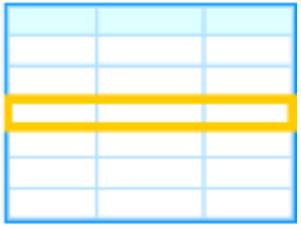
Structured Data

Unstructured Data

Semi-structured Data

Structured Data



RDD and structured data

RDD doesn't know anything about the schema

In the exercise, Article's class is not used by RDD.

RDD pitfalls

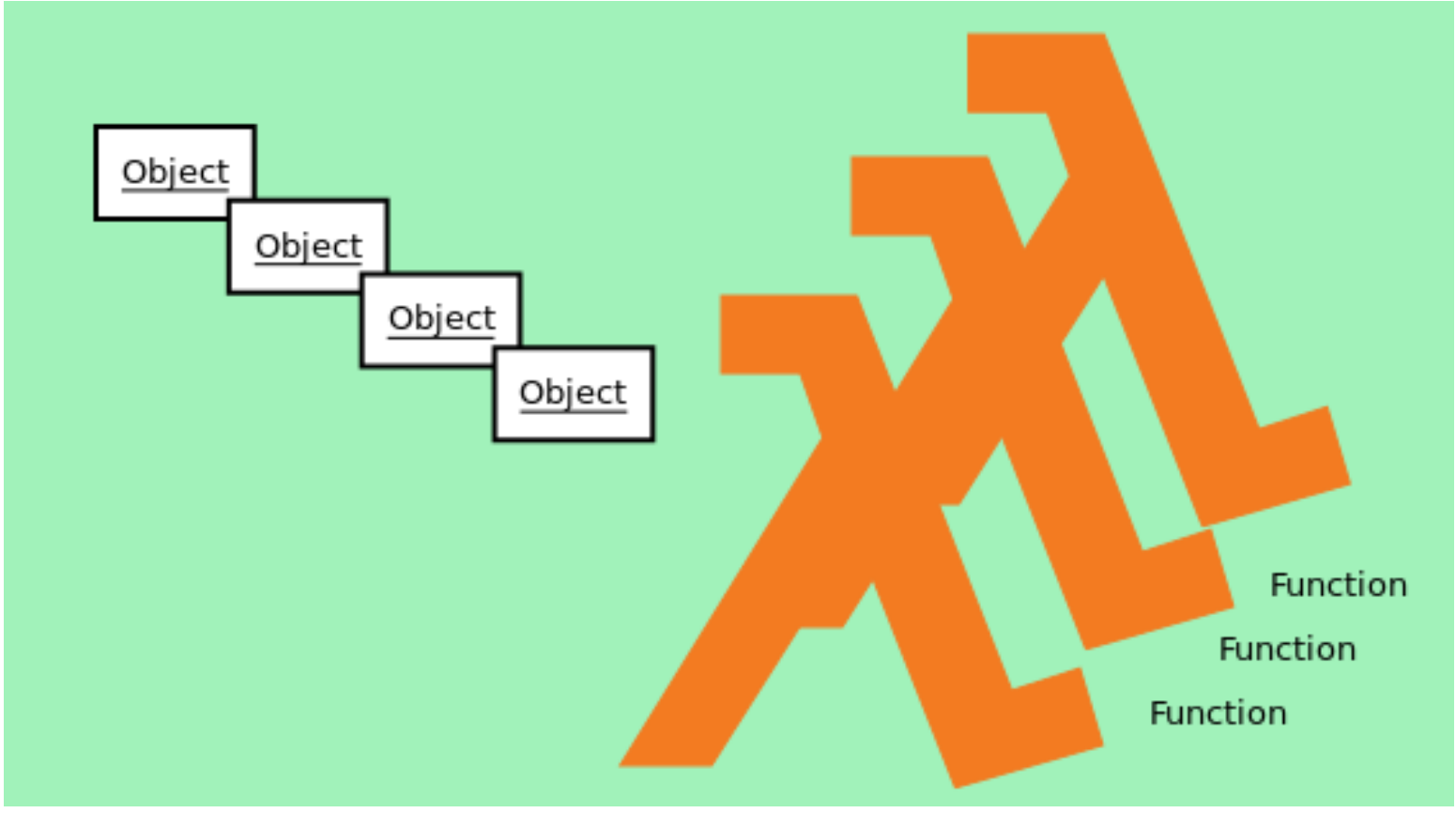
Good

- Abstracts multi files across many nodes
- Handles for us partitioning and failure
- Easier to us than Hadoop's Map/Reduce

Improvement points

- Usability: manipulation of tuples
- Can't optimize

Summary



Introduction

Spark SQL is a library that adds support of structured/unstructured data to Spark.

