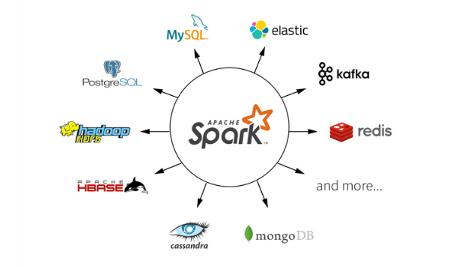
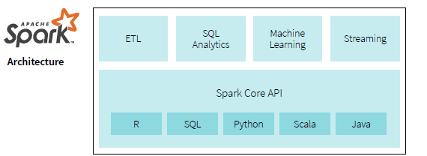
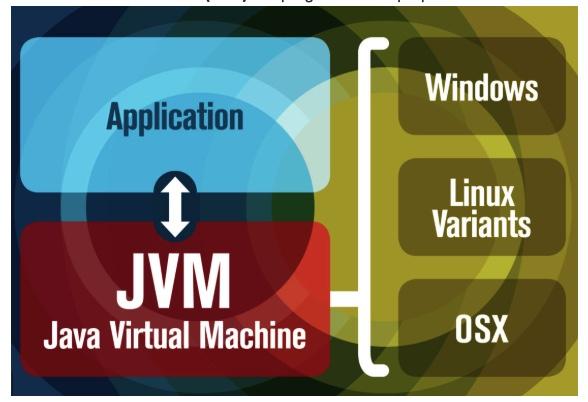
**#Question1**:  
What is Apache Spark?  
  
Apache Spark is a unified analytic engine for large-scale data processing. It is a lightning-fast engine for big data and machine learning. The largest open source project in data processing. It works seamlessly on almost all open source big data technologies.

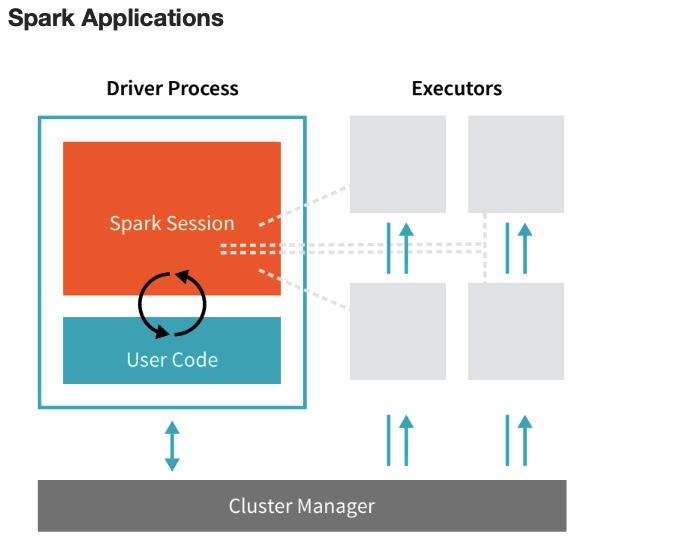


**#Question2**:  
What are main components of Apache Spark ?  
  
Spark has 2 major components:  
  
Spark Core : Which consists of Spark Engine and the sets of APIs ( Java, Python, R, Sql , Scala)  
  
Set of libraries : Spark Sql ( Sql queries), Streaming (Real Time) , Mllib (Machine Learning), GraphX (Graph Computation) etc.

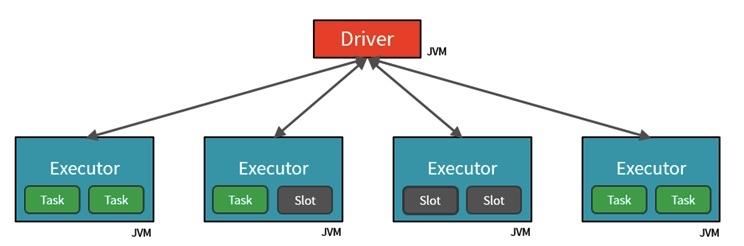


**#Question3**:  
What is a JVM?  
  
The JVM manages system memory and provides a portable execution environment for Java-based applications  
  
Technical definition: The JVM is the specification for a software program that executes code and provides the runtime environment for that code.  
  
Everyday definition: The JVM is how we run our Java programs. We configure the JVM's settings and then rely on it to manage program resources during execution. The Java Virtual Machine (JVM) is a program whose purpose is to execute other programs.  
  
   

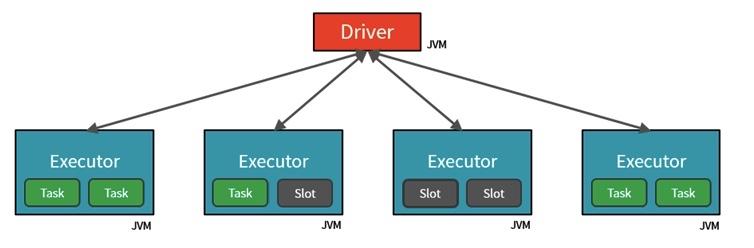
**#Question4**:  
Explain Spark's Basic Architecture ?  
  
A cluster, or group of machines, pools the resources of many machines together allowing us to use all the cumulative resources as if they were one. Now a group of machines sitting somewhere alone is not powerful, you need a framework to coordinate work across them.  
  
Spark is a tailor-made engine exactly for this, managing and coordinating the execution of tasks on data across a cluster of computers.  
  
The cluster of machines that Spark will leverage to execute tasks will be managed by a cluster manager like Spark’s Standalone cluster manager, YARN - Yet Another Resource Negotiator, Kubernetes. We then submit Spark Applications to these cluster managers which will grant resources to our application so that we can complete our work.  
  
Spark Applications consist of a driver process and a set of executor processes. In the illustration we see below, our driver is on the left and four executors on the right.



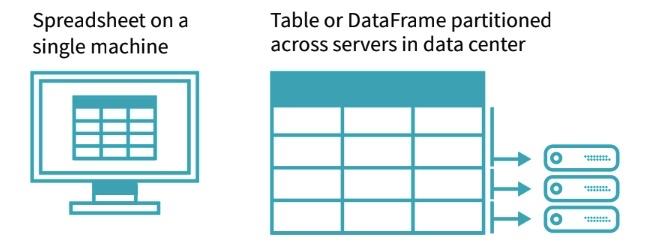
**#Question5**: What is a Driver and Executor process in Spark ?  
  
Driver: The driver process runs your main() functions , sits on the node in the cluster and is responsible for 3 main things:  
  
1. Maintaining information about the spark application. Its a heart of a spark application and maintains all the I information during the lifetime of the application.  
2. Responding to user’s program or input.  
3. Analyzing, distributing and scheduling work across the executors.  
  
Executor: The executors are responsible for carrying out the work that the driver assigns them.  
  
1. Execute code assigned to it by the driver.  
2 Reporting the state of the computation on that executor back to driver.



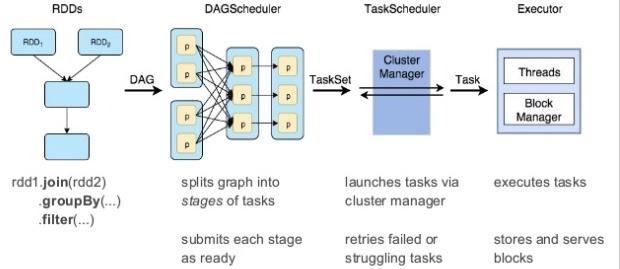
**#Question6**: What is Cores / Slots / Threads?  
  
Spark parallelizes at two levels. One is the splitting the work among executors. The other is the slot. Each executor has a number of slots. Each slot can be assigned a Task.  
  
For example: the diagram below is showing 2 Core Executor nodes:  
  
1. The JVM is naturally multithreaded, but a single JVM, such as our Driver, has a finite upper limit.  
  
2. By creating Tasks, the Driver can assign units of work to Slots on each Executor for parallel execution.  
  
3. Additionally, the Driver must also decide how to partition the data so that it can be distributed for parallel processing (see below).  
4. Consequently, the Driver is assigning a Partition of data to each task - in this way each Task knows which piece of data it is to process.  
5. Once started, each Task will fetch from the original data source (e.g. An Azure Storage Account) the Partition of data assigned to it.



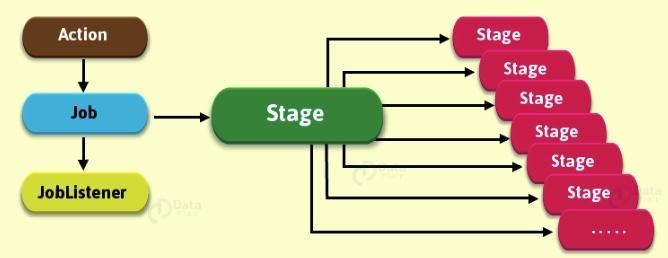
**#Question7**:  
What is a Partition in Spark ?  
  
In order to allow every executor to perform work in parallel, Spark breaks up the data into chunks, called partitions.  
  
A partition is a collection of rows that sit on one physical machine in our cluster. A DataFrame’s partitions represent how the data is physically distributed across your cluster of machines during execution:  
  
1. If you have one partition, Spark will only have a parallelism of one, even if you have thousands of executors.  
  
2. If you have many partitions, but only one executor, Spark will still only have a parallelism of one because there is only one computation resource.  
  
An important thing to note is that with DataFrames, we do not (for the most part) manipulate partitions manually (on an individual basis). We simply specify high level transformations of data in the physical partitions and Spark determines how this work will actually execute on the cluster.



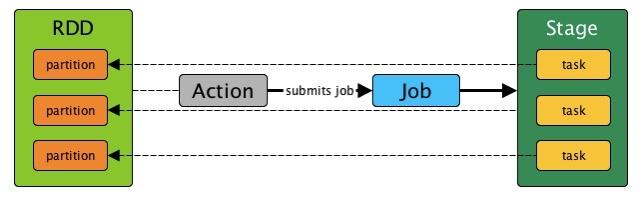
**#Question8**:  
What is a DAG in Spark ?  
  
Directed Acyclic Graph ( DAG ) in Apache Spark is a set of Vertices and Edges, where vertices represent the RDDs and the edges represent the Operation to be applied on RDDs.  
  
DAGScheduler is the scheduling layer of Apache Spark that implements stage-oriented scheduling. It transforms a logical execution plan to a physical execution plan (using stages).  
  
After an action has been called, SparkContext hands over a logical plan to DAGScheduler that it in turn translates to a set of stages that are submitted as a set of tasks for execution.  
  
