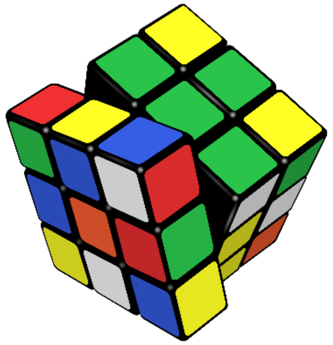


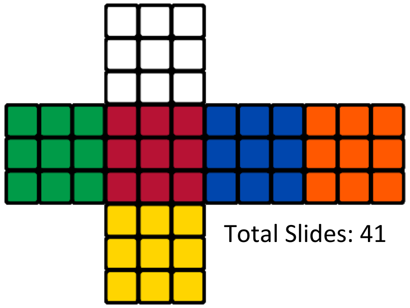
Big Data Engineering For Analytics

Spark Query Language with Python & Scala

Dr Venkat Ramanathan

rvenkat@nus.edu.sg

NUS-ISS


Total Slides: 41

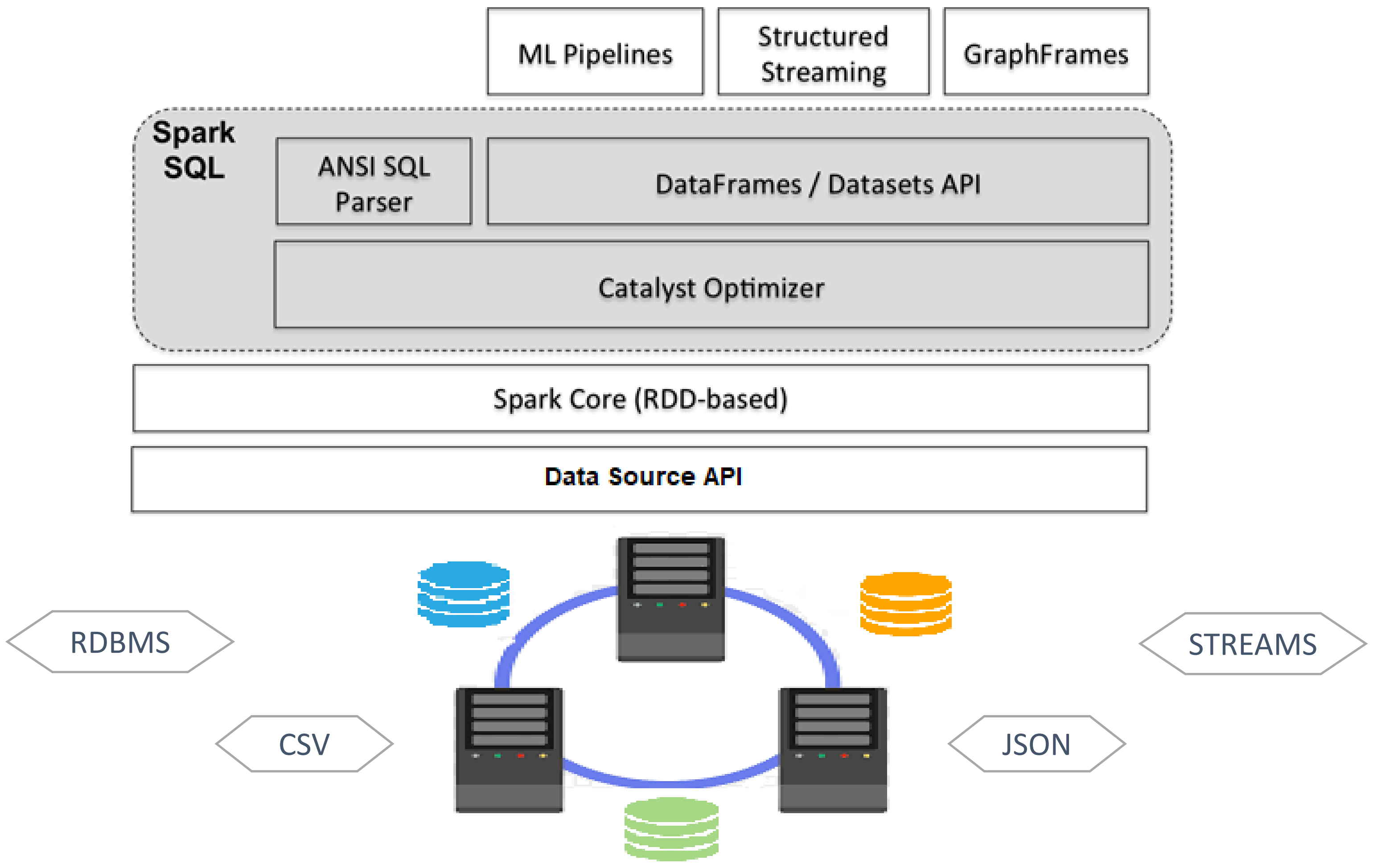
Agenda

- Introduction to Spark SQL
- Spark SQL Architecture
- Spark SQL components
  - ➢Spark Context, Spark Session, SQL Contexts
  - ➢DataFrames
  - ➢Programming with Spark SQL
  - ➢Working with various data sources and formats
- Catalyst optimizer

