**Define RDD Lineage?**

**Answer:**RDD Lineage is a process of reconstructing the lost data partitions because Spark cannot support the data replication process in its memory. It helps in recalling the method used for building other datasets.

**What is the basic difference between Spark SQL, HQL, and SQL?**

**Answer:**Spark SQL supports SQL and Hiver Query language without changing any syntax. We can join SQL and HQL table with the Spark SQL.

**Can we trigger automated clean-ups in Spark?**

**Answer:**Yes, we can trigger automated clean-ups in Spark to handle the accumulated metadata. It can be done by setting the parameters, namely, “spark.cleaner.ttl.”

[Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/) is an open source big data framework. It has an expressive APIs to allow big data professionals to efficiently execute streaming as well as the batch. It provides faster and more general data processing platform engine. It is basically designed for fast computation.

It was developed to overcome the limitations of [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) cluster computing paradigm. Spark keeps things in memory whereas map reduce keep shuffling things in and out of disk. It allows to cache data in memory which is beneficial in iterative algorithm those used in machine learning.

**Apache spark consists of following components**  
1.Spark Core  
2.Spark SQL  
3.Spark Streaming  
4.MLlib  
5.GraphX

Spark Core is the fundamental unit of the whole Spark project. It provides all sort of functionalities like task dispatching, scheduling, and input-output operations etc.Spark makes use of Special data structure known as [RDD (Resilient Distributed Dataset)](http://data-flair.training/blogs/rdd-in-apache-spark/). It is the home for API that defines and manipulate the RDDs. Spark Core is distributed execution engine with all the functionality attached on its top.

Apache Spark surpasses Hadoop in many cases such as  
1. Processing the data in memory which is not possible in Hadoop  
2. Processing the data that is in batch, iterative, interactive & [streaming](https://data-flair.training/blogs/apache-spark-streaming-tutorial/) i.e. Real Time mode. Whereas Hadoop processes only in batch mode.  
3. Spark is faster because it reduces the number of disk read-write operations due to its virtue of storing intermediate data in memory. Whereas in Hadoop MapReduce intermediate output which is output of Map() is always written on local hard disk  
4. Apache Spark is easy to program as it has hundreds of high-level operators with [RDD (Resilient Distributed Dataset)](https://data-flair.training/blogs/apache-spark-rdd-tutorial/)  
5. Apache Spark code is compact due compared to Hadoop MapReduce. Use of Scala makes it very short, reduces programming efforts. Also, Spark provides rich APIs in various languages such as Java, [Scala](https://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/), Python, and [R](https://data-flair.training/blogs/r-programming-tutorial/).  
6. Spark & Hadoop are both highly [fault-tolerant](https://data-flair.training/blogs/fault-tolerance-in-apache-spark/).  
7. Spark application running in Hadoop clusters is up to 10 times faster on disk than Hadoop MapReduce.

A SparkContext is a client of Spark’s execution environment and it acts as the master of the [Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/)application. SparkContext sets up internal services and establishes a connection to a Spark execution environment. You can [create RDDs](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/), accumulators and broadcast variables, access Spark services and run jobs (until SparkContext stops) after the creation of SparkContext. Only one SparkContext may be active per JVM. You must stop() the active SparkContext before creating a new one.

The first step of any Spark driver application is to create a SparkContext. The SparkContext allows the Spark driver application to access the cluster through a resource manager. The resource manager can be [YARN](http://data-flair.training/blogs/category/yarn/), or [Spark’s Cluster Manager](http://data-flair.training/blogs/apache-spark-cluster-managers-tutorial/).

**Few functionalities which SparkContext offers are:**  
1. We can get the current status of a Spark application like configuration, app name.  
2. We can set Configuration like master URL, default logging level.  
3. One can create Distributed Entities like [RDDs.](http://data-flair.training/blogs/rdd-in-apache-spark/)

Starting from [Apache Spark](https://data-flair.training/forums/topic/what-is-sparksession-in-apache-spark) 2.0, Spark Session is the new entry point for Spark applications.

Prior to 2.0, [SparkContext](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/) was the entry point for spark jobs. [RDD](http://data-flair.training/blogs/rdd-in-apache-spark/) was one of the main APIs then, and it was created and manipulated using Spark Context. For every other APIs, different contexts were required – For SQL, SQL Context was required; For [Streaming](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/), Streaming Context was required; For [Hive](http://data-flair.training/blogs/category/hive/), Hive Context was required.

But from 2.0, RDD along with DataSet and its subset [DataFrame](http://data-flair.training/blogs/apache-spark-dataframe-tutorial/) APIs are becoming the standard APIs and are a basic unit of data abstraction in Spark. All of the user defined code will be written and evaluated against the DataSet and DataFrame APIs as well as RDD.

So, there is a need for a new entry point build for handling these new APIs, which is why Spark Session has been introduced. Spark Session also includes all the APIs available in different contexts – Spark Context, SQL Context, Streaming Context, Hive Context.

