RDD vs DataFrame vs DataSet

Tushar B. Kute,
http://tusharkute.com

Spark RDD APIs

- An RDD stands for Resilient Distributed Datasets.
  - It is Read-only partition collection of records. RDD is the fundamental data structure of Spark.
  - It allows a programmer to perform in-memory computations on large clusters in a fault-tolerant manner.

Spark Dataframe APIs

- Unlike an RDD, data organized into named columns. For example a table in a relational database.
- It is an immutable distributed collection of data.
- DataFrame in Spark allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction.

Spark Dataset APIs

- Datasets in Apache Spark are an extension of DataFrame API which provides type-safe, object-oriented programming interface.
- Dataset takes advantage of Spark's Catalyst optimizer by exposing expressions and data fields to a query planner.

Spark Release

- RDD –
  - The RDD APIs have been on Spark since the 1.0 release.
- DataFrames –
  - Spark introduced DataFrames in Spark 1.3 release.
- DataSet –
  - Spark introduced Dataset in Spark 1.6 release.

Data Representation

- RDD – RDD is a distributed collection of data elements spread across many machines in the cluster. RDDs are a set of Java or Scala objects representing data.
- DataFrame – A DataFrame is a distributed collection of data organized into named columns. It is conceptually equal to a table in a relational database.
- DataSet – It is an extension of DataFrame API that provides the functionality of – type-safe, object-oriented programming interface of the RDD API and performance benefits of the Catalyst query optimizer and off heap storage mechanism of a DataFrame API.

Data Formats

- RDD – It can easily and efficiently process data which is structured as well as unstructured. But like Dataframe and DataSets, RDD does not infer the schema of the ingested data and requires the user to specify it.
- DataFrame – It works only on structured and semi-structured data. It organizes the data in the named column. DataFrames allow the Spark to manage schema.
- DataSet – It also efficiently processes structured and unstructured data. It represents data in the form of JVM objects of row or a collection of row object. Which is represented in tabular forms through encoders.

Data Sources API

- RDD – Data source API allows that an RDD could come from any data source e.g. text file, a database via JDBC etc. and easily handle data with no predefined structure.
- DataFrame – Data source API allows Data processing in different formats (AVRO, CSV, JSON, and storage system HDFS, HIVE tables, MySQL). It can read and write from various data sources that are mentioned above.
- DataSet – Dataset API of spark also support data from different sources.

Immutability and Interoperability

- RDD – RDDs contains the collection of records which are partitioned.
- The basic unit of parallelism in an RDD is called partition. Each partition is one logical division of data which is immutable and created through some transformation on existing partitions.
- Immutability helps to achieve consistency in computations. We can move from RDD to DataFrame (If RDD is in tabular format) by toDF() method or we can do the reverse by the .rdd method.

Immutability and Interoperability

- DataFrame – After transforming into DataFrame one cannot regenerate a domain object. For example, if you generate testDF from testRDD, then you won't be able to recover the original RDD of the test class.
- DataSet – It overcomes the limitation of DataFrame to regenerate the RDD from Dataframe.
- Datasets allow you to convert your existing RDD and DataFrames into Datasets.

Compile-time type safety

- RDD – RDD provides a familiar object-oriented programming style with compile-time type safety.
- DataFrame – If you are trying to access the column which does not exist in the table in such case Dataframe APIs does not support compile-time error. It detects attribute error only at runtime.
- DataSet – It provides compile-time type safety.