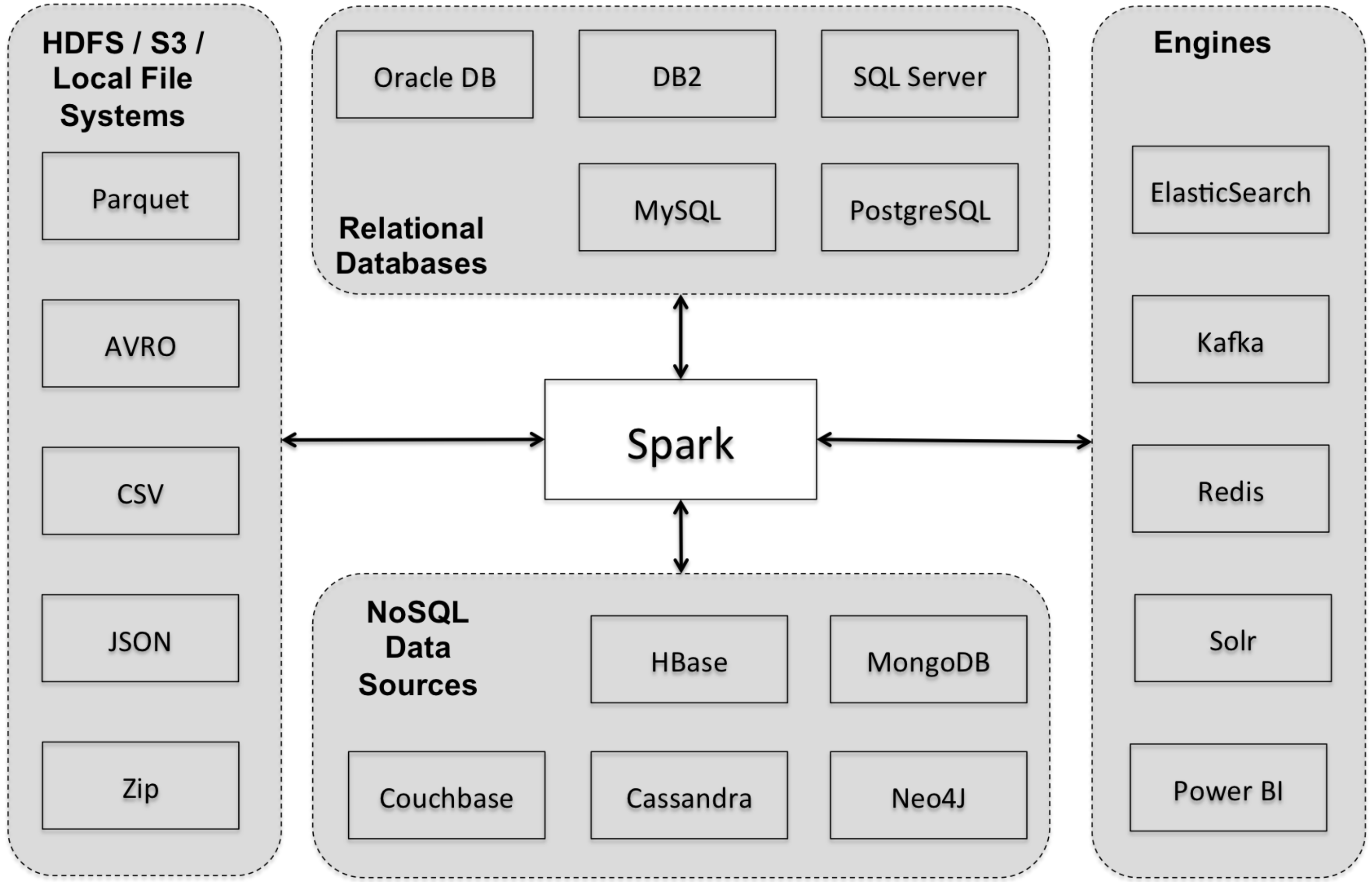
Spark SQL

- What is Spark SQL
  - Spark's interface for working with structured and semi-structured data.
  - Provides the foundation for Spark machine learning applications, streaming applications, graph applications, and many other types of application architectures.
  - Supports relational processing both within Spark programs (on native RDDs) and on external data sources using a programmer-friendly API
  - Flexible in supporting new data sources, including semi-structured data and external databases amenable to query federation
  - Adopts extension with advanced analytics algorithms such as graph processing and machine learning
  - Scalable and provides high performance using established DBMS techniques
  - Built on top of SPARK CORE, Spark SQL is Spark's advanced set of components that provide the foundation for Spark machine learning applications, streaming applications, graph applications, and many other types of application architectures.