[Spark Context:](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/)  
Prior to [Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/) 2.0.0 sparkContext was used as a channel to access all spark functionality.  
The spark driver program uses spark context to connect to the cluster through a resource manager ([YARN](http://data-flair.training/blogs/category/yarn/) orMesos..).  
sparkConf is required to create the spark context object, which stores configuration parameter like appName (to identify your spark driver), application, number of core and memory size of executor running on worker node.

In order to use APIs of [SQL](http://data-flair.training/blogs/spark-sql-tutorial/)**,**[HIVE](http://data-flair.training/blogs/category/hive/)**, and**[Streaming](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/), separate contexts need to be created.

**Example:**  
creating sparkConf :

val conf = new SparkConf().setAppName(“RetailDataAnalysis”).setMaster(“spark://master:7077”).set(“spark.executor.memory”, “2g”)

creation of sparkContext:

val sc = new SparkContext(conf)

**Spark Session:**

SPARK 2.0.0 onwards, SparkSession provides a single point of entry to interact with underlying Spark functionality and  
allows programming Spark with [DataFrame](http://data-flair.training/blogs/apache-spark-dataframe-tutorial/) and Dataset APIs. All the functionality available with sparkContext are also available in sparkSession.

In order to use APIs of SQL, HIVE, and Streaming, no need to create separate contexts as sparkSession includes all the APIs.

Once the SparkSession is instantiated, we can configure Spark’s run-time config properties.

**Example:**

Creating Spark session:  
val spark = SparkSession  
.builder  
.appName(“WorldBankIndex”)  
.getOrCreate()

Configuring properties:  
spark.conf.set(“spark.sql.shuffle.partitions”, 6)  
spark.conf.set(“spark.executor.memory”, “2g”)

Spark 2.0.0 onwards, it is better to use sparkSession as it provides access to all the spark Functionalities that sparkContext does. Also, it provides APIs to work on DataFrames and Datasets.

 Immutable data is always safe to share across multiple processes as well as multiple threads.  
– Since [RDD](http://data-flair.training/blogs/rdd-in-apache-spark/) is immutable we can recreate the RDD any time. (From lineage graph).  
– If the computation is time-consuming, in that we can cache the RDD which result in performance improvement.

**Introduction**  
Paired RDD is a distributed collection of data with the key-value pair. It is a subset of [Resilient Distributed Dataset](https://data-flair.training/blogs/apache-spark-rdd-tutorial/). So it has all the [feature of RDD](https://data-flair.training/blogs/apache-spark-rdd-features/) and some new feature for the key-value pair. There are many [transformation operations](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/) available for Paired RDD. These operations on Paired RDD are very useful to solve many use cases that require sorting, grouping, reducing some value/function.  
Commonly used operations on paired RDD are: groupByKey() reduceByKey() countByKey() join() etc

Transformations are [operations on RDD](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/) that create one or more new [RDDs](http://data-flair.training/blogs/rdd-in-apache-spark/). E.g. map, filter, reduceByKey etc. In other words, transformations are functions that take an RDD as the input and produce one or more RDDs as the output. There is no change in the input RDD, but it always produces one or more new RDDs by applying the computations they represent.Transformations are lazy, i.e. are not executed immediately. Only after calling an action are transformations executed.

Actions are RDD operations that produce non-RDD values. In other words, an RDD operation that returns a value of any type but an RDD is an action. They trigger execution of RDD transformations to return values. Simply put, an action evaluates the [RDD lineage graph](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/). E.g. collect, reduce, count, foreach etc.

<li style=”list-style-type: none”>

* RDD (Resilient Distributed Dataset) is a basic abstraction in [Apache Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/).
* RDD is an immutable, partitioned collection of elements on the cluster which can be operated in parallel.

<li style=”list-style-type: none”>

* By Default, Spark creates one Partition for each block of the file (For HDFS)
* Default block size for HDFS block is 64 MB (Hadoop Version 1) / 128 MB (Hadoop Version 2).
* However, one can explicitly specify the number of partitions to be created.

val rdd1 = sc.textFile(“/home/hdadmin/wc-data.txt”)

Consider the size of wc-data.txt is of 1280 MB and Default block size is 128 MB. So there will be 10 blocks created and 10 default partitions(1 per block).

For a better performance, we can increase the number of partitions on each block. Below code will create 20 partitions on 10 blocks(2 partitions/block). Performance will be improved but need to make sure that each cluster is running on 2 cores minimum.

val rdd1 = sc.textFile(“/home/hdadmin/wc-data.txt”, 20)

**Introduction**  
**DataFrame** consists of two words data and frame, means data has to be fit in some kind of frame. We can understand a frame as a schema of the relational database.

In Spark, DataFrame is a collection of distributed data over the network with some schema. We can understand it as the data formatted as row/column manner. DataFrame can be created from Hive data, JSON file, CSV, Structured data or raw data that can be framed in structured data. We can also create a DataFrame from [RDD](https://data-flair.training/blogs/apache-spark-rdd-tutorial/) if some schema can be applied on that RDD.  
Temporary view or table can also be created from DataFrame as it has data and schema. We can also run [SQL](https://data-flair.training/blogs/spark-sql-tutorial/) query on created table/view to get the faster result.  
It is also evaluated lazily ([Lazy Evaluation](https://data-flair.training/blogs/apache-spark-lazy-evaluation/)) for better resource utilization.