The fundamental concepts of DAGScheduler are jobs and stages that it tracks through internal registries and counters.



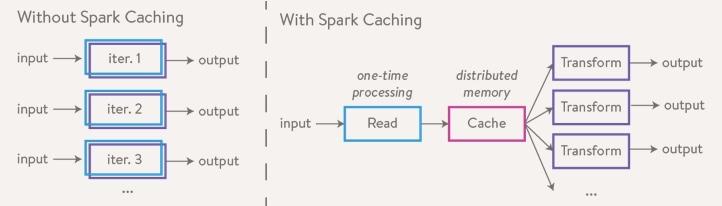
**#Question9**:  
What is a Job and Stage in Spark ?  
  
A Job is a sequence of stages, triggered by an action such as count(), collect(), read() or write().  
  
1. Each parallelized action is referred to as a Job.  
2. The results of each Job (parallelized/distributed action) is returned to the Driver from the Executor.  
3. Depending on the work required, multiple Jobs will be required.  
  
Each job that gets divided into smaller sets of tasks is a stage.  
  
A Stage is a sequence of Tasks that can all be run together - i.e. in parallel - without a shuffle. For example: using ".read" to read a file from disk, then runnning ".filter" can be done without a shuffle, so it can fit in a single stage.  
  
The number of Tasks in a Stage also depends upon the number of Partitions your datasets have.



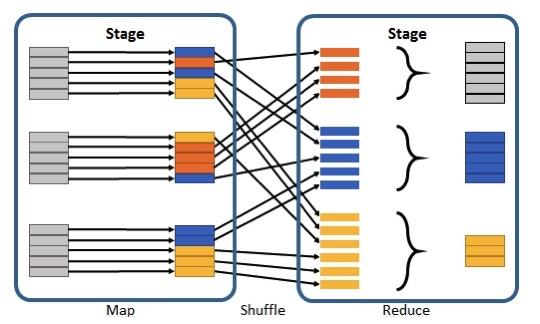
**#Question10**:  
What is a Task in Spark?  
  
A task is a unit of work that is sent to the executor. Each stage has some tasks, one task per partition. The same task is done over different partitions of the RDD.  
  
In the example of Stages below, each Step is a Task.



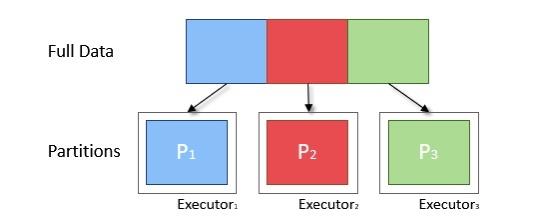
**#Question11**:  
What is a Caching in Spark ?  
  
In applications that reuse the same datasets over and over, one of the most useful optimizations is caching. Caching will place a DataFrame or table into temporary storage across the executors in your cluster and make subsequent reads faster.



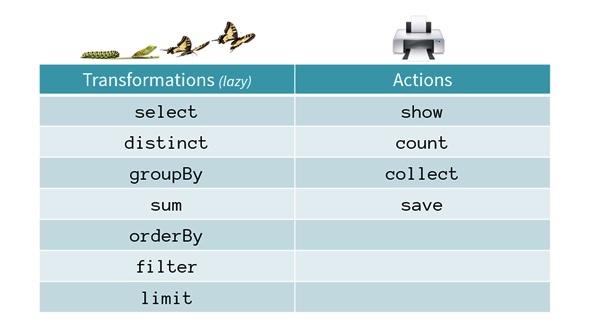
**#Question12**:  
What is a Shuffling in Spark ?  
  
A Shuffle refers to an operation where data is re-partitioned across a Cluster - i.e. when data needs to move between executors.  
  
Join and any operation that ends with "ByKey" will trigger a Shuffle. It is a costly operation because a lot of data can be sent via the network.  
  
For example, to group by color, it will serve us best if...  
1. All the reds are in one partitions  
2. All the blues are in a second partition  
3. All the greens are in a third  
  
From there we can easily sum/count/average all of the reds, blues, and greens.



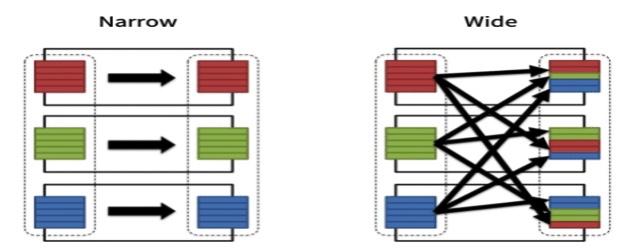
**#Question13**:  
What is a Partitioning in Spark ?  
  
A Partition is a logical chunk of your DataFrame.  
  
Data is split into Partitions so that each Executor can operate on a single part, enabling parallelization.  
  
It can be processed by a single Executor core/thread.  
  
For example: If you have 4 data partitions and you have 4 executor cores/threads, you can process everything in parallel, in a single pass.



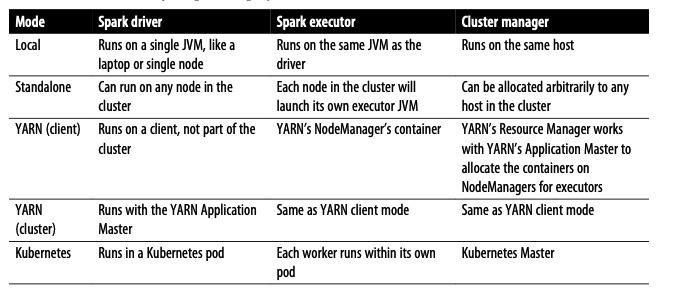
**#Question14**:  
Transformations Vs Actions in Spark ?  
  
Transformations:  
In Spark, the core data structures are immutable meaning they cannot be changed once created. In order to "change" a DataFrame you will have to instruct Spark how you would like to modify the DataFrame you have into the one that you want. These instructions are called transformations. Let’s perform a simple transformation to find all even numbers in our currentDataFrame. Examples – Select, Filter, GroupBy, Join, Union, Partition etc  
  
Actions:  
Transformations allow us to build up our logical transformation plan. To trigger the computation, we run an action. An action instructs Spark to compute a result from a series of transformations. The simplest action is count which gives us the total number of records in the DataFrame.



**#Question15**:  
Narrow Transformations Vs Wide Transformations in Spark ?  
  
There are two types of transformations: Narrow and Wide.  
  
For narrow transformations, the data required to compute the records in a single partition reside in at most one partition of the parent dataset.  
  
Examples include:  
filter(..)  
drop(..)  
coalesce()  
  
For wide transformations, the data required to compute the records in a single partition may reside in many partitions of the parent dataset.  
  
Examples include:  
distinct()  
groupBy(..).sum()  
repartition(n)  
  
Remember, spark partitions are collections of rows that sit on physical machines in the cluster. Narrow transformations mean that work can be computed and reported back to the executor without changing the way data is partitioned over the system. Wide transformations require that data be redistributed over the system. This is called a shuffle.  
  
Shuffles are triggered when data needs to move between executors.



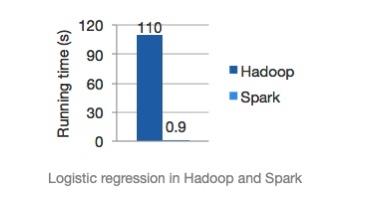
**#Question16**:  
What are different deployment mode in Apache Spark ?  
  
An attractive feature of Spark is its support for various deployment modes, enabling Spark to run in different configurations and environments.  
  
Because the cluster manager is agnostic to where it runs (as long as it can manage Spark’s executors and fulfil resource requests), Spark can be deployed in some of the most popular environments.  
  
1. Apache Hadoop YARN ( Client & Cluster Mode )  
2. Kubernetes  
3. Localhost  
4. Standalone



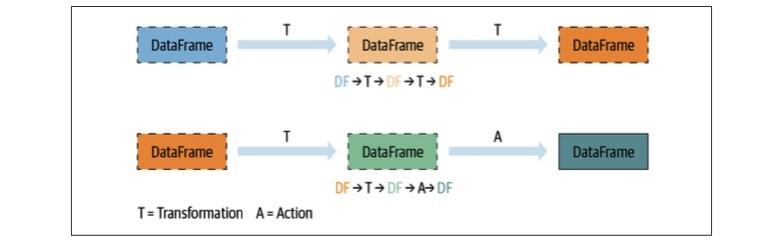
**#Question17**:  
What is a SparkSession in Apache Spark ?  
  
In Spark 2.0, the SparkSession became a unified entry point to all Spark operations and data. Not only did it subsume previous entry points to Spark like the SparkContext, SQLContext, HiveContext, SparkConf, and StreamingContext, but it also made working with Spark simpler and easier.  
  
Through this , you can create JVM runtime parameters, define DataFrames and Datasets, read from data sources, access catalog metadata, and issue Spark SQL queries. SparkSession provides a single unified entry point to all of Spark’s functionality.  
  
In a standalone Spark application, you can create a SparkSession using one of the high-level APIs in the programming language of your choice.  
  
In the Spark shell, the SparkSession is created for you, and you can access it via a global variable called spark or sc.



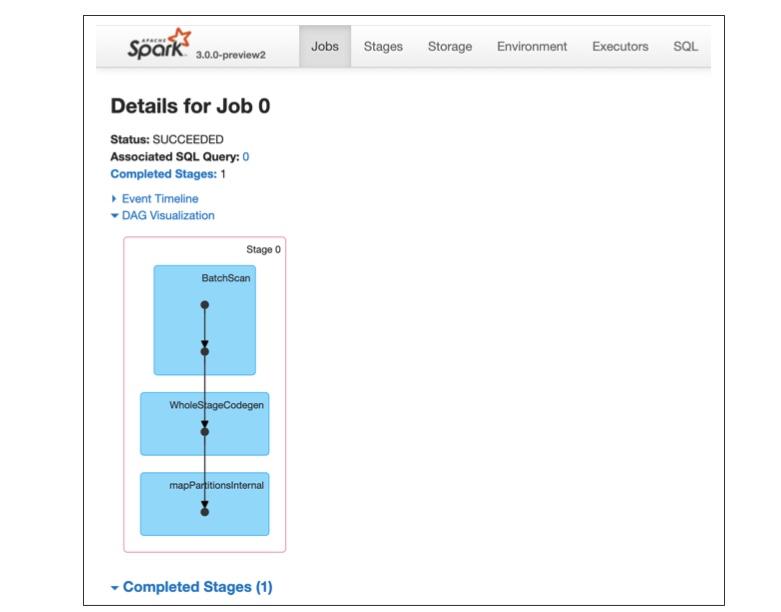
**#Question18**:  
What is Spark’s design philosophy ?  
  