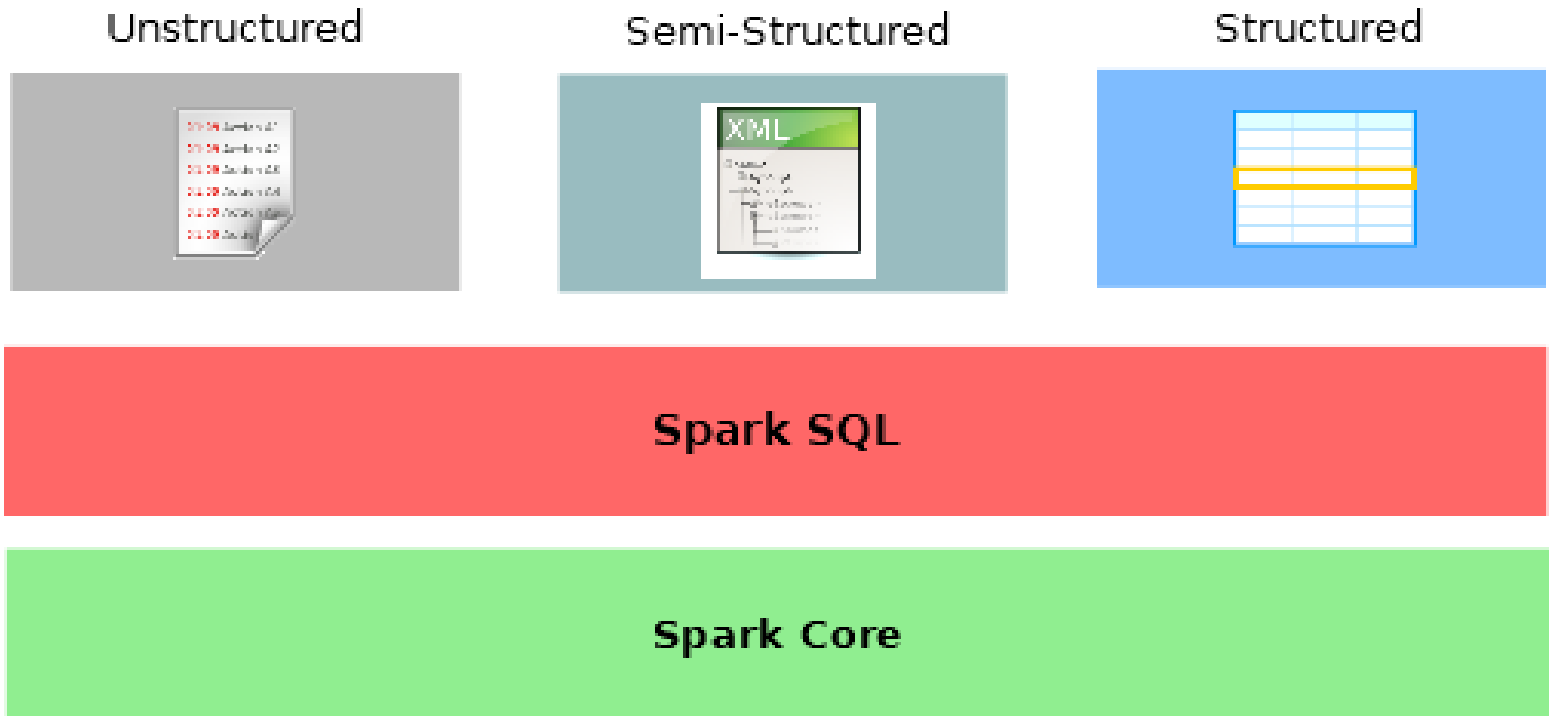
Too different sources of data

- CVS
- LogFile
- XML
- JSON
- BI Tools
- RDBMS...