A **Dataset** is an immutable collection of objects, those are mapped to a relational schema. They are strongly-typed in nature.  
There is an encoder, at the core of the Dataset API. That Encoder is responsible for converting between JVM objects and  
tabular representation. By using Spark’s internal binary format, the tabular representation is stored that allows to carry out operations on serialized data and improves memory utilization. It also supports automatically generating encoders for a wide variety of types, including primitive types

benefits of Dataset in Apache Spark?

**1)Static typing-**  
With Static typing feature of Dataset, a developer can catch errors at compile time (which saves time and costs).  
**2)Run-time Safety**:-  
[Dataset](https://data-flair.training/blogs/apache-spark-dataset-tutorial) APIs are all expressed as lambda functions and JVM typed objects, any mismatch of typed-parameters will be  
detected at compile time. Also, analysis error can be detected at compile time too, when using Datasets,  
hence saving developer-time and costs.  
**3)Performance and Optimization**

Parquet is a columnar format supported by many data processing systems. The benifits of having a columnar storage are –

1- Columnar storage limits IO operations.

2- Columnar storage can fetch specific columns that you need to access.

3-Columnar storage consumes less space.

4- Columnar storage gives better-summarized data and follows type-specific encoding.

**Introduction**  
**Lazy evaluation** means the execution will not start until an[action](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/) is triggered. [Transformations](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/) are lazy in nature i.e. when we call some operation on [RDD](https://data-flair.training/blogs/apache-spark-rdd-tutorial/), it does not execute immediately. [Spark](https://data-flair.training/blogs/apache-spark-for-beginners/) adds them to a [DAG](https://data-flair.training/blogs/dag-in-apache-spark/) of computation and only when driver requests some data, this DAG actually gets executed

Advantages of lazy evaluation.

1) It is an optimization technique i.e. it provides optimization by reducing the number of queries.  
2) It saves the round trips between driver and cluster, thus speeds up the process.

In [Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/), Data Transfer can be reduced by avoiding operation which results in data shuffle.  
Avoid operations like repartition and coalesce, ByKey operations like groupByKey and reduceByKey, and join operations like cogroup and join.

Spark Shared Variables help in reducing data transfer. There two types for shared variables-Broadcast variable and Accumulator.

**Broadcast variable:**

If we have a large dataset, instead of transferring a copy of data set for each task, we can use a broadcast variable which can be copied to each node at one time  
and share the same data for each task in that node. Broadcast variable help to give a large data set to each node.  
First, we need to create a broadcast variable using SparkContext.broadcast and then broadcast the same to all nodes from driver program. Value method  
can be used to access the shared value. The broadcast variable will be used only if tasks for multiple stages use the same data.

**Accumulator:**  
Spark functions used variables defined in the driver program and local copied of variables will be generated. Accumulator are shared variables which help to update  
variables in parallel during execution and share the results from workers to the driver.

The spark driver is that the program that defines the [transformations and actions on RDDs](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/) of knowledge and submits request to the master. Spark driver is a program that runs on the master node of the machine which declares transformations and actions on knowledge RDDs.

In easy terms, the driver in Spark creates [SparkContext](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/), connected to a given Spark Master.It conjointly delivers the RDD graphs to Master, wherever the standalone cluster manager runs.

What is role of Driver program in Spark Application ?

* Driver program is responsible for launching various parallel operations on the cluster.
* Driver program contains application’s *main()* function.
* It is the process which is running the user code which in turn create the SparkContext object, [create RDDs](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/)and performs [transformation and action operation on RDD](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/).
* Driver program access [Apache Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/)through a [SparkContext](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/) object which represents a connection to computing cluster (From Spark 2.0 onwards we can access SparkContext object through SparkSession).
* Driver program is responsible for converting user program into the unit of physical execution called task.
* It also defines distributed datasets on the cluster and we can apply different operations on Dataset (transformation and action).
* Spark program creates a logical plan called [Directed Acyclic graph](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/) which is converted to physical execution plan by the driver when driver program runs.
* fold() is an action. It is wide operation (i.e. shuffle data across multiple partitions and output a single value)
* It takes function as an input which has two parameters of the same type and outputs a single value of the input type.
* It is similar to reduce but has one more argument ‘ZERO VALUE’ (say initial value) which will be used in the initial call on each partition.

**Reduce:**  
Reduce methods walk through the elements in a collection,  
applying your function to neighboring elements to yield a new result,  
which is then compared to the next element in the sequence to yield a new result

def reduce[T]((value1,value1) => res)

**Fold:**  
Fold also works similar to Reduce and aggregate over a collection by executing an operation  
but with a specified initial value

def fold[T](acc:T)((acc,value) => acc)