Spark SQL Architecture



Data Sources

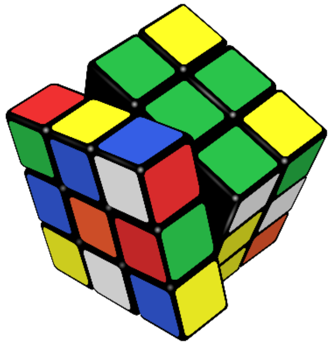


Spark SQL - Genesis

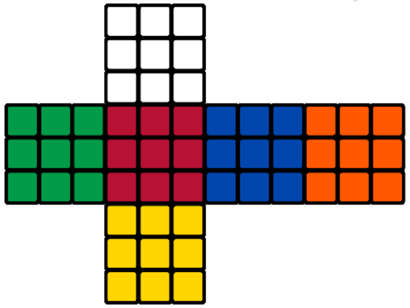
- Why Spark SQL was created?
  - ➢To address the limitations of Hive, like
    - ▪ Hive launches MapReduce jobs internally for executing the ad-hoc queries. MapReduce lags in the performance when it comes to the analysis of medium-sized datasets (10 to 200 GB).
    - ▪ Hive has no resume capability. This means that if the processing dies in the middle of a workflow, you cannot resume from where it got stuck.
    - ▪ Hive cannot drop encrypted databases in cascade when trash is enabled and leads to an execution error. To overcome this, users have to use Purge option to skip trash instead of drop.

Spark SQL

- What does Spark SQL provide?
  - A SQL Engine and command line interface
  - DataFrame API – a library for working with data as tables
    - Defines DataFrames containing Rows and Columns
    - It provides a DataFrame abstraction in Python, Java, and Scala to simplify working with structured datasets. DataFrames are similar to tables in a relational database.
    - It can read and write data in a variety of structured formats (e.g., JSON, Hive Tables, and Parquet).
    - It lets you query the data using SQL, both inside a Spark program and from external tools that connect to Spark SQL through standard database connectors (JDBC/ ODBC), such as business intelligence tools like Tableau.
  - Catalyst Optimizer – an extensible optimization framework



Big Data Engineering For Analytics



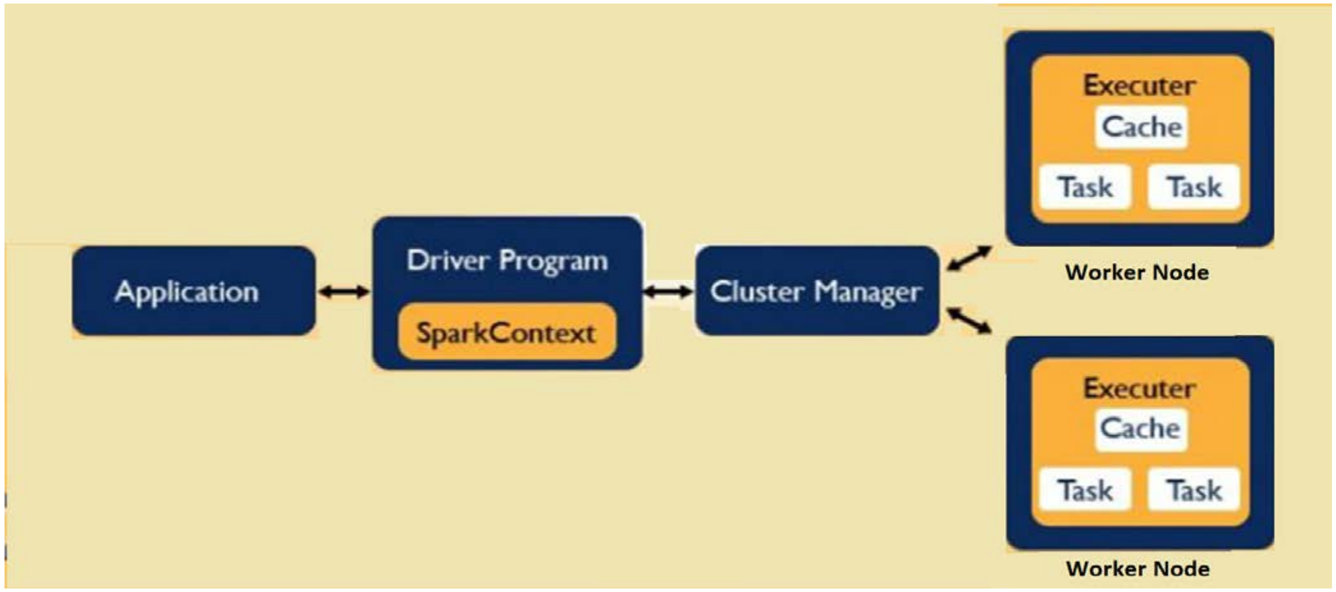
Spark SQL APIs & Objects

Spark Configuration

- The **SparkConf** object sets the configuration for the Spark Application.
- The two important methods in the object are:
  - setAppName
    - Sets the application name for reference, if required
  - setMaster
    - The master URL to connect to. Eg:
      - **"local"** to run locally with one thread
      - **"local[4]"** to run locally with 4 cores, or
      - **"spark://master:7077"** to run on a Spark standalone cluster.

SparkContext

- The SparkContext is the entry point of Spark functionality
- It allows your Spark Application to access Spark Cluster with the help of Resource Manager.
  - ➢The resource manager can be one of these three:
    - Spark Standalone, YARN, Apache Mesos.



