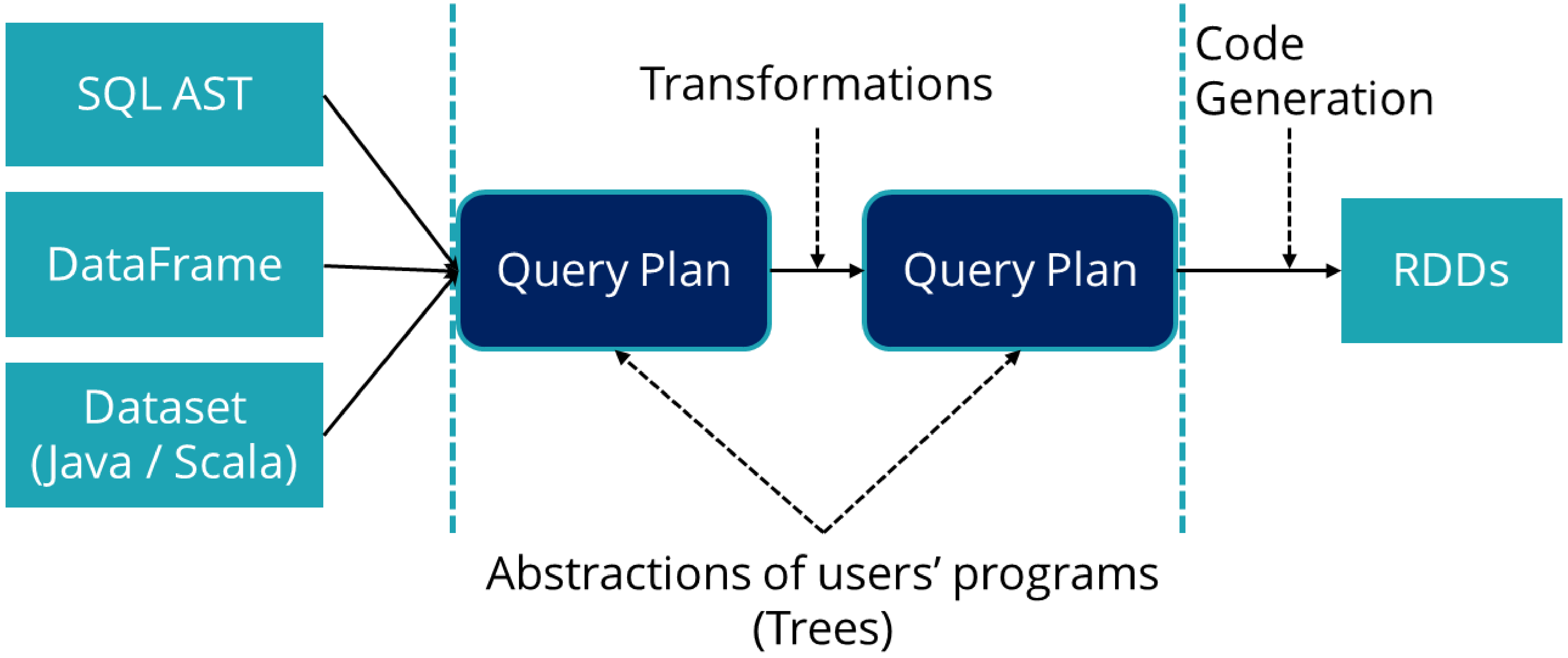
DataFrames can be created:

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

Spark DataFrames: Catalyst Optimizer

Catalyst Optimizer

The Catalyst optimizer is a component of Apache Spark, which optimizes structural queries written in SQL, DataFrame, or Dataset APIs.