Spark’s design philosophy centers around four key characteristics:  
  
1. Speed : Run workloads 100x faster.  
Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.  
  
2. Ease of use: Write applications quickly in Java, Scala, Python, R, and SQL  
  
3. Modularity: Combine SQL, streaming, and complex analytics.  
  
4. Extensibility : Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.



**#Question19**:  
Explain Lazy Evaluation in Spark ?  
  
Spark operations on distributed data can be classified into two types: transformations and actions. An operation such as select() or filter() will not change the original DataFrame; instead, it will return the transformed results of the operation as a new DataFrame.  
  
All transformations are evaluated lazily. That is, their results are not computed immediately, but they are recorded or remembered as a lineage. A recorded lineage allows Spark, at a later time in its execution plan, to rearrange certain transformations, coalesce them, or optimize transformations into stages for more efficient execution.  
  
Lazy evaluation is Spark’s strategy for delaying execution until an action is invoked or data is “touched”.  
  
While lazy evaluation allows Spark to optimize your queries by peeking into your chained transformations, lineage and data immutability provide fault tolerance.  
  
All transformations T are recorded until the action A is invoked. Each transformation T produces a new DataFrame.



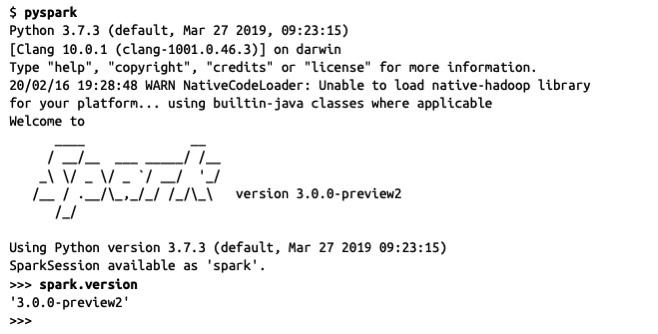
**#Question20**:  
What is Spark UI?  
  
Spark includes a graphical user interface that you can use to inspect or monitor Spark applications in their various stages of decomposition—that is jobs, stages, and tasks. Depending on how Spark is deployed, the driver launches a web UI, running by default on port 4040, where you can view metrics and details such as:  
  
1. A list of scheduler stages and tasks  
2. A summary of RDD sizes and memory usage  
3. Information about the environment  
4. Information about the running executors  
5. All the Spark SQL queries  
  
In local mode, you can access this interface at http://<localhost>:4040 in a web browser.



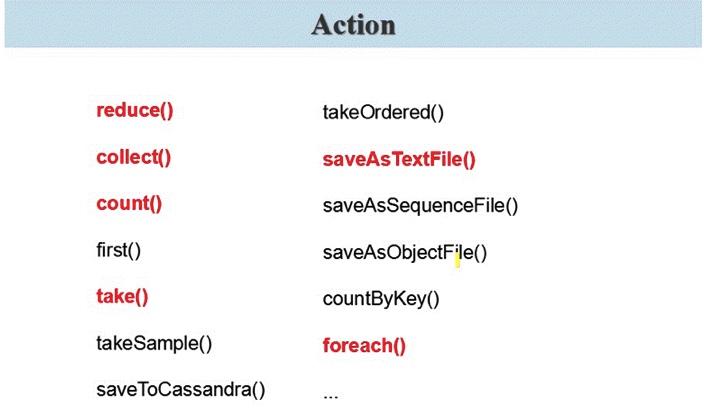
**#Question21**:  
How can we download Apache Spark Software ?  
  
Go to the Spark download page - **https://lnkd.in/g5b5mD5** , select “Pre-built for Apache Hadoop **2.7”** from the drop-down menu in step 2, and click the “Download Spark” link in step 3.  
  
This will download the tarball **spark-3.0.0-preview2-bin-hadoop2.7.tgz**, which con‐ tains all the Hadoop-related binaries you will need to run Spark in local mode on your laptop. Alternatively, if you’re going to install it on an existing HDFS or Hadoop installation, you can select the matching Hadoop version from the drop-down menu.  
  
Once you have finished downloading the tarball, cd to the downloaded directory, extract the tarball contents with  
tar -xf spark-3.0.0-preview2-bin- hadoop2.7.tgz,  
and cd into that directory and take a look at the contents:  
  
$ cd spark-3.0.0-preview2-bin-hadoop2.7  
$ls  
LICENSE R RELEASE conf examples kubernetes python yarn NOTICE **README.md** bin data jars licenses sbin



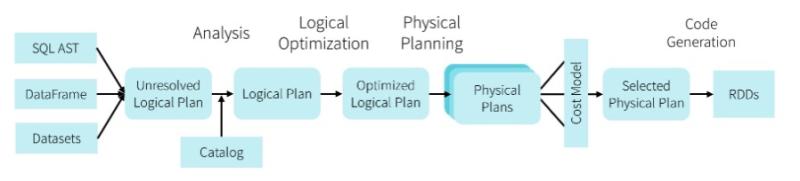
**#Question22**:  
How can we use the Scala or PySpark Shell?  
  
Spark comes with four widely used interpreters that act like interactive “shells” and enable ad hoc data analysis:  
  
1. pyspark  
2. spark-shell  
3. spark- sql  
4. sparkR  
  
To start PySpark, cd to the bin directory and launch a shell by typing pyspark.  
  
To start a similar Spark shell with Scala, cd to the bin directory and type spark-shell



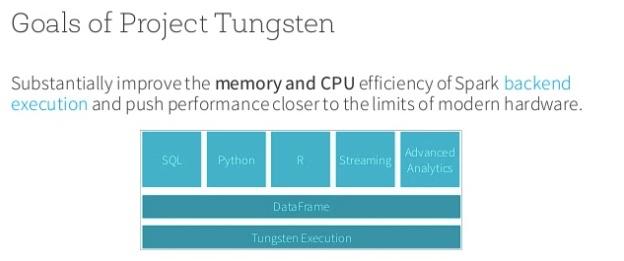
**#Question23**:  
How many kind of actions are possible in Spark?  
  
There are 3 main actions.  
  
1. To view the data on console  
2. To collect the data to native objects in the respective language ( python / Scala / java etc )  
3. To write to output data sources



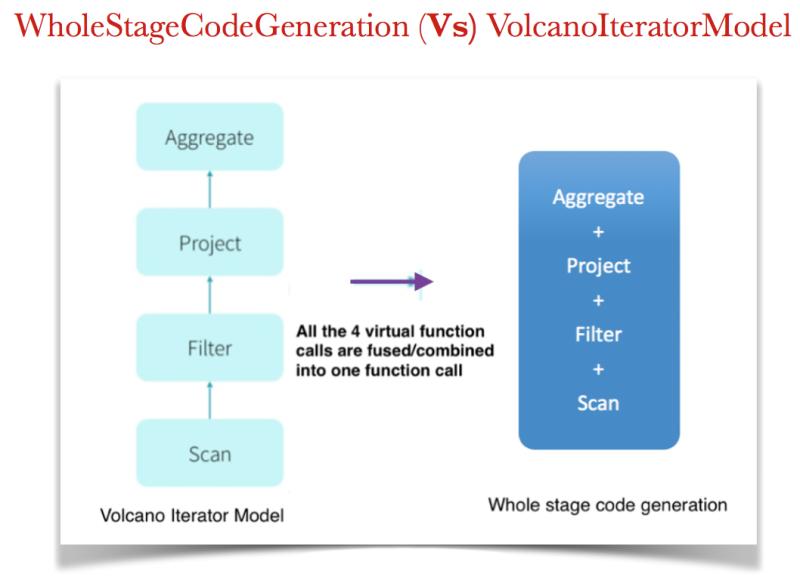
**#Question24**:  
What is Catalyst Optimiser ?  
  
Among the most powerful components of Spark are Spark SQL. At its core lies the Catalyst optimizer. This extensible query optimizer supports both rule-based and cost-based optimization.  
  
In rule-based optimization the rule based optimizer use set of rule to determine how to execute the query.  
  
While the cost based optimization finds the most suitable way to carry out SQL statement. In cost-based optimization, multiple plans are generated using rules and then their cost is computed.  
  
Catalyst contains the tree and the set of rules to manipulate the tree.  
  
Catalyst basically generates an optimised physical query plan from the logical query plan by applying a series of transformations like predicate pushdown, column pruning, and constant folding on the logical plan.



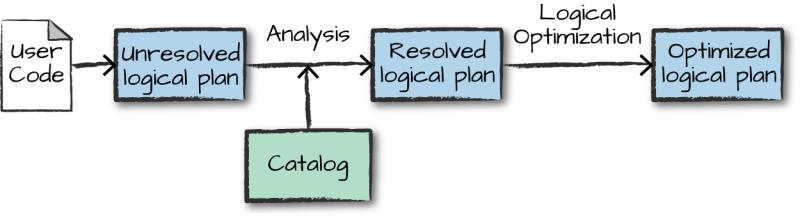
**#Question25**:  
What is Tungsten Optimizer ?  
  
The goal of Project Tungsten is to improve Spark execution by optimizing Spark jobs for CPU and memory efficiency ( As opposed to network and disk I/O which are considered fast enough ).  
  
1. Off-Heap Memory Management using binary in-memory data representation aka Tungsten row format and managing memory explicitly,  
2. Cache Locality which is about cache-aware computations with cache-aware layout for high cache hit rates  
3. Whole-Stage Code Generation (aka CodeGen).



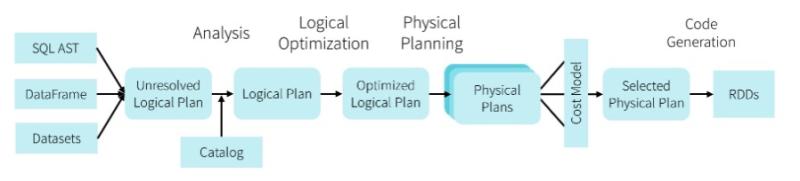
**#Question26**:  
What is "Whole-Stage Codegen" in Apache Spark ?  
  
Whole-Stage CodeGen is also known as Whole-Stage Java Code Generation, which is a physical query optimization phase in Spakr SQL that clubs multiple physical operations together to form a single Java function.  
  
Whole-Stage Java code generation improves the execution performance by converting a query tree into an optimized function that eliminates unnecessary calls and leverages CPU registers for intermediate data.  
  