SQL Context

- The main Spark SQL entry point in v1.x is a SQL Context object
  - SQL context has been included in Spark Context in v2.x
  - Requires a **`SparkContext`**
  - The SQL Context in Spark SQL is similar to Spark Context in core Spark

- There are two implementations
  - **`SQLContext`**
    - basic implementation
  - **`HiveContext`**
    - Reads and writes Hive/HCatalog tables directly
    - Supports full HiveQL language
    - Requires the Spark application be linked with Hive libraries

<u>Note:</u> With Spark 2.x onward, the SQL Context & Hive Context have been encapsulated into Spark Context and the need for explicitly creating SQL Context is not required.

Spark Session

- The **SparkSession** is the entry point into the Structured API.

- In Spark SQL programming context:
  - ➢It can be used to read data from various sources, such as CSV, JSON, JDBC, stream, and so on.
  - ➢It can be used to execute SQL statements, register User Defined Functions (UDFs), and work with Datasets and DataFrames.

Data Frames

- A DataFrame is an immutable, table-like, distributed data organized into rows, where each one consists a set of columns and each column has a name and an associated type.
  - DataFrames can process data that's available in different formats, such as CSV, AVRO, and JSON, or stored in any storage media, such as Hive, HDFS, and RDBMS
  - DataFrames can process data volumes from kilobytes to petabytes
  - Use the Spark-SQL query optimizer to process data in a distributed and optimized manner
  - Support for APIs in multiple languages, including Java, Scala, Python, and R

File: people.json

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

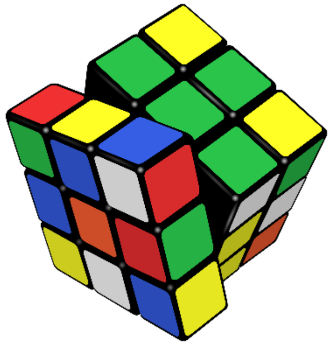
age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

Spark Built-in Data Sources

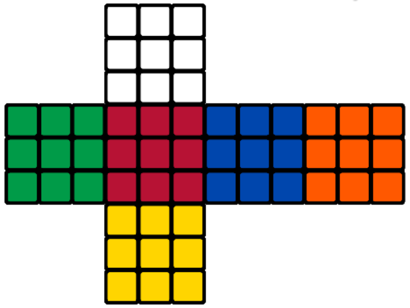
DataFrames can be created

- From an existing **structured data source** (Parquet file, JSON file, etc.)
- From an existing **RDD**
- By performing an **operation** or **query** on another DataFrame
- By **programmatically** defining a schema

Name	Data Format	Comments
Text file	Text	No structure.
CSV	Text	Comma-separated values. This can be used to specify another delimiter. The column name can be referred from the header.
JSON	Text	Popular semi structured format. The column name and data type are inferred automatically.
Parquet	Binary	(Default format.) Popular binary format in the Hadoop community.
ORC	Binary	Another popular binary format in the Hadoop community.
JDBC	Binary	A common format for reading and writing to the RDBMS.



Big Data Engineering For Analytics



Spark Query Language Programming

Spark SQL API Packages

- To run Spark SQL the following packages need to be import
  - **Spark:**

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
```

Scala

  - **PySpark:**

```
from pyspark import SparkContext
from pyspark import SparkConf
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

Python

Class Structure of Scala Revisited

- Using **`object`** in place of **`class`** for these samples.
- The **`main`** method
- Code structure:

Scala

```scala
object MyFirstProgram {
  def main (args: Array[String])
  {
    // Program Codes
  }
}
```

Python Programming Structure

- The **`main`** method.
- Directive to run the main()
- Imports
- Intends
- Multiline codes: use \ for continuing to next like or use **(…)** to encapsulate

Python

```
from xxxx import yyy

def main():
        …
        …
        …

if __name__ == '__main__':
        main()
```

Code Pre-requisites

- Establishing the Configuration, Context & Session:
  - ➢Create a spark configuration (SparkConf)
  - ➢Instantiate a new SparkContext, using the configuration defined
  - ➢Create a Spark Session (instantiate or use SparkSession.builder)

Scala

```
val cnfg = SparkConf().setAppName("CustomerApplication").setMaster("local[2]")
val sc = SparkContext(cnfg)
val spark = SparkSession(sc)
import spark.implicits._
```

Python

```
cnfg = SparkConf().setAppName("CustomerApplication").setMaster("local[2]")
sc = SparkContext(conf=cnfg)
spark = SparkSession(sc)
```

Loading data into Data Frame

- To read data into a data frame you should know the following:
  - ➢The location of the file (or database) where data is stored.
    - Example:

`D:\\Workspace\\Data\\Customer.CSV`	← Windows
`file:/Home/user/Customer.csv`	← Linux
`hdfs://quickstart.cloudera/user/Customer.csv`	← Hadoop
`jdbc:mysql://localhost/videoshop`	← MySQL

  - ➢The format of storage/retrival
    - Example: CSV, Json, jdbc etc.
  - ➢Driver information (if any)

Loading data into Data Frame

- We can use the read or load method

Scala

```
val inputFilePath = "D:\\workspace\\Country.csv"
val df = spark.read
        .option("header", "true")
        .option("inferSchema", "true")
        .csv(inputFilePath)
```

Python

```
inputFilePath = "D:\\workspace\\Country.csv"
df = ( spark.read
        .option("header", "true")
        .option("inferSchema", "true")
        .csv(inputFilePath) )
```

Note: In the above example we have assumed that the first row of the CSV is the header *(hence provided header as true and also have asked the framework to infer the schema which will make the data frame use the first row information for column headers and will examine the data of that column to infer a suitable data type.*

Working with the Data Frame

Big Data Engineering For Analytics

- After loading the data into a DataFrame (df) you may perform various actions.