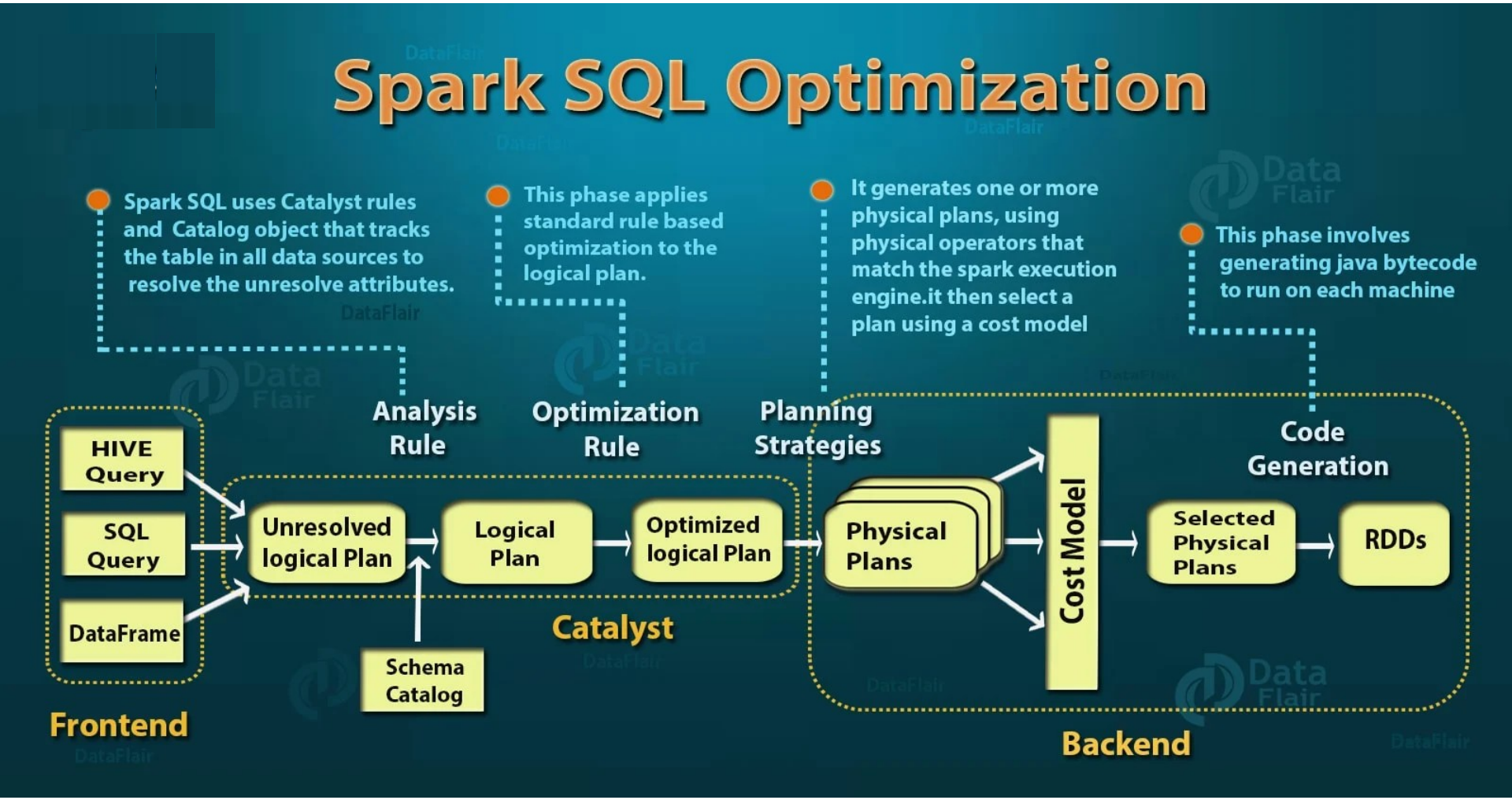
Optimization

- RDD – No inbuilt optimization engine is available in RDD. When working with structured data, RDDs cannot take advantages of sparks advance optimizers.
- For example, catalyst optimizer and Tungsten execution engine. Developers optimise each RDD on the basis of its attributes.

Optimization

- DataFrame – Optimization takes place using catalyst optimizer. Dataframes use catalyst tree transformation framework in four phases:
  - a) Analyzing a logical plan to resolve references.
  - b) Logical plan optimization.
  - c) Physical planning.
  - d) Code generation to compile parts of the query to Java bytecode.
- Dataset – It includes the concept of Dataframe Catalyst optimizer for optimizing query plan.

Optimization



Garbage Collection

- RDD – There is overhead for garbage collection that results from creating and destroying individual objects.
- DataFrame – Avoids the garbage collection costs in constructing individual objects for each row in the dataset.
- DataSet – There is also no need for the garbage collector to destroy object because serialization takes place through Tungsten. That uses off heap data serialization.

Efficiency/Memory use

- RDD – Efficiency is decreased when serialization is performed individually on a java and scala object which takes lots of time.
- DataFrame – Use of off heap memory for serialization reduces the overhead. It generates byte code dynamically so that many operations can be performed on that serialized data. No need for deserialization for small operations.
- DataSet – It allows performing an operation on serialized data and improving memory use. Thus it allows on-demand access to individual attribute without deserializing the entire object.

Serialization

- RDD – Whenever Spark needs to distribute the data within the cluster or write the data to disk, it does so use Java serialization.
- The overhead of serializing individual Java and Scala objects is expensive and requires sending both data and structure between nodes.

Serialization

- DataFrame – Spark DataFrame Can serialize the data into off-heap storage (in memory) in binary format and then perform many transformations directly on this off heap memory because spark understands the schema.
- There is no need to use java serialization to encode the data.
- It provides a Tungsten physical execution backend which explicitly manages memory and dynamically generates bytecode for expression evaluation.

Serialization

- DataSet – When it comes to serializing data, the Dataset API in Spark has the concept of an encoder which handles conversion between JVM objects to tabular representation.
- It stores tabular representation using spark internal Tungsten binary format.
- Dataset allows performing the operation on serialized data and improving memory use.
- It allows on-demand access to individual attribute without desterilizing the entire object.

Programming Language

- RDD – RDD APIs are available in Java, Scala, Python, and R languages. Hence, this feature provides flexibility to the developers.
- DataFrame – It also has APIs in the different languages like Java, Python, Scala, and R.
- DataSet – Dataset APIs is currently only available in Scala and Java. Spark version 2.1.1 does not support Python and R.

Schema Projection

- RDD – In RDD APIs use schema projection is used explicitly. Hence, we need to define the schema (manually).
- DataFrame – Auto-discovering the schema from the files and exposing them as tables through the Hive Meta store. We did this to connect standard SQL clients to our engine. And explore our dataset without defining the schema of our files.
- DataSet – Auto discover the schema of the files because of using Spark SQL engine.

Aggregation

- RDD – RDD API is slower to perform simple grouping and aggregation operations.
- DataFrame – DataFrame API is very easy to use. It is faster for exploratory analysis, creating aggregated statistics on large data sets.
- DataSet – In Dataset it is faster to perform aggregation operation on plenty of data sets.

Usage Area

- RDD-
  - You can use RDDs When you want low-level transformation and actions on your data set.
  - Use RDDs When you need high-level abstractions.
- DataFrame and DataSet-
  - One can use both DataFrame and dataset API when we need a high level of abstraction.
  - For unstructured data, such as media streams or streams of text.
  - You can use both Data Frames or Dataset when you need domain specific APIs.

Usage Area

- DataFrame and DataSet-
  - When you want to manipulate your data with functional programming constructs than domain specific expression.
  - We can use either datasets or DataFrame in the high-level expression. For example, filter, maps, aggregation, sum, SQL queries, and columnar access.
  - When you do not care about imposing a schema, such as columnar format while processing or accessing data attributes by name or column. in addition, If we want a higher degree of type safety at compile time.

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com