Good

Would be nice to have a common language to rule them all?

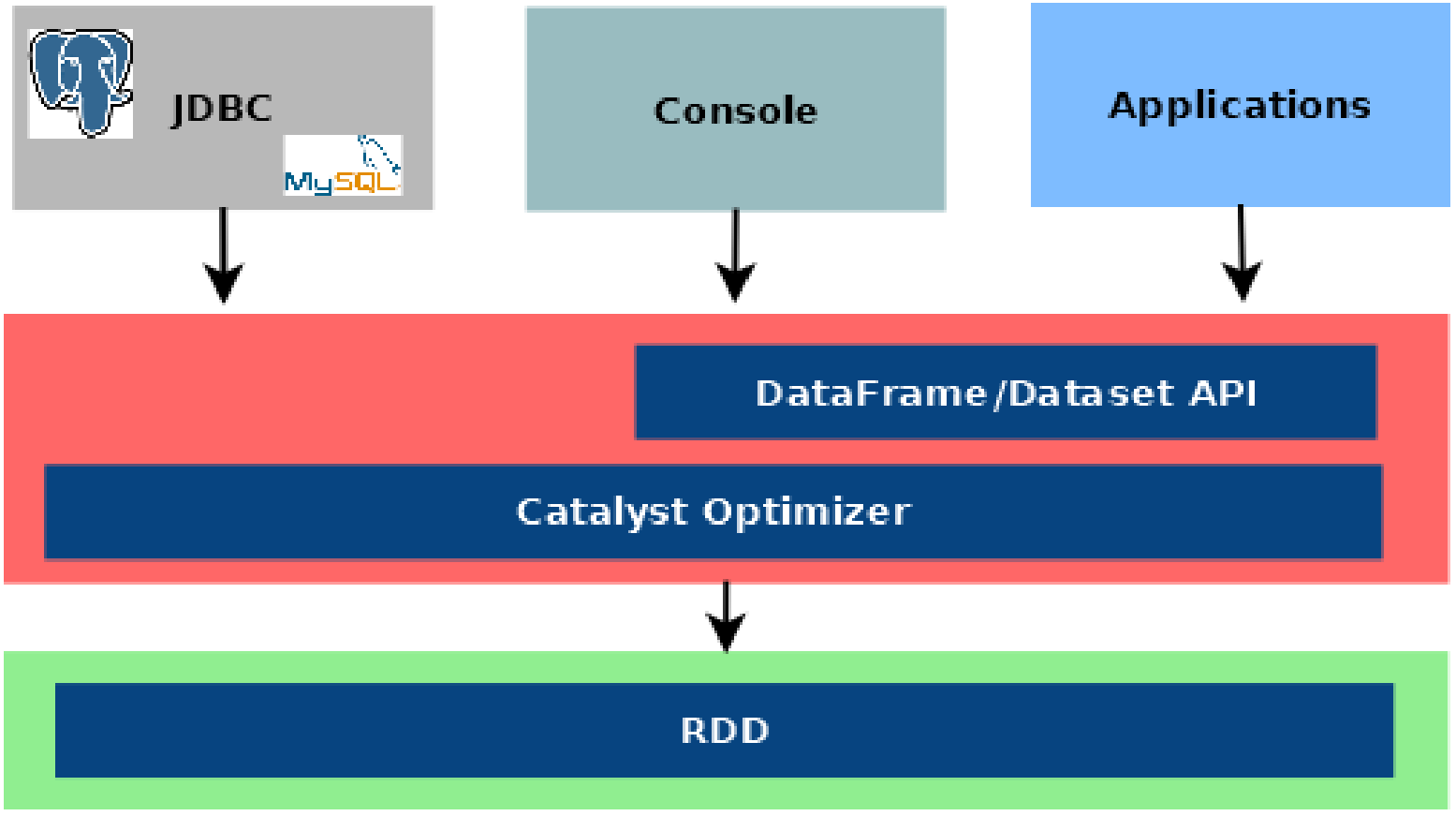
Spark SQL Overview



Spark SQL API

- DataFrame
- SQL literal syntax
- Dataset *(only for Scala and Java)*
- Optmizer (Catalyst)

Spark SQL Overview



Concept

DataFrame is a equivalent of a **table** in a relational database. A DataFrame has the knowledge of the **data schema** like in a relational database.