Whole-Stage CodeGen is enabled by default in Spark 2.x. This can be controlled by the property **spark.sql.codegen.wholeStage.**  
  
Whole-Stage CodeGen is getting used by some of the modern massively parallel processing databases to achieve efficiency in execution performance.



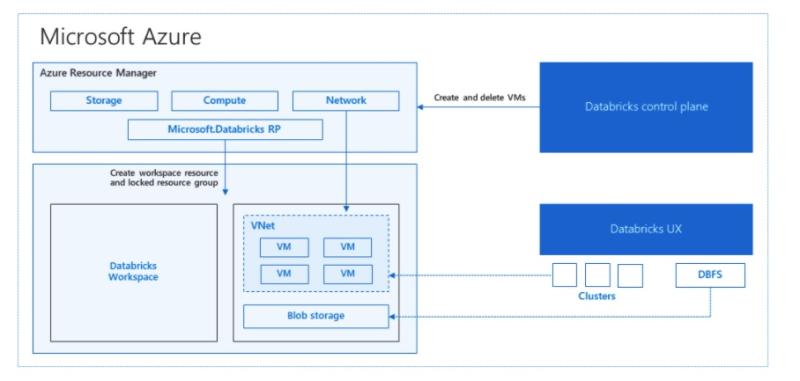
**#Question27**:  
How the Catalyst Optimiser finds out the CATALOG objects ?  
  
Basic catalog is SessionCatalog which serves only as a proxy to actual ExternalCatalog.  
  
Spark provides two different implementations of the ExternalCatalog out-of-the-box: InMemoryCatalog and HiveExternalCatalog which correspond to standard SQLContext and HiveContext respectively.  
  
Obviously the former one may access Hive metastore but there should be no data access otherwise.  
  
In Spark 2.0+ catalog can be queried directly using **SparkSession.catalog** for example: **Spark.catalog.listTables**



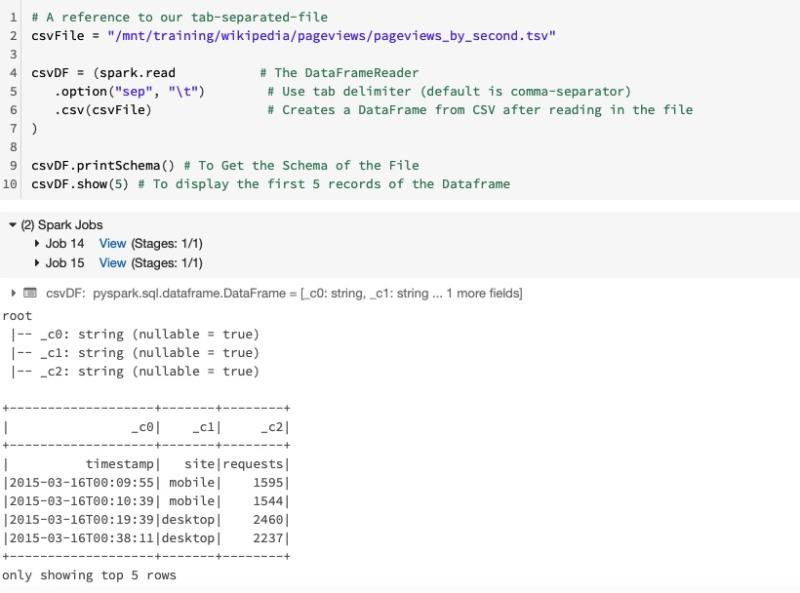
**#Question28**:  
What are different stages in the Spark Optimisation ?  
  
4 Stages:  
  
1. Analysis : Spark SQL Optimization starts from relation to be computed. It is computed either from abstract syntax tree (AST) returned by SQL parser or dataframe object created using API. Both may contain unresolved attribute references or relations. By unresolved attribute, it means we don’t know its type or have not matched it to an input table. Spark SQL make use of Catalyst rules and a Catalog object that track data in all data sources to resolve these attributes. It starts by creating an unresolved logical plan, and then apply the analysis rules  
  
2. Logical Optimization : In this phase of Spark SQL optimization, the standard rule-based optimization is applied to the logical plan. It includes constant folding, predicate pushdown, projection pruning and other rules.  
  
3. Physical Planning: In this phase, one or more physical plan is formed from the logical plan, using physical operator matches the Spark execution engine. And it selects the plan using the cost model.  
  
4. Code Generation: The final phase of Spark SQL optimization is code generation. It involves generating Java bytecode to run on each machine.



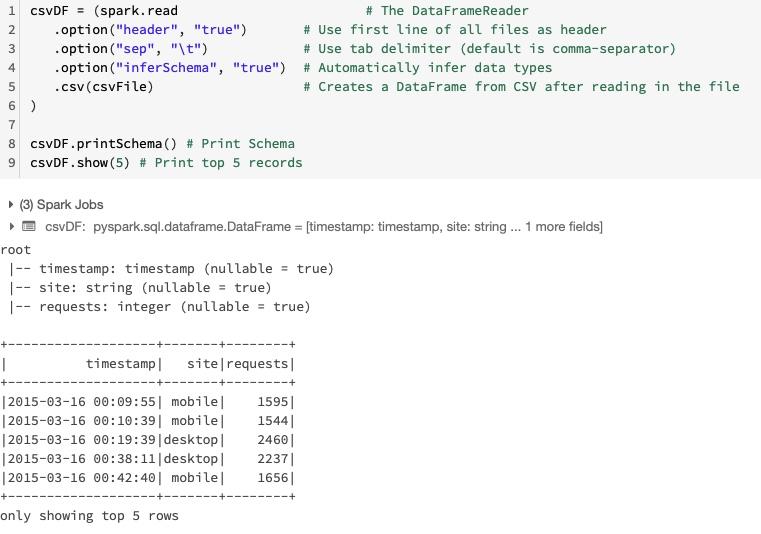
**#Question29**:  
What happens under the covers when you create an Azure Databricks service ?  
  
When you create an Azure Databricks service, a "Databricks appliance" is deployed as an Azure resource in your subscription.  
  
At the time of cluster creation, you specify the types and sizes of the virtual machines (VMs) to use for both the Driver and Worker nodes, but Azure Databricks manages all other aspects of the cluster.  
  
The "Databricks appliance" is deployed into Azure as a managed resource group within your subscription. This resource group contains the Driver and Worker VMs, along with other required resources, including a virtual network, a security group, and a storage account.  
  
All metadata for your cluster, such as scheduled jobs, is stored in an Azure Database with geo-replication for fault tolerance.  
  
Internally, Azure Kubernetes Service (AKS) is used to run the Azure Databricks control-plane and data-planes via containers running on the latest generation of Azure hardware.  
  
Once the services within this managed resource group are ready, you will be able to manage the Databricks cluster through the Azure Databricks UI and through features such as auto-scaling and auto-termination.



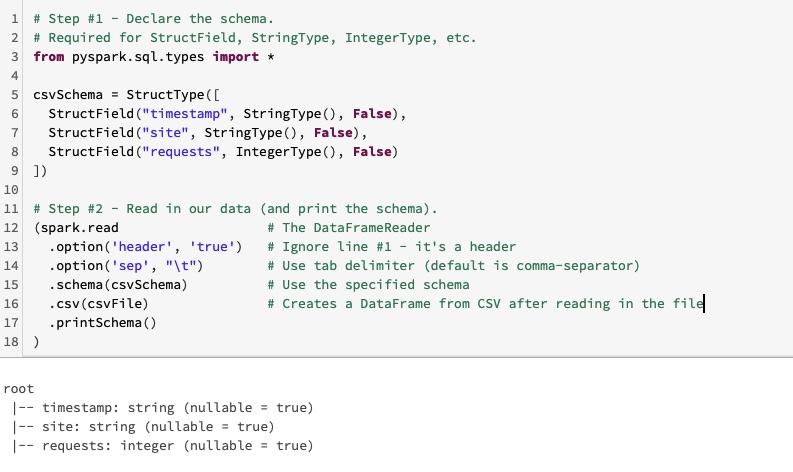
**#Question30**:  
Read The CSV File in Spark ?  
  
Below are the points worth noting about our file:  
  
1. The file is in TSV format.  
2. The file has a header.  
3. The file is tab separated  
4. The first two columns are strings and the third is a number.  
  
Knowing those details, we can read in the "CSV" file.  
  
A Spark job will trigger . A Job is triggered anytime we are "physically" required to touch the data.  
  
In some cases, one action may create multiple jobs (multiple reasons to touch the data ).  
  
In this case, the reader has to "peek" at the first line of the file to determine how many columns of data we have.



**#Question31**:  
Read The CSV File in Spark by Using the File's Header  
and Inferring the Schema ?  
  
Below are the points worth noting about our file:  
  
1. The file is in TSV format.  
2. The file has a header.  
3. The file is tab separated  
4. The first two columns are strings and the third is a number.  
  
  
Note the below points:  
1. Now, All three columns have their proper names ( header is True )  
2. Two jobs were executed (not one as in the previous example) - Because inferring the schema requires additional job  
3. Our three columns now have distinct data types ( Because we have inferred the schema ) :  
timestamp == timestamp  
site == string  
requests == integer



**#Question32**:  
Reading from CSV with User-Defined Schema ?  
  
Below are the points worth noting about our file:  
  
1. The file is in TSV format.  
2. The file has a header.  
3. The file is tab separated  
4. The first two columns are strings and the third is a number.  
  
  
Note the below points:  
1. All three columns are NOT nullable because we declared them as such. ( In the Schema )  
2. All three columns have their proper names ( Header is True)  
3. Zero jobs were executed ( No need to touch the data )  
4. Our three columns now have distinct data types:  
timestamp == string  
site == string  
requests == integer



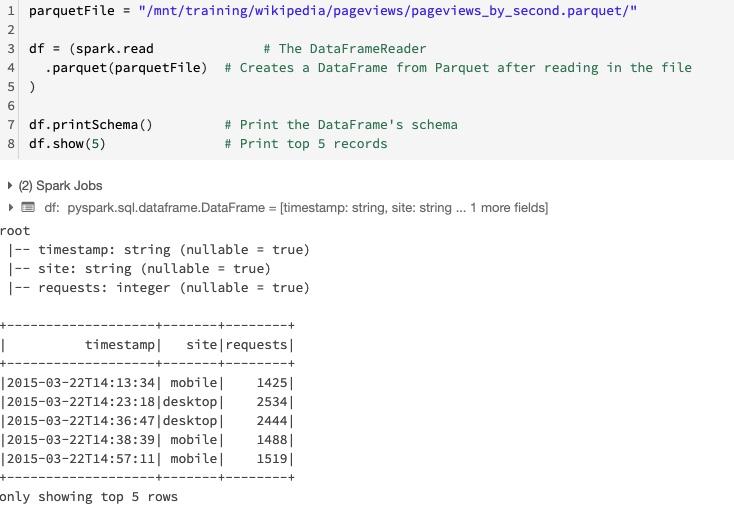
**#Question33**:  
Print the Number of Partitions in the DataFrame ?  
  