  - ➢Examining the schema:

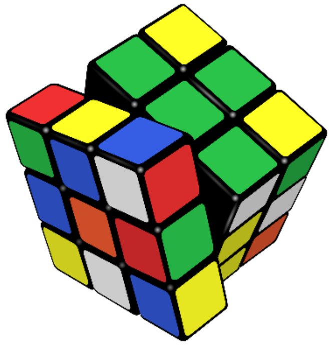
    ```
    df.printSchema()
    ```

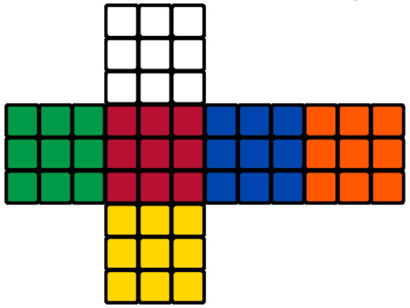
  - ➢Getting the number of records:

    ```
    df.count()
    ```

  - ➢Displaying records:

    ```
    df.show()
    ```





Spark SQL Examples

Live Demo
Refer also to Workshop Document

DataFrame Structured Transformations - 1

Operation	Description
select	This selects one or more columns from an existing set of columns in the DataFrame. A more technical term for select is projection. During the projection process, columns can be transformed and manipulated.
selectExpr	This supports powerful SQL expressions in transforming columns while performing projection.
filter **where**	Both filter and where have the same semantics. where is more relational than filter, and it is similar to the where condition in SQL. They are both used for filtering rows based on the given Boolean conditions.
distinct **dropDuplicates**	This removes duplicate rows from the DataFrame.
sort **orderBy**	This sorts the DataFrame by the provided columns.

DataFrame Structured Transformations - 2

Operation	Description
limit	This returns a new DataFrame by taking the first n rows.
union	This combines two DataFrames and return them as a new DataFrame.
withColumn	This is used to add a new column or replace an existing column in the DataFrame.
withColumnRenamed	This renames an existing column. If a given column name doesn't exist in the schema, then it is a no-op.
drop	This drops one or more columns from a DataFrame. This operation does nothing if a specified given column name doesn't exist.
sample	This randomly selects a set of rows based on the given fraction parameter, an optional seed value, and an optional replacement option.

DataFrame Structured Transformations – 3.

Operation	Description
randomSplit	This splits the DataFrames into one or more DataFrames based on the given weights. It is commonly used to split the master data set into training and test data sets in the machine learning model training process.
join	This joins two DataFrames. Spark supports many types of **joins**.
groupBy	This groups a DataFrame by one or more columns. A common pattern is to perform some kind of aggregation after the **groupBy** operation.
describe	This computes the common statistics about numeric and string columns in the DataFrame. Available statistics are count, mean, stddev, min, max, and arbitrary approximate percentiles.

Join Types

Type	Description
Inner join (aka equi-join)	Returns rows from both datasets when the join expression evaluates to true.
Left outer join	Returns rows from the left dataset even when the join expression evaluates to false.
Right outer join	Returns rows from the right dataset even when the join expression evaluates to false.
Outer join	Returns rows from both datasets even when the join expression evaluates to false.
Left anti join	Returns rows only from the left dataset when the join expression evaluates to false.
Left semi join	Returns rows only from the left dataset when the join expression evaluates to true.
Cross (aka Cartesian)	Returns rows by combining each row from the left dataset with each row in the right dataset. The number of rows will be a product of the size of each dataset.

Aggregation Functions - 1

Operation	Description
count(col)	Returns the number of items per group.
countDistinct(col)	Returns the unique number of items per group.
approx_count_distinct(col)	Returns the approximate number of unique items per group.
min(col)	Returns the minimum value of the given column per group.
max(col)	Returns the maximum value of the given column per group.
sum(col)	Returns the sum of the values in the given column per group.
sumDistinct(col)	Returns the sum of the distinct values of the given column per group.
avg(col)	Returns the average of the values of the given column per group.

Aggregation Functions - 2.

Operation	Description
skewness(col)	Returns the skewness of the distribution of the values of the given column per group.
kurtosis(col)	Returns the kurtosis of the distribution of the values of the given column per group.
variance(col)	Returns the unbiased variance of the values of the given column per group.
stddev(col)	Returns the standard deviation of the values of the given column per group.
collect_list(col)	Returns a collection of values of the given column per group. The returned collection may contain duplicate values.
collect_set(col)	Returns a collection of unique values per group.