Catalyst Optimizer

Catalyst Optimizer

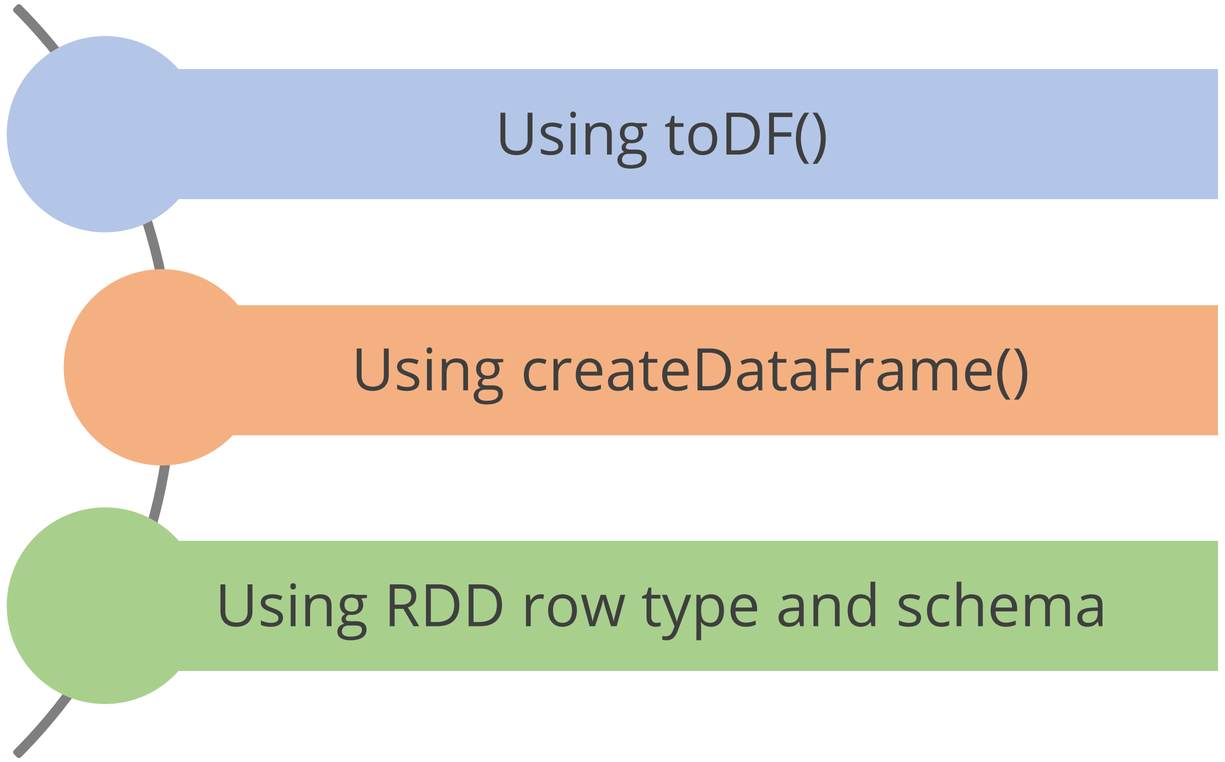
Spark SQL

- It can reduce the runtime of programs and save costs.
- It uses multiple techniques such as filtering and indexes.
- It ensures that the data source joins are performed in the most efficient order.

Interoperating with RDDs

Interoperating with RDDs

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



Using toDF()

Using createDataFrame()

Using RDD row type and schema

Create PySpark RDDs

Example:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Simplilearn’).getOrCreate()

emp = [(“Andrew",10),("Aiden",20),(“Michael",30),(“Stephan",40)]

rdd = spark.sparkContext.parallelize(emp)
for emp in rdd.collect():
    print(emp)
```

Output:

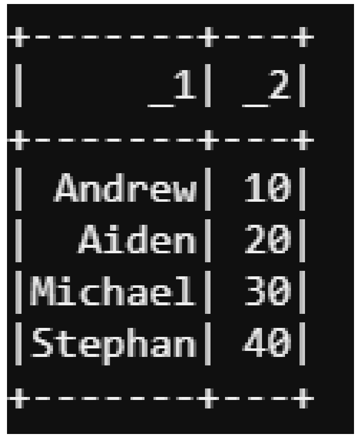
```
(‘Andrew”, 10)
(‘Aiden”, 20)
(‘Michael”, 30)
(‘Stephan”,40)
```

Convert PySpark RDDs to DataFrame Using toDF()

Example:

```
empDF = rdd.toDF()

empDF.show()
```

DataFrame

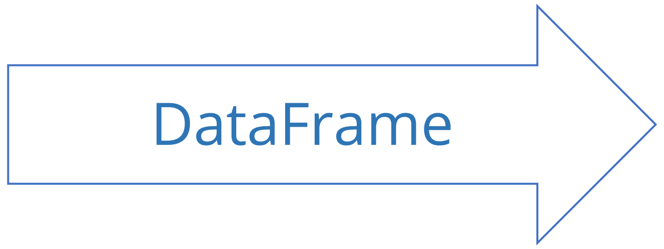


```
+-------+---+
|     _1| _2|
+-------+---+
| Andrew| 10|
|  Aiden| 20|
|Michael| 30|
|Stephan| 40|
+-------+---+
```

Defining Logical Column Names

Example:

```
rdd.toDF(“Name”,”Age”)

rdd.show()
```



DataFrame

Output:

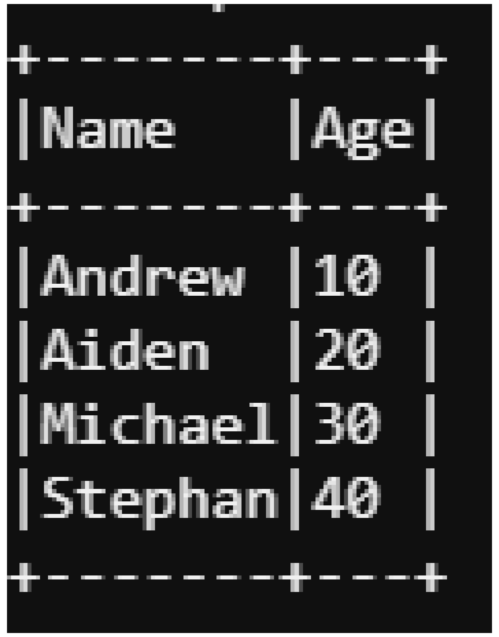
```
+-------+-----+
  Name    Age
 Andrew    10
 Aiden     20
 Michael   30
 Stephan   40
+-------+-----+
```

Using createDataFrame

Example:

```
empDF = spark.createDataFrame(rdd, schema = ["Name",
"Age"])

empDF.show(truncate=False)
```

DataFrame



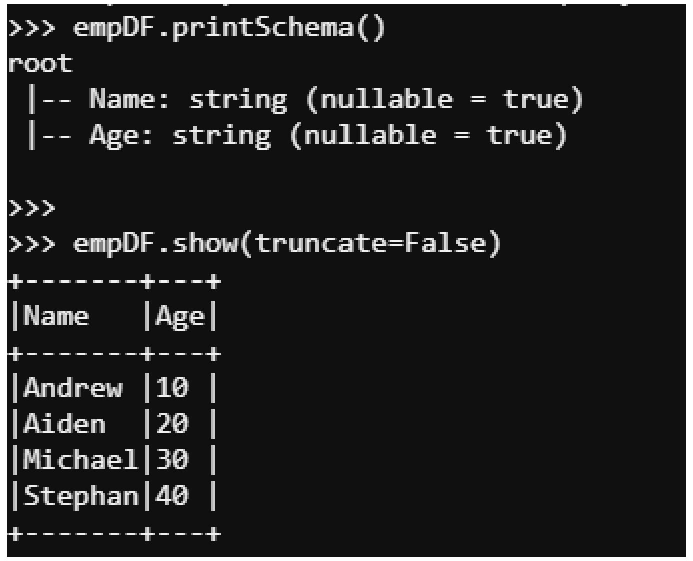
```
+-------+---+
|Name   |Age|
+-------+---+
|Andrew |10 |
|Aiden  |20 |
|Michael|30 |
|Stephan|40 |
+-------+---+
```

Using createDataFrame with StructType

Example:

```
from pyspark.sql.types import StructType,StructField,
StringType

empSchema = StructType([
    StructField('Name', StringType(), True),
    StructField('Age', StringType(), True)
])

empDF = spark.createDataFrame(rdd, schema = empSchema)

empDF.printSchema()

empDF.show(truncate=False)
```

DataFrame



```
>>> empDF.printSchema()
root
 |-- Name: string (nullable = true)
 |-- Age: string (nullable = true)