Below are the points worth noting about our file:  
  
1. The file is in TSV format.  
2. The file has a header.  
3. The file is tab separated  
4. The first two columns are strings and the third is a number.  
  
  
Note the below points:  
  
1. We need to convert the Dataframe in the RDD ( .rdd )  
2. We will use the RDD method - getNumPartitions() to get the partitions



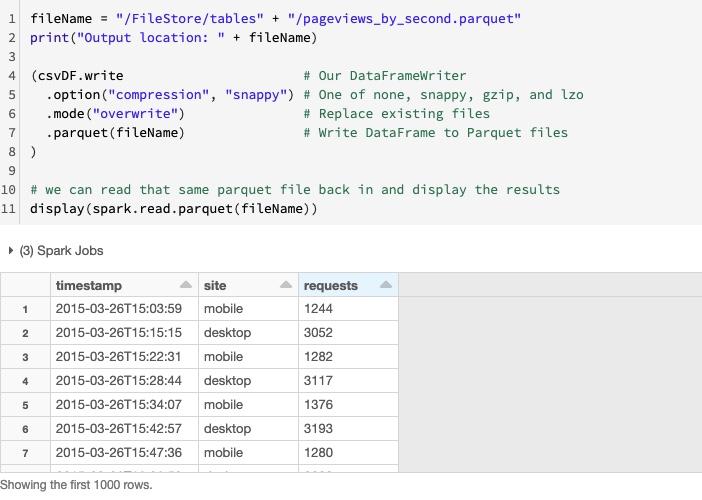
**#Question34**:  
Read The JSON File in Spark ?  
  
Much like the CSV reader, the JSON reader also assumes.  
  
1. There is one JSON object per line.  
2. it's delineated by a new-line.  
  
This format is referred to as JSON Lines or newline-delimited JSON



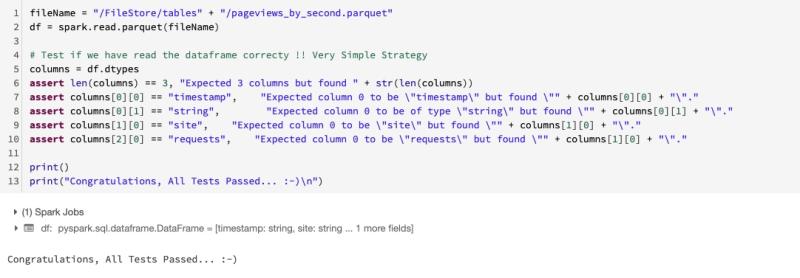
**#Question35**:  
Read The PARQUET File in Spark ?  
  
"Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language."  
  
Reading from Parquet Files:  
  
1. We do not need to specify the schema - the column names and data types are stored in the parquet files.  
2. Only one job is required to read that schema from the parquet file's metadata.  
3. Unlike the CSV or JSON readers that have to load the entire file and then infer the schema, the parquet reader can "read" the schema very quickly because it's reading that schema from the metadata.



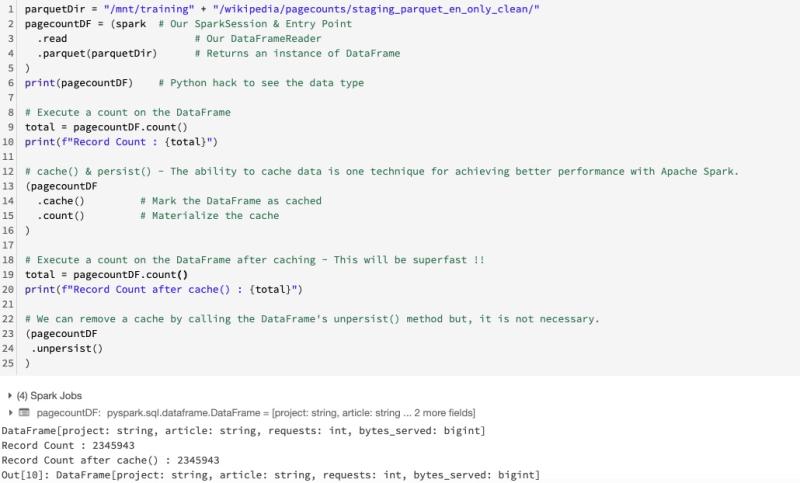
**#Question36**:  
Writing Data in PARQUET Format in Spark ?  
  
We have a DataFrame, we can write it back out as Parquet files or other various formats.



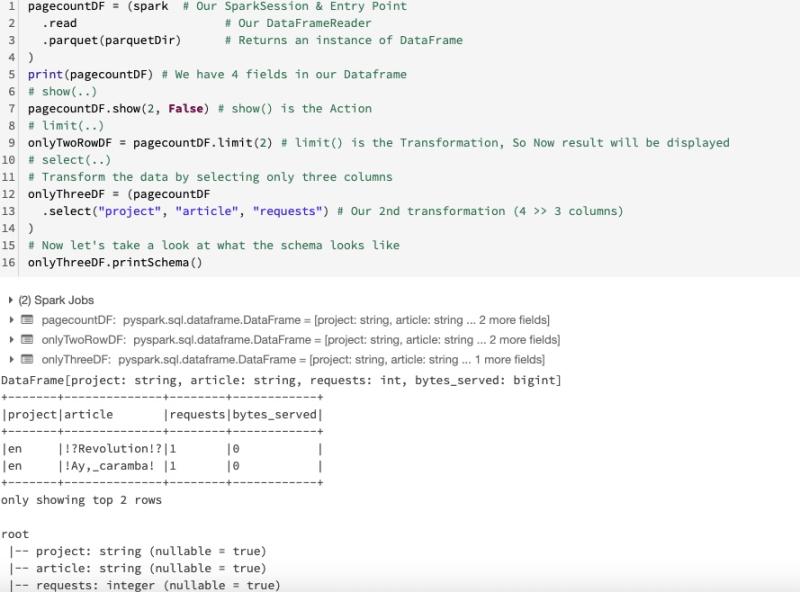
**#Question37**:  
How can we test our code in Spark ?  
  
We can have a Very Simple Strategy to test our code by using the assert method in pyspark.



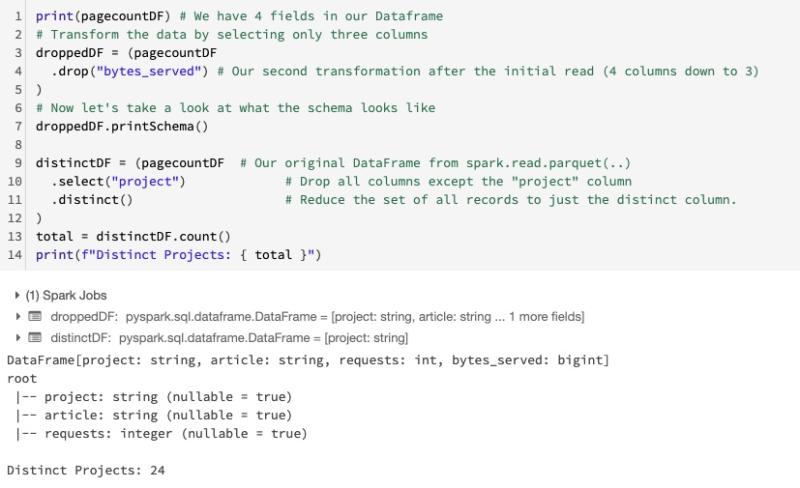
**#Question38**:  
count(), cache(), persist() & unpersist() method in Spark ?  
  
cache() & persist()  
  
The ability to cache data is one technique for achieving better performance with Apache Spark.  
  
This is because every action requires Spark to read the data from its source (Azure Blob, Amazon S3, HDFS, etc.) but caching moves that data into the memory of the local executor for "instant" access.  
  
cache() is just an alias for persist()  
  
When Caching Data you are placing it on the workers of the cluster.  
  
Caching takes resources, before moving a notebook into production please check and verify that you are appropriately using cache.  
  
unpersist()  
We can remove a cache by calling the DataFrame's unpersist() method but, it is not necessary.



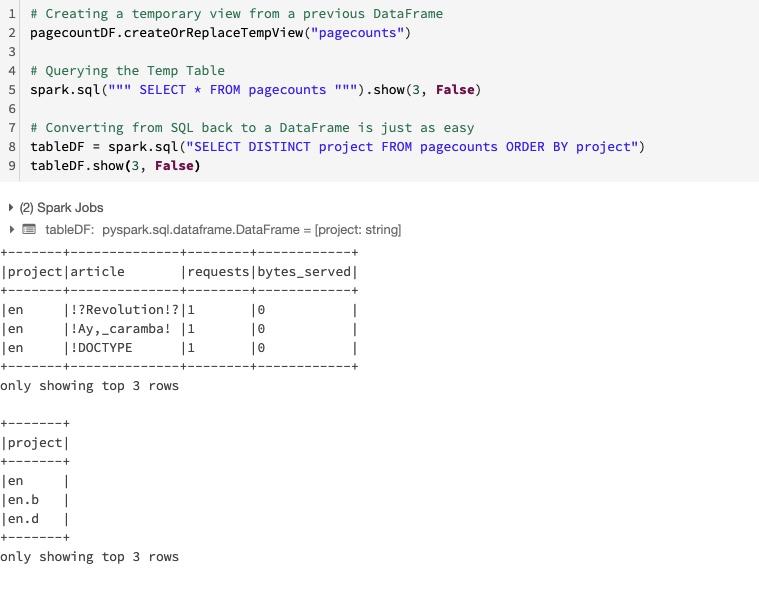
**#Question39**:  
show(), limit(), & select() method in Spark ?  
  
show()  
The show(..) method effectively has two optional parameters:  
  
1. n: The number of records to print to the console, the default being 20.  
2. truncate: If true, columns wider than 20 characters will be truncated, where the default is true.  
  
limit()  
  
1. Returns a new Dataset by taking the first n rows..  
2. limit(..) return a new DataFrame  
  
select()  
  
1. The call to select(..) does not trigger a job.  
2. That's because select(..) is a transformation.



**#Question40**:  
drop(), distinct(), & dropDuplicates() method in Spark ?  
  
drop():  
  
Returns a new Dataset with a column dropped.  
  
distinct() & dropDuplicates():  
  
These two transformations do the same thing. In fact, they are aliases for one another.  
  