Different Ways of Referring to a Column

Way	Example	Description
""	"columName"	This refers to a column as a string type.
col	col("columnName")	The col function returns an instance of the Columnclass.
column	column("columnName")	Similar to col, this function returns an instance of the Column class.
$	$"columnName"	This is a syntactic sugar way of constructing a Columnclass in Scala.
' (tick mark)	'columnName	This is a syntactic sugar way of constructing a Columnclass in Scala by leveraging the Scala symbolic literals feature.

Defining Schema

- If the file does not have a schema (or if you do not wish to use the original header), then a schema can be defined & used to read data.

Scala

```
  val personschema = StructType(Array(
                          StructField("NRIC", IntegerType, true),
                          StructField("Name", StringType, true),
                          StructField("Age", IntegerType, true)    ))
   val df = sqlContext.read.schema(personschema).csv(inputFilePath)
```

Python

```
personschema = ( StructType([
                    StructField("NRIC", IntegerType(), True),
                    StructField("Name", StringType(), True),
                    StructField("Age", IntegerType(), True) ])    )
df1 = (spark.read.schema(schema=personschema).csv(inputFilePath))
```

Spark Scala Types equivalent to SQL

Big Data Engineering For Analytics

Data Type	Scala Type	Python Type
BooleanType	Boolean	bool
ByteType	Byte	int or long
ShortType	Short	int or long
IntegerType	Int	int
LongType	Long	long
FloatType	Float	float
DoubleType	Double	float
StringType	String	str

Data Type	Scala Type	Python Type
DecimalType	java.math.BigDecial	decimal
BinaryType	Array[Byte]	bytearray
TimestampType	java.sql.Timestamp	datetime.datetime
DateType	java.sql.Date	datetime.date
ArrayType	scala.collection.Seq	list, tuple, or array
MapType	scala.collection.Map	dict
StructType	org.apache.spark.sql.Row	list or tuple

Other Supported Data Sources

- You can also use third party data source libraries, such as
  - Cassandra, Redis, HBase, MySQL and more being added all the time

- To work with third party data sources:
  - You may have to download the appropriate driver
  - You will need to add class paths and/or set project references
  - Refer to the product API for driver and configuration details

Caution: Avoid direct access to databases in production environments, which may overload the DB or be interpreted as service attacks. Use Sqoop to import instead.

Supported Data Sources: MySQL Example

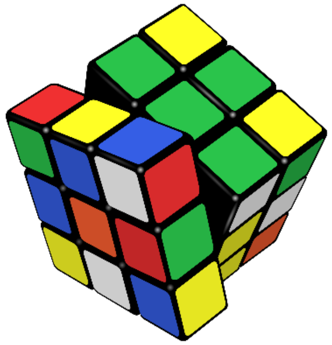
- Loading from a MySQL database

Scala

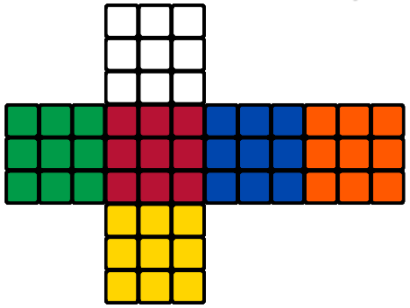
```
val df =  spark.load("jdbc“,
                        Map(url="jdbc:mysql://localhost/videoshop“?
                                   user=“venkat”&password=“P@ssw0rd1”)
                        dbtable=“SomeTableName“)
```

Python

```
cnfg.set("spark.jars","D:\\mysql-connector-java-5.1.49.jar")
...
df = ( spark.read.format("jdbc")
         .options(url="jdbc:mysql://localhost/videoshop",
             driver="com.mysql.jdbc.Driver",
             dbtable=“SomeTableName",
             user="venkat", password="P@ssw0rd1").load())
```