>>>
>>> empDF.show(truncate=False)
+-------+---+
|Name   |Age|
+-------+---+
|Andrew |10 |
|Aiden  |20 |
|Michael|30 |
|Stephan|40 |
+-------+---+
```

PySpark DataFrames

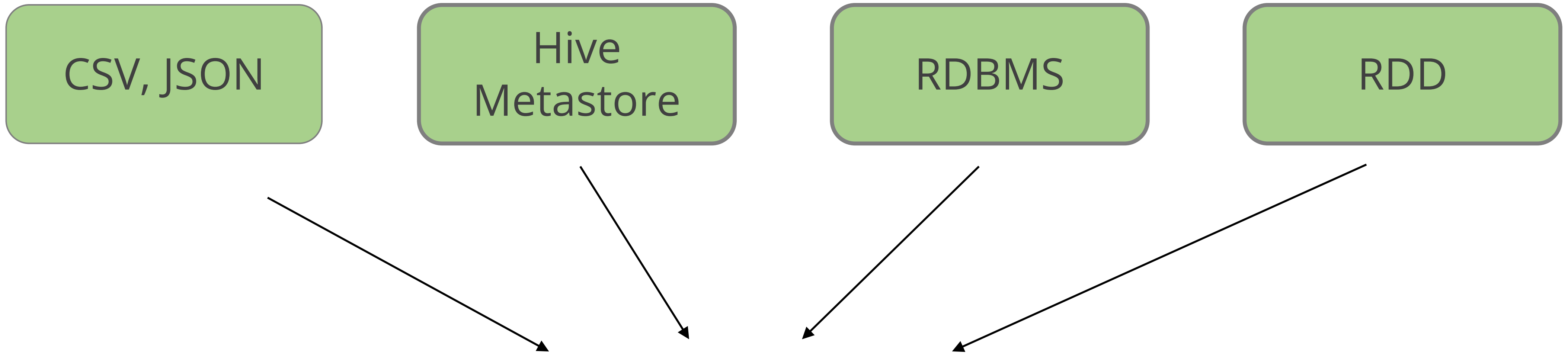
Creating PySpark DataFrames

SparkSession.createDataFrame(data, schema=None, samplingRatio=None, verifySchema=True) can be used to create a dataFrame in Pyspark where:

- Data is an RDD of any kind of SQL data representation.
- Schema can be provided using a list of column names. The type information for each column is inferred from the data.
- When schema is none, PySpark tries to infer the schema from the data itself. In this scenario, data should be an RDD of type row named tuple or dictionary.
- samplingRatio can be *float, optional.*
- verifySchema verifies data types of every row against schema and is enabled by default.

Creating PySpark DataFrames

DataFrame can be created by reading data from multiple file formats.



CSV, JSON

Hive Metastore

RDBMS

RDD

Column 1	Column 2	Column 3

PySpark DataFrame from CSV

Step 1: Create a SparkSession

```
spark = SparkSession \
     .builder \
     .appName("Scenario1“) \
     .getOrCreate()
```

Step 2: Use SparkSession to read a CSV file

```
df = spark.read \ .option("delimiter", "\t") \ .option("inferSchema", "true") \
    .csv("../data-files/customers-tab-delimited") //HDFS path
```

PySpark DataFrame from Parquet

Step 1: Create a SparkSession

```
spark = SparkSession \
    .builder \
    .appName("Scenario1“) \
    .getOrCreate()
```

Step 2: Use SparkSession to read a Parquet file

```
df = spark.read.parquet("../data-files/orders_parquet“) //HDFS path
```

PySpark DataFrame from Avro

Step 1: Create a SparkSession

```
Spark = SparkSession \ .builder \ .appName("Scenario1") \ .config("spark.jars", "../jars/spark-avro_2.12-3.0.0.jar") \ .getOrCreate()
```

Step 2: Use SparkSession to read an Avro file