Table: Soccer score

Team	P	W	D	L	F	A	Pts
Manchester United	6	4	0	2	10	5	12
Celtic	6	3	0	3	8	9	9
Benfica	6	2	1	3	7	8	7
FC Copenhagen	6	2	1	3	5	8	7

Create a DataFrame

- From an existing RDD using schema inference or existing schema
- Reading through a datasource (semi structured/strutured)

Let's take a look at the code!!

Supported datasoures

- JSON
- XML
- Parquet
- JDBC

See this link to get the complete list:
`https://spark.apache.org/docs/2.0.2/api/java/org/apache/spark/sql/DataFrameReader.html`

SQL Literals

View

DataFrame allows the creation of *SQL views*. These views are **temporary**. The current session is their lifetime.

Global View

Global view allows to expand the view lifetime.

SQL Queries

" **SELECT [DISTINCT]** [column names]|[wildcard] **FROM** [keyspace name.]table name [**JOIN** clause table name **ON** join condition] [**WHERE** condition] [**GROUP BY** column name] [**HAVING** conditions] [**ORDER BY** column names [**ASC** | **DSC**]]"

`https://docs.datastax.com/en/archived/datastax_enterprise/4.6/datastax_enterprise/spark/sparkSqlSupportedSyntax.html`

Interesting

Cool feature!

Now you can use SQL to query **any** of the datasources supported by Spark. You do **not** need to **learn** how to traverse XML/JSON and then query a database.

Definition

- Offers a **relational** API over RDD
- Automatically **optimized** using experience from relational databases
- **untyped**! DataFrame are composed of Rows.

Data type

A small set of datatype allows optimization. `https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-DataType.html` The API allows to create your own date. *(UDT as User-Defined Types)*

Example of UDT

Go to console...

DataFrames API: preview

- select
- where
- limit
- orderBy
- groupBy
- join

It looks a lot like SQL, doesn't it?

Preview your data

Method ***show***

```
In [6]: df.show()
+-----+-----+--------------------+------+--------+--------------------+
|count|limit|                next|offset|previous|             results|
+-----+-----+--------------------+------+--------+--------------------+
|36713|   50|http://api.lobbyf...|     0|   false|[[2015-12-05T00:5...|
+-----+-----+--------------------+------+--------+--------------------+
```

Information about the schema

Method ***printSchema***

```
In [7]: df.printSchema()
root
 |-- count: long (nullable = true)
 |-- limit: long (nullable = true)
 |-- next: string (nullable = true)
 |-- offset: long (nullable = true)
 |-- previous: boolean (nullable = true)
 |-- results: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- created_at: string (nullable = true)
 |    |    |-- entity: string (nullable = true)
 |    |    |-- first_name: string (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- last_name: string (nullable = true)
 |    |    |-- name: string (nullable = true)
 |    |    |-- position: string (nullable = true)
 |    |    |-- status: string (nullable = true)
 |    |    |-- title: string (nullable = true)
 |    |    |-- updated_at: string (nullable = true)
 |    |    |-- uri: string (nullable = true)
```

Transformation

- select: projection to only a list of columns
- agg: aggregate data using an expression
- groupBy: group by the specified column using the aggregation
- join: join to another dataframe
- filter/where: filter according to the provided predicate expression

And also: limit, orderBy, sort, union..

Action

- collect: return an array of *Rows*
- count: return the number of occurrences
- first: return the first row
- show: display the top 20 row *nicely*
- take(n): return the first n rows

Join

- inner
- outer
- leftouter
- rightouter
- leftsemi

Cleaning Data

- drop: remove all rows for which any column is null
- drop("all"): remove all rows for which any column is null
- drop(List("col1", "col2"))): remove all rows for which specified column is null

Cleaning Data

- fill(0): replace all null occurrences from a numerical value to 0
- df['col'].fill(value): replace the specified column with the value in parameter
- df['col'].replace(["oldVal", "newVal"])): replace all rows with an old value to the new value

Optimization: Catalyst