Big Data Engineering For Analytics

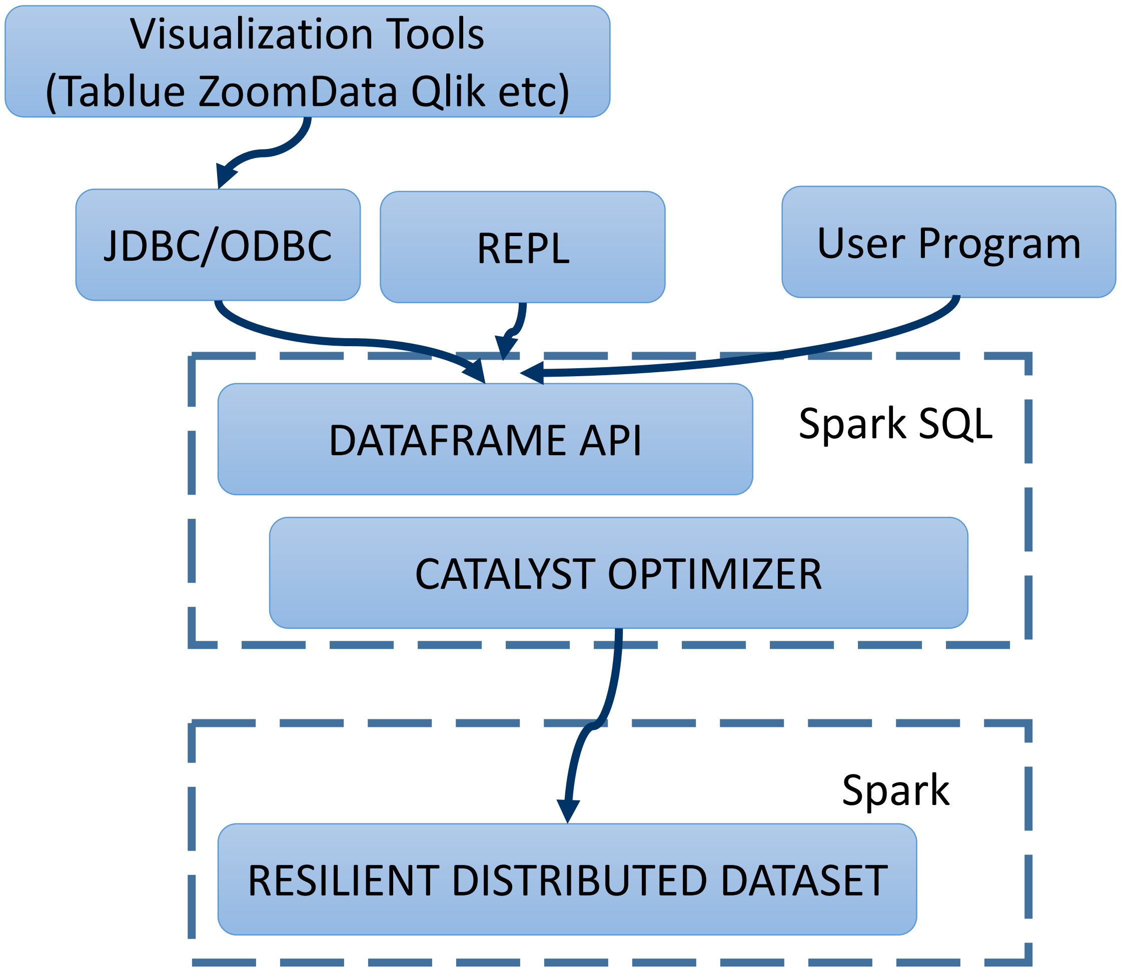


Spark Query Optimization

"Big data is at the foundation of all the megatrends that are happening."