```
df = spark.read.format("avro") \ .load("../data-files/products_avro") //HDFS path
```

PySpark DataFrame from JSON

Step 1: Create a SparkSession

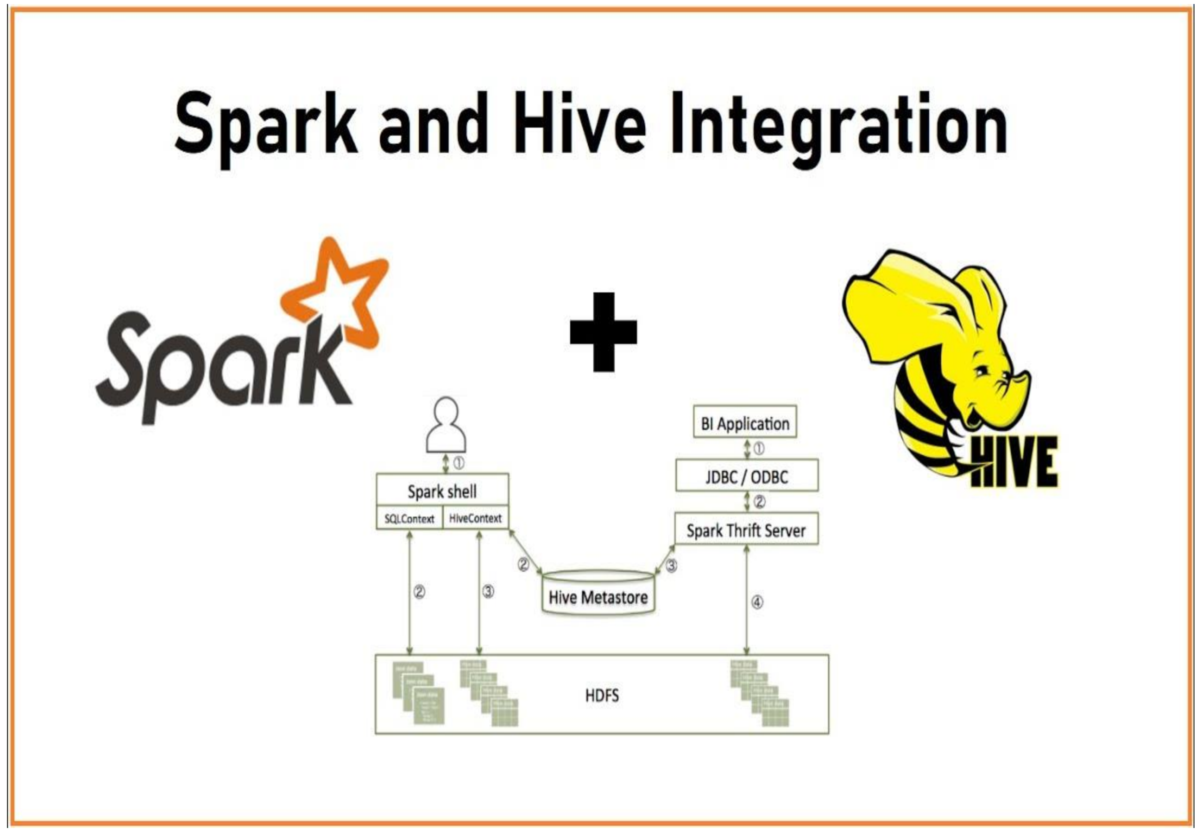
```
spark = SparkSession \ .builder \ .appName("Scenario1") \
    .config("spark.jars", "../jars/spark-avro_2.12-3.0.0.jar") \
    .getOrCreate()
```

Step 2: Use SparkSession to read a JSON file

```
df = spark.read.json("../data-files/categories_json") //HDFS path


df = spark.read.format('org.apache.spark.sql.json') \.load("../data-files/categories_json")
```

Spark-Hive Integration

Spark-Hive Integration



- We need to call integration of hive with spark to leverage the current tables produced in hive.
- To do this, we need the following integration points:
  - HiveContext: It is a superset of the SQLContext which supports writing queries using the HiveQL parser, access to Hive UDFs, and the ability to read data from Hive tables.
  - Metastore URI: It confirms the location of the metastore which can be used by spark. This is the main integration point.
  - Read Data: As the DataFrame is created, we can directly call the table from spark SQL.

Spark-Hive Integration

Libraries required

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
```

Set Hive metastore uri

```
sparkSession = (SparkSession.builder.appName('spark-hive').enableHiveSupport().getOrCreate())
```

Read a table from Hive

```
df_load = sparkSession.sql('SELECT * FROM myTable’)
df_load.show()
```

Write a table into Hive

```
df.write.saveAsTable('myHiveTable')
```

Assisted Practice 16.1: Create DataFrame Using PySpark to Process Records

Duration: 10 Minutes

Problem Scenario: Create a PySpark DataFrame to filter 10 records from a real-world retail business dataset

Objective: To utilize a PySpark DataFrame for reading data from HDFS and filtering only complete orders

Dataset to be Used: order_parquet

Assisted Practice 16.1: Create DataFrame Using PySpark to Process Records

Duration: 10 Minutes

Steps Overview:

Step 1: Download the dataset from the **Reference Materials section** and upload it into the **HDFS** using Hue

Step 2: Login into the **Web desktop** and open the **PySpark shell**

Step 3: Import functions as F from **pyspark.sql**

Step 4: As a PySpark DataFrame, read the **order_parquet** data from **HDFS**

Step 5: Get data for orders marked as COMPLETE

Step 6: Filter the data to display 10 records of COMPLETE orders

Note: The solution to this assisted practice is provided under the Reference Materials section.