You can see this by looking at the source code for these two methods.  
  
Returns a new Dataset that contains only the unique rows from this Dataset.  
  
dropDuplicates(columns...)  
The method dropDuplicates(..) has a second variant that accepts one or more columns.  
  
1. The distinction is not performed across the entire record unlike distinct() or even dropDuplicates().  
2. The distinction is based only on the specified columns.  
3. This allows us to keep all the original columns in our DataFrame.



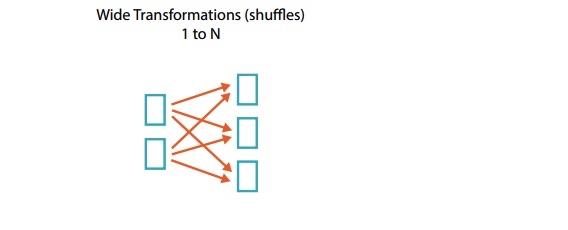
**#Question41**:  
DataFrames vs SQL & Temporary Views ?  
  
The DataFrames API is built upon an SQL engine.  
  
As such we can "convert" a DataFrame into a temporary view (or table) and then use it in "standard" SQL.



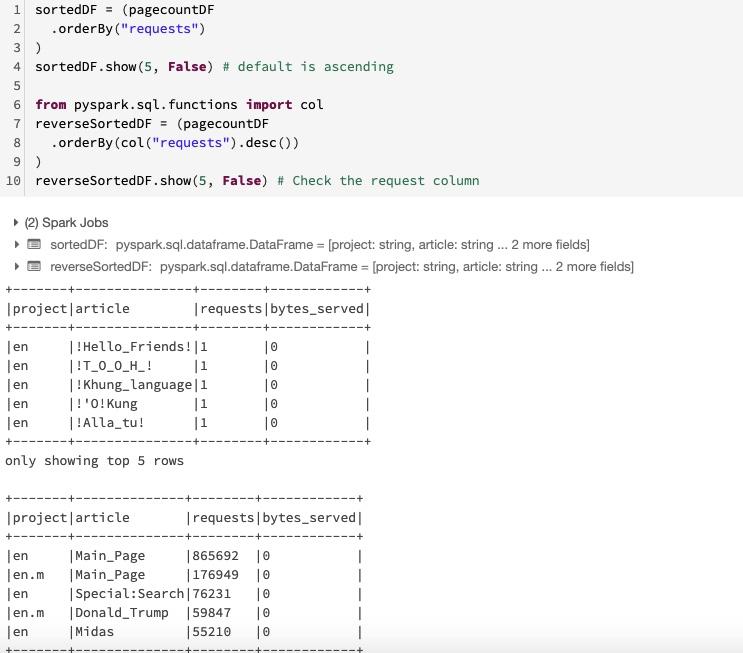
**#Question42**:  
Give some example of Narrow Transformations in Spark ?  
  
The data required to compute the records in a single partition reside in at most one partition of the parent Dataframe.  
  
Examples include:  
  
select(..)  
filter(..)  
where(..)  
drop(..)  
coalesce()



**#Question43**:  
Give some example of Wide Transformations in Spark ?  
  
The data required to compute the records in a single partition may reside in many partitions of the parent Dataframe.  
  
These operations require that data is shuffled between executors.  
  
Examples include:  
  
distinct()  
groupBy(..).sum()  
repartition(n)



**#Question44**:  
orderBy(..) & sort(..) in Spark ?  
  
sort(..) and orderBy(..) are aliases for each other.  
  
There are two variants of these two methods:  
  
orderBy(Column)  
orderBy(String)  
  
sort(Column)  
sort(String)



**#Question45**:  
filter(..) & where(..) in Spark ?  
  
filter(..) & where(..) are aliases for each other.  
  
There are two variants of these two methods:  
  
filter(Column)  
filter(String)  
  
where(Column)  
where(String)  
  
Unlike orderBy(String) which requires a column name, filter(String) and where(String) both expect an SQL expression.



**#Question46**:  
first() & head() in Spark ?  
  
first() & head() are aliases for each other.  
  
Returns the first row.



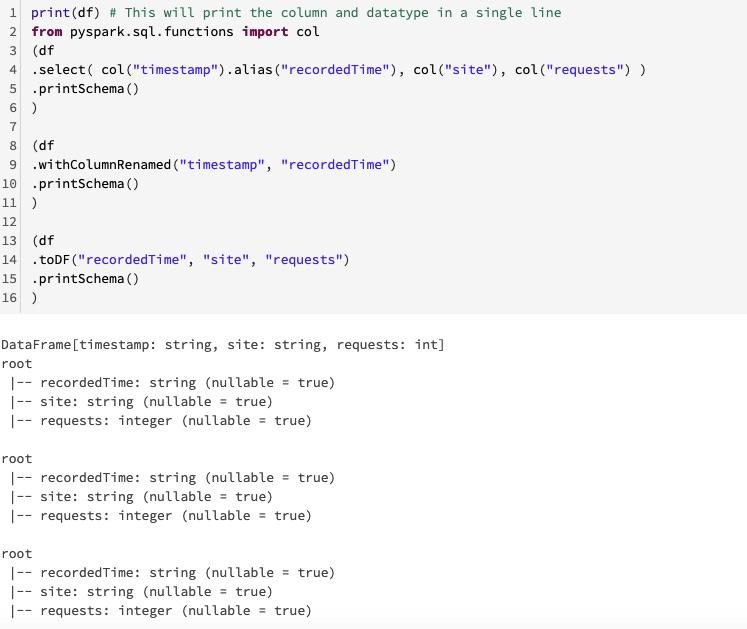
**#Question47**:  
collect() & take() in Spark ?  
  
collect()  
  
Returns an array that contains all of Rows in this Dataset.  
  
take(n)  
  
Returns the first n rows in the Dataset.  
  
it's the same basic function as collect() except you specify as the first parameter the number of records to return.



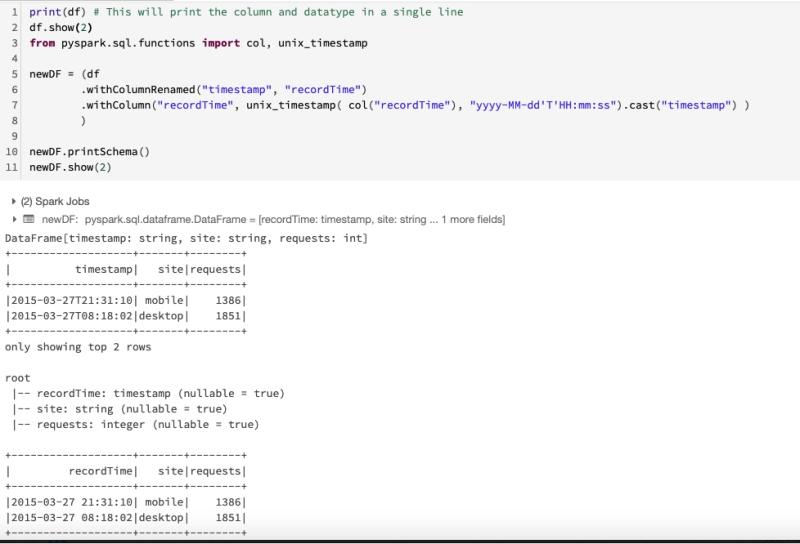
**#Question48**:  
DataFrame vs Dataset Spark ? How to convert a DataFrame to a Dataset ?  
  
Datasets are a Java and Scala concept and brings to those languages the type safety.  
  
DataFrame = Dataset[Row]  
  
Python and R have no such concept because they are dynamically typed.  
  
Note:  
  
Dataframe:  
org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]  
  
Dataset:  
org.apache.spark.sql.Dataset[CustomSchema]



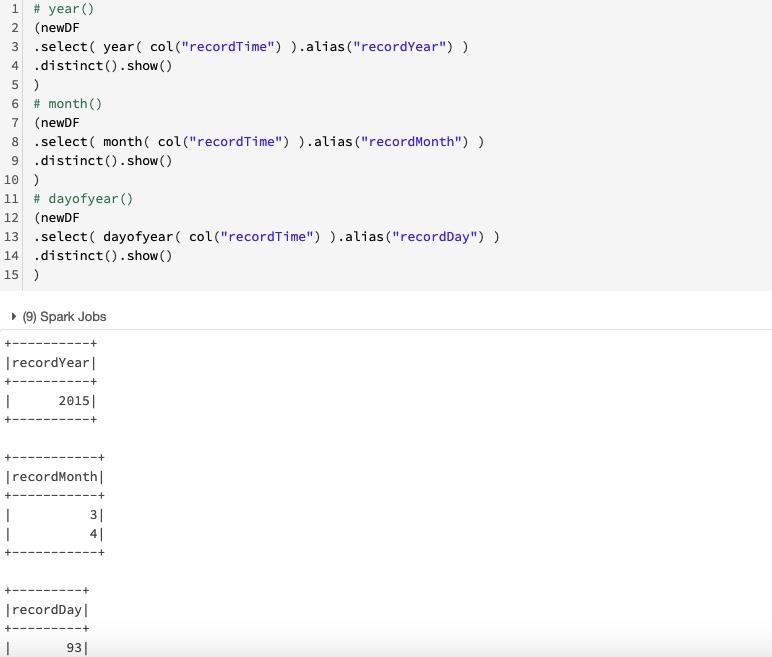
**#Question49**:  
How can we rename a column in Spark Dataframe ?  
  
1. By using col() Function with alias()  
  
2. By using withColumnRenamed() method  
  
3. By using toDF() method



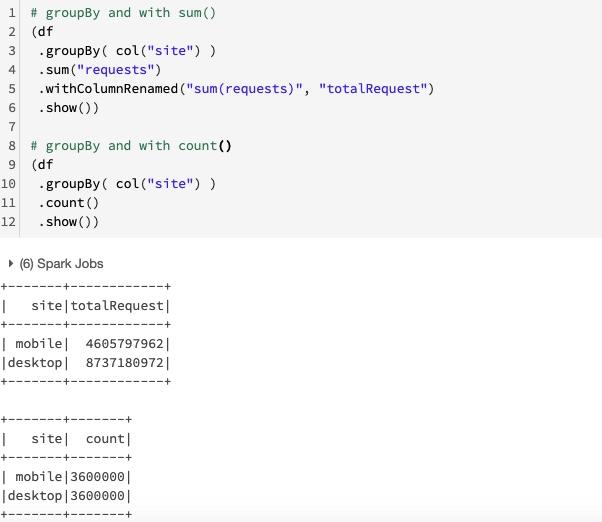
**#Question50**:  
How can we Covert a time string Column into Timestamp Column in Spark Dataframe ?  
  