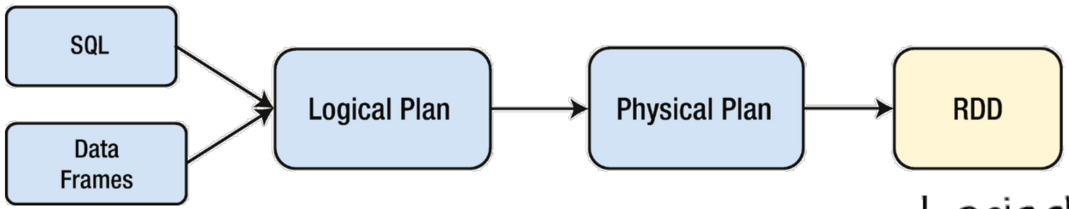
~Chris Lynch

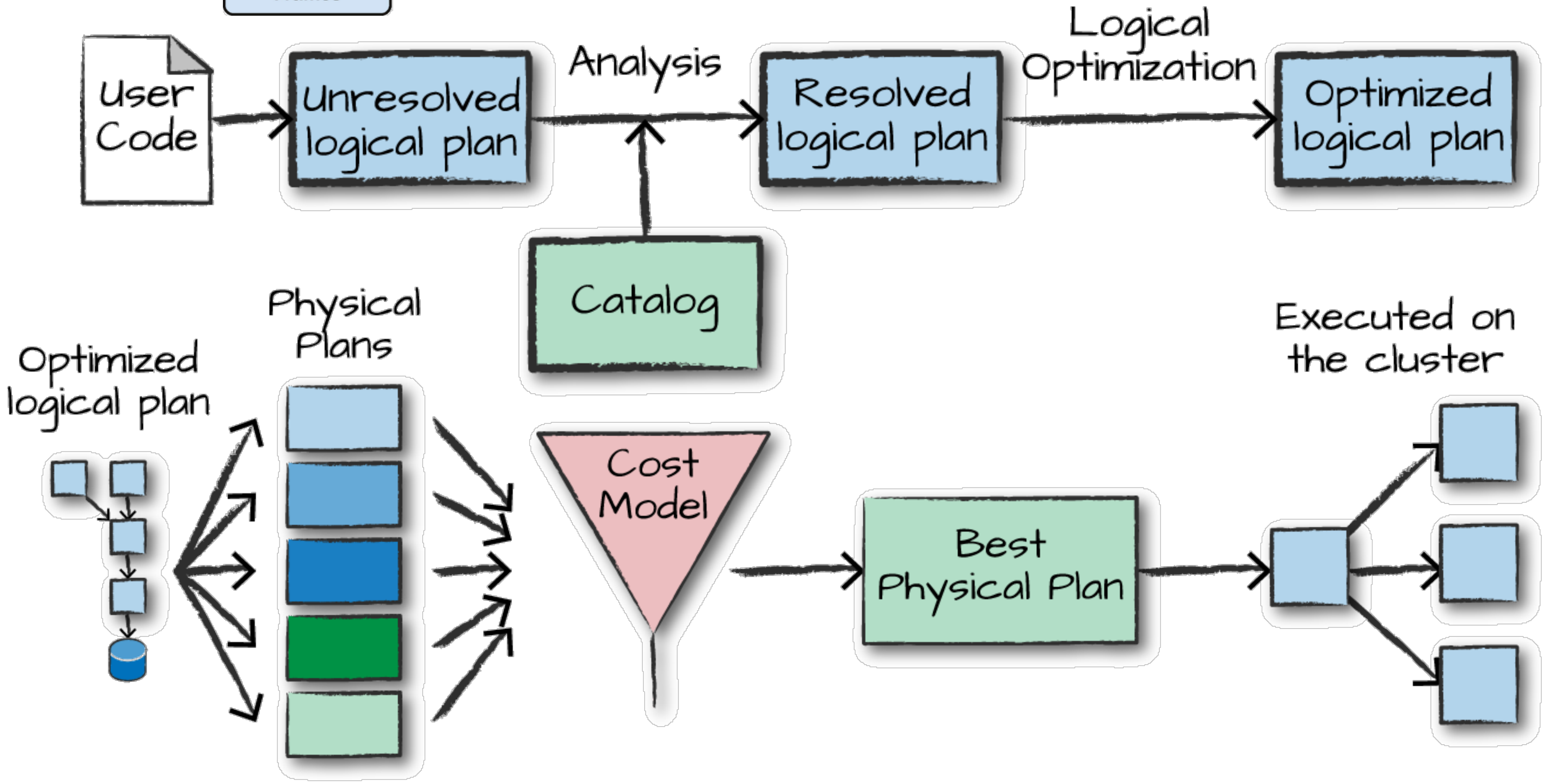
Catalyst Optimizer

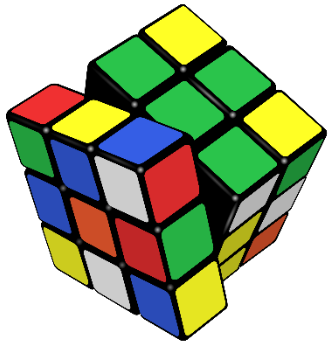


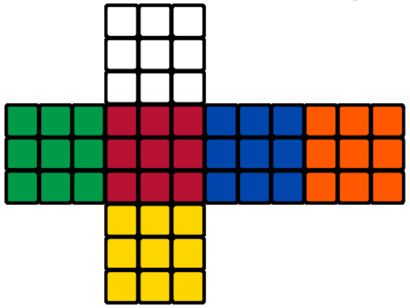
- Catalyst optimizer primarily leverages functional programming constructs of Scala such as pattern matching.
- Catalyst optimizer has two primary goals:
  - Make adding new optimization techniques easy
  - Enable external developers to extend the optimizer
- Catalyst Optimizer Tasks: (1)Analysis (2)Optimization (3)Physical Planning (4)Cost Model Application (5)Physical Plan Selection (6)Optimized Code Generation

Logical and Physical Planning









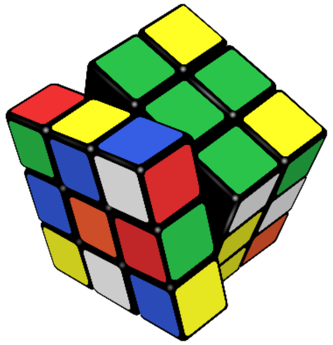
Summary

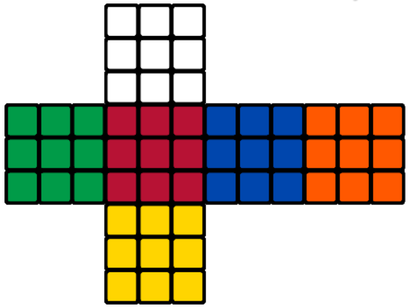
"Without big data analytics, companies are blind and deaf, wandering out onto the web like deer on a freeway."

- Geoffrey Moore, author of Crossing the Chasm

In Essence

- We learnt about Spark SQL or Sparkql
  - ➢Spark SQL is a Spark API for handling structured and semi--structured data
    - Entry point is a SQLContext
    - DataFrames are the key unit of data
    - DataFrames are based on an underlying RDD of Row objects
- We described some of the internals of Spark SQL, including the Catalyst and Project Tungsten-based optimizations.
- We learnt to use Spark SQL to explore structured and semi-structured data





References

“War is 90% information.”

– Napoleon Bonaparte, French military and political leader

Books You May Enjoy. . .

- Spark: The Definitive Guide - Bill Chambers and Matei Zaharia