Assisted Practice 16.2: UDF with DataFrame

Duration: 10 Minutes

Problem Scenario: Create a DataFrame and define a Python function to convert it into a User Defined Function (UDF)

Objective: In this demonstration, you will create a built-in function using UDF to convert the first letter of every word into uppercase

Steps Overview:

Step 1: Import required packages and create a DataFrame with two columns (S_No, Name)

Step 2: Create a function **convertCase(),** which will convert the first letter of every word into a capital letter

Step 3: Convert a Python function to PySpark UDF

Step 4: Apply the convertUDF function on a DataFrame column as a built-in function

Note: The solution to this assisted practice is provided under the Reference Materials section.

Key Takeaways

- Spark SQL was developed to address the shortcomings of Apache hive.
- User-Defined Functions (UDFs) allow to create a custom function and utilize it in SQL.
- A DataFrame is a distributed collection of data in which the data is grouped into named columns.
- The Catalyst optimizer is an Apache Spark component that optimizes structural queries written in SQL, DataFrame, and Dataset API structural queries.

Knowledge Check

Knowledge Check

1

Which of the following are the components of the Spark project?

A. Spark Core and RDDs

B. Spark SQL

C. Spark Streaming

D. All of the above

Knowledge Check

1

Which of the following are the components of the Spark project?

A. Spark Core and RDDs

B. Spark SQL

C. Spark Streaming

D. All of the above

The correct answer is **D**

Spark Core and RDDs, Spark SQL, and Spark Streaming are some of the components of Spark project.

Knowledge Check

2

“Spark driver runs on the host from where the job is submitted”. Which mode is being talked here?

A. Client Mode

B. Cluster Mode

C. Standalone Mode

D. Mesos

Knowledge Check

2

“Spark driver runs on the host from where the job is submitted”. Which mode is being talked here?

A. Client Mode

B. Cluster Mode

C. Standalone Mode

D. Mesos

The correct answer is **A**

In client mode, the driver is launched directly within the spark-submit process which acts as a client to the cluster. The input and output of the application is attached to the console.

Knowledge Check

3

Which of the following are the supported Cluster Managers?

A. Standalone

B. Apache Mesos

C. Hadoop Yarn

D. All of the above

Knowledge Check

3

Which of the following are the supported Cluster Managers?

A. Standalone

B. Apache Mesos

C. Hadoop Yarn

D. All of the above

The correct answer is **D**

Standalone, Apache Mesos, and Hadoop Yarn are all supported Cluster Managers.

Knowledge Check

4

Suppose you want to run spark job on production server to gather data of all those bike station where count of vehicles is greater than 10 at any time. The job will be submitted to YARN via :

A. Spark shell console

B. SparkR way

C. Jar file using Spark-Submit

D. Executable file via Spark-Shell

Knowledge Check

4

Suppose you want to run spark job on production server to gather data of all those bike station where count of vehicles is greater than 10 at any time. The job will be submitted to YARN via :

A. Spark shell console

B. SparkR way

C. Jar file using Spark-Submit

D. Executable file via Spark-Shell

The correct answer is **C**

The Spark-Submit script in Spark's bin directory is used to launch applications on a cluster. It can use all of Spark’s supported cluster managers through a uniform interface so to configure application for each one is not required.

Knowledge Check
5

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

5

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 **A**

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

Lesson-End Project: Retail Business Analytics

Problem Scenario:

Retail businesses sell items or services to customers for their consumption, usage, or pleasure. They typically sell items and services in-store, but some items are sold online or over the phone, and then shipped to the customer. Examples of retail businesses include clothing, drug, grocery, and convenience stores. As a Spark developer, you are required to find the fraud transactions per month with the number of orders and count.

Objective: The objective is to analyze the order table and perform the analytics using PySpark

Dataset to be Used: part-m-00000

Lesson-End Project: Retail Business Analytics

Steps Overview:

1. Download the dataset from the **Reference Materials section**
2. In HUE, make a folder called **data-files** and within it, create a folder called **orders.** Upload **part-m-00000** into orders
3. Open the PySpark shell in **Web desktop**
4. Read the data in PySpark Shell
5. Create a program to find all fraud transactions where order_status must be equal to SUSPECTED_FRAUD
6. Use YYYY-MM format for order_date
7. The output should contain order_date and count, sorted by order_date in descending order

Thank You