By using unix\_timestamp(..) & cast(..)  
  
unix\_timestamp(..)  
Convert time string with given pattern to Unix time stamp ( in seconds), return null if fail.



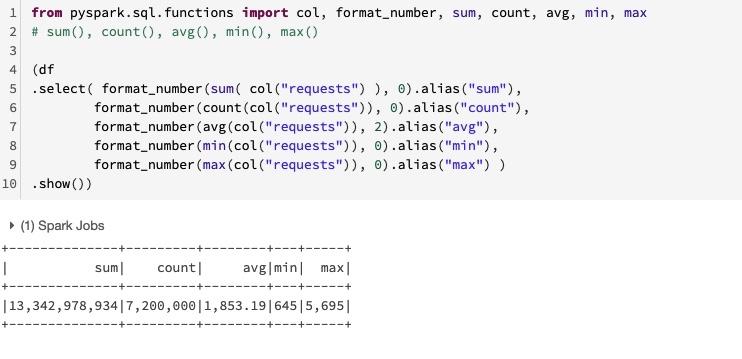
**#Question51**:  
year(..), month(..), dayofyear(..) function in Spark Dataframe ?  
  
  
year() - Gives the year from the timestamp column.  
  
month() - Gives the month from the timestamp column.  
  
dayofyear() - Gives the day of the year from the timestamp column  
  
There are many more functions like this.



**#Question52**:  
groupBy() function in Spark ?  
  
groupBy()  
  
Groups the Dataset using the specified columns, so that we can run aggregation on them.  
  
This function is a wide transformation - it will produce a shuffle and conclude a stage boundary.  
  
It supports the following aggregations:  
  
avg(..) Compute the mean value for each numeric columns for each group.  
  
count(..) Count the number of rows for each group.  
  
sum(..) Compute the sum for each numeric columns for each group.  
  
min(..) Compute the min value for each numeric column for each group.  
  
max(..) Compute the max value for each numeric columns for each group.  
  
mean(..) Compute the average value for each numeric columns for each group.  
  
agg(..). Compute aggregates by specifying a series of aggregate columns.  
  
pivot(..) Pivots a column of the current DataFrame and perform the specified aggregation.



**#Question53**:  
sum(), count(), avg(), min(), max() function in Spark ?  
  
Aggregating data is one of the more common tasks when working with big data.  
  
**sql.functions** provides lots of these functions like - sum(), count(), avg(), min(), max()



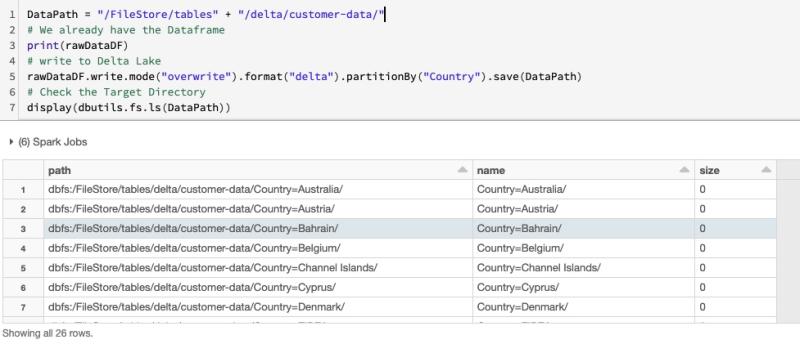
**#Question54**:  
What is DATA Lake ? list down some of the challenge with data lakes ?  
  
A data lake is a storage repository that inexpensively stores a vast amount of raw data, both current and historical, in native formats such as XML, JSON, CSV, and Parquet.  
  
It may also contain operational relational databases with live transactional data.  
  
But the data is not ready for data science & ML. The majority of these projects are failing due to unreliable data!  
  
Operational Challenges:  
1. Schema enforcement when new tables are introduced.  
2. Table repairs when any new data is inserted into the data lake.  
3. Frequent refreshes of metadata.  
4. Bottlenecks of small file sizes for distributed computations.  
5. Difficulty sorting data by an index if data is spread across many files and partitioned.  
  
Performance challenges:  
1. Too many small or very big files - more time opening & closing files rather than reading contents (worse with streaming).  
2. Partitioning - breaks down if you picked the wrong fields or when data has many dimensions, high cardinality columns.  
3. No caching - cloud storage throughput is low ( cloud object storage is 20-50MB/s/core vs 300MB/s/core for local SSDs ).



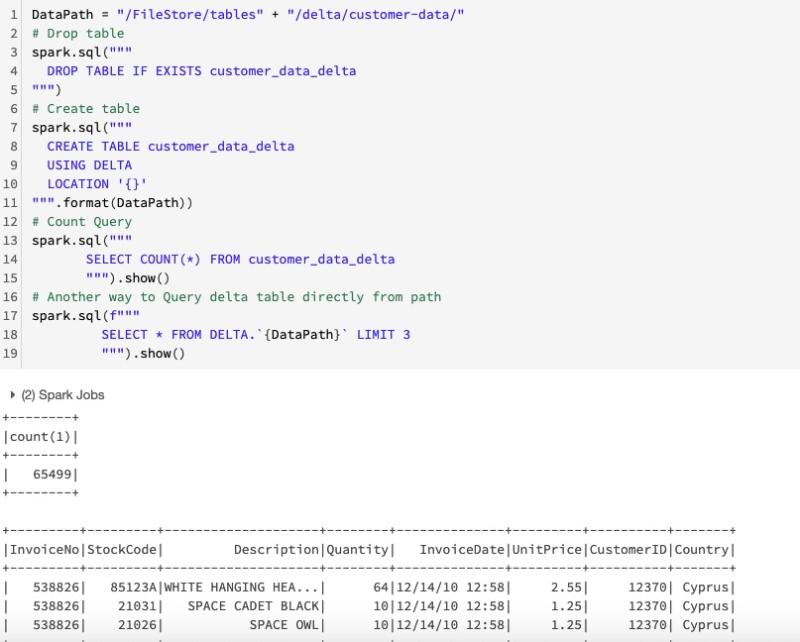
**#Question55**:  
What is Delta Lake ?  
  
Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.  
  
Delta Lake is a file format that can help you build a data lake comprised of one or many tables in Delta Lake format. Delta Lake integrates tightly with Apache Spark, and uses an open format that is based on Parquet.  
  
Delta Lake provides the following functionality:  
  
1. ACID Transactions  
2. Scalable Metadata Handling:  
3. Time Travel (data versioning)  
4. Open Format  
5. Unified Batch and Streaming Source and Sink  
6. Schema Enforcement  
7. Schema Evolution  
8. 100% Compatible with Apache Spark API



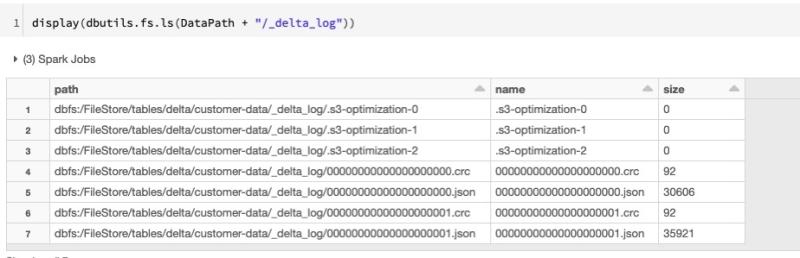
**#Question56**:  
How to write the file in DELTA format from spark dataframe ?  
  
Creating Delta Lakes is as easy as changing the file type while performing a write.  
  
instead of - "Parquet"  
just use - "delta"



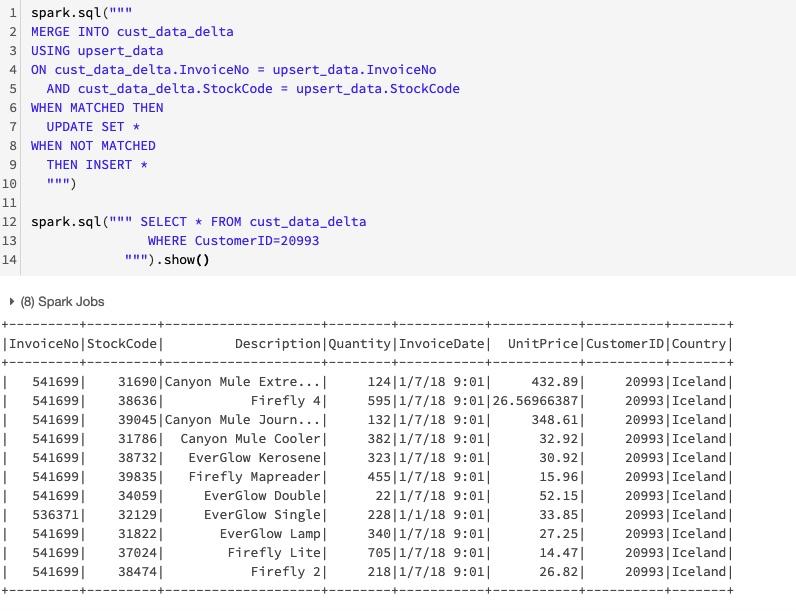
**#Question57**:  
CREATE A Table Using Delta Lake ? and Querying the delta lake?  
  
The notation is:  
  
CREATE TABLE  
USING DELTA  
LOCATION  
  
Tables created with a specified LOCATION are considered unmanaged by the metastore.  
  
Unlike a managed table, where no path is specified, an unmanaged table’s files are not deleted when you DROP the table.  
  
However, changes to either the registered table or the files will be reflected in both locations.



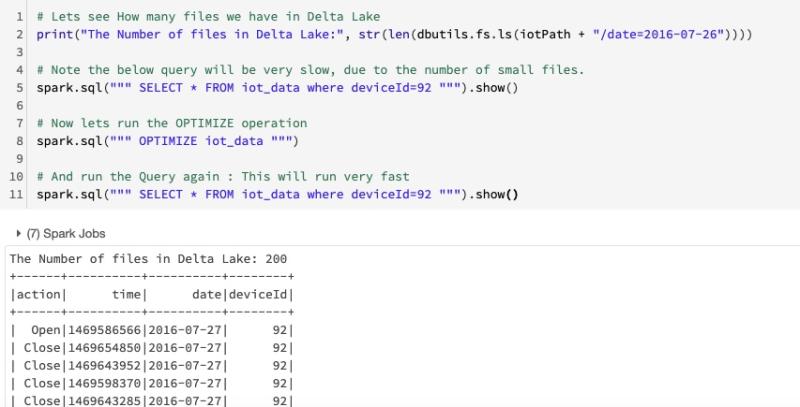
**#Question58**:  
Metadata in Delta Lake ?  
  
The table in the Hive metastore automatically inherits the schema, partitioning, and table properties of the existing data.  
  
Note that we only store table name, path, database info in the Hive metastore, the actual schema is stored in the \_delta\_log



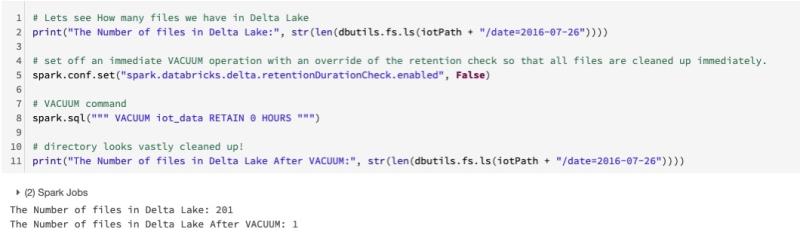
**#Question59**:  
Delta Lake Upsert Operations (MERGE)?  
  
UPSERT means to "UPdate" and "inSERT". In other words, UPSERT is literally TWO operations.  
  
It is not supported in traditional data lakes, as running an UPDATE could invalidate data that is accessed by the subsequent INSERT operation.  
  
Using Delta Lake, however, we can do UPSERTS. Delta Lake combines these operations to guarantee atomicity to  
  
1. INSERT a row  
2. if the row already exists, UPDATE the row.



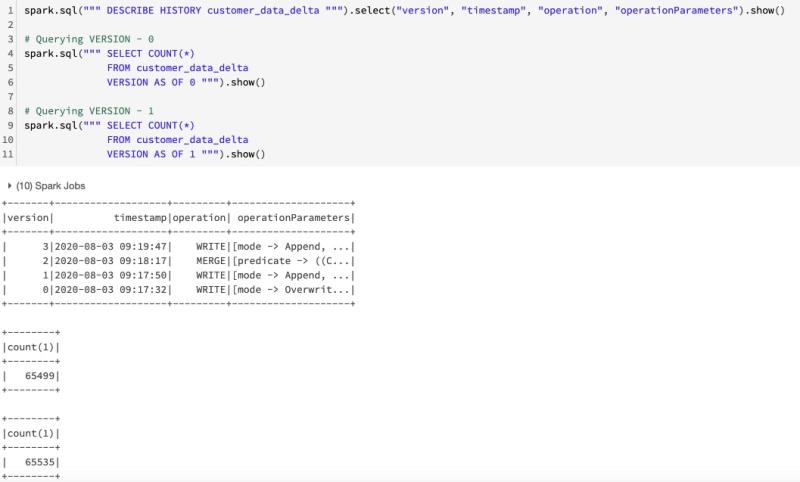
**#Question60**:  
What is the SMALL FILE PROBLEM in Big Data ? How Delta Lake addresses it ( OPTIMIZE ) ?  
  
Historical and new data is often written in very small files and directories. This data may be spread across a data center or even across the world (that is, not co-located).  
The result is that a query on this data may be very slow due to  
  
1. network latency  
2. volume of file metatadata  
  
The solution is to compact many small files into one larger file. Delta Lake has a mechanism for compacting small files.  
  
OPTIMIZE  
Delta Lake supports the OPTIMIZE operation, which performs file compaction.  
  
Small files are compacted together into new larger files up to 1GB. Thus, at this point the number of files increases!  
  
The 1GB size was determined by the Databricks optimization team as a trade-off between query speed and run-time performance when running Optimize.



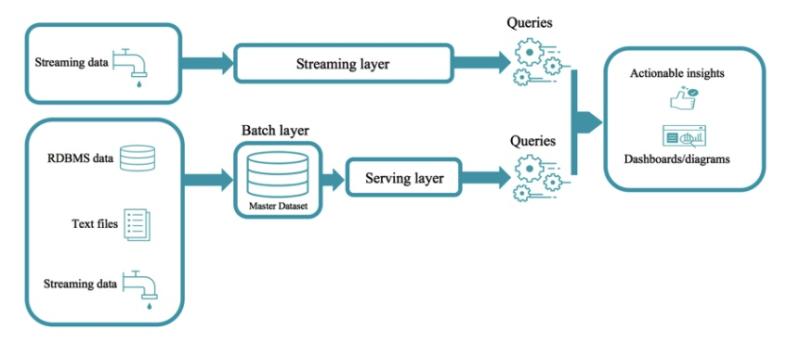
**#Question61**:  
Delta Lake VACUUM Operation ?  
  
To save on storage costs you should occasionally clean up invalid files using the VACUUM command.  
  
Invalid files are small files compacted into a larger file with the OPTIMIZE command.  
  
The syntax of the VACUUM command is  
  
VACUUM name-of-table RETAIN number-of HOURS;



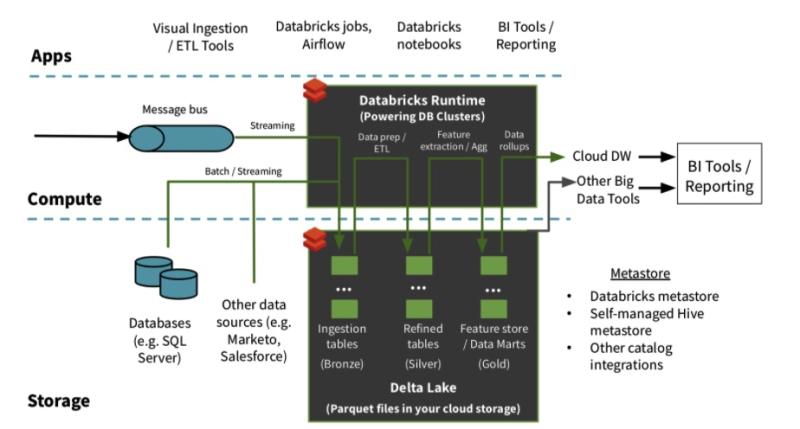
**#Question62**:  
Delta Lake TIME TRAVEL Operation ?  
  
Because Delta Lake is version controlled, We have the option to query past versions of the data by looking at the history of our current Delta table.  
  
Querying an older version is as easy as adding VERSION AS OF desired\_version



**#Question63**:  
What is the Lambda architecture ?  
  
The lambda architecture is a big data processing architecture which combines both batch- and real-time processing methods.  
  
It features an append-only immutable data source that serves as system of record. Timestamped events are appended to existing events (nothing is overwritten).  
  
Data is implicitly ordered by time of arrival.  
  
There are basically 2 pipelines, one batch and one streaming, hence the name lambda architecture.  
  
Drawaback :  
It is difficult to combine processing of batch and real-time data.



**#Question64**:  
What is the Delta Lake Architecture ?  
  
The Delta Lake Architecture is a huge improvement upon the traditional Lambda architecture.  
  
At each stage, we enrich our data through a unified pipeline that allows us to combine batch and streaming workflows through a shared filestore with ACID-compliant transactions.  
  
Bronze tables - contain raw data ingested from various sources (JSON files, RDBMS data, IoT data, etc.).  
  
Silver tables - will provide a more refined view of our data. We can join fields from various bronze tables to enrich streaming records, or update account statuses based on recent activity.  
  
Gold tables - provide business level aggregates often used for reporting and dashboarding. This would include aggregations such as daily active website users, weekly sales per store, or gross revenue per quarter by department.  
  
The end outputs are actionable insights, dashboards, and reports of business metrics.  
  
By considering our business logic at all steps of the extract-transform-load (ETL) pipeline, we can ensure that storage and compute costs are optimized by reducing unnecessary duplication of data and limiting ad hoc querying against full historic data.



**#Question65**:  
Reading Streaming Data in Spark ?  
  
Reading a Stream  
The method **SparkSession.readStream** returns a DataStreamReader used to configure the stream.  
  
There are a number of key points to the configuration of a DataStreamReader:  
  
1. The schema  
2. The type of stream: Files, Kafka, TCP/IP, etc  
3. Configuration specific to the type of stream  
  
For files, the file type, the path to the files, max files, etc...  
  
For TCP/IP the server's address, port number, etc...  
  
For Kafka the server's address, port, topics, partitions, etc...



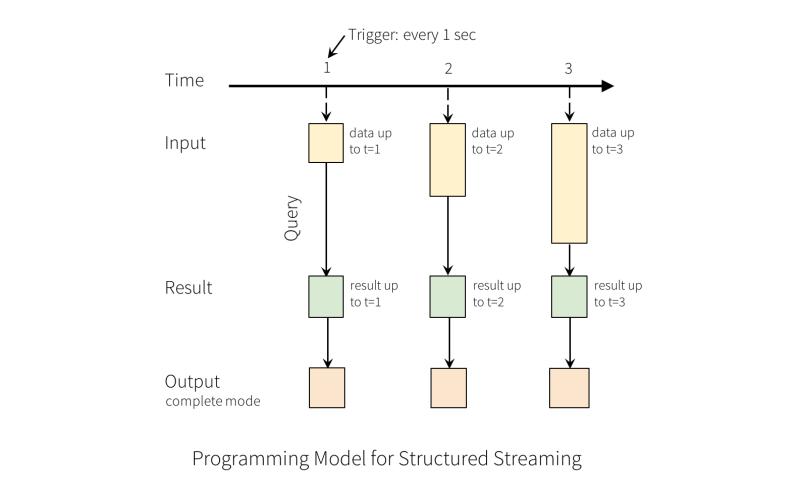
**#Question66**:  
Why must a schema be specified for a streaming DataFrame ? Or Why are streaming DataFrames unable to infer/read a schema?  
  
If you have enough data, you can infer the schema.  
  
If you don't have enough data you run the risk of miss-inferring the schema.  
  
For example, you think you have all integers but the last value contains "1.123" (a float) or "foo" (a string).  
  
With a stream, we have to assume we don't have enough data because we are starting with zero records. ( The data yet not arrived )  
  
And unlike reading from a table or parquet file, there is nowhere from which to "read" the stream's schema.  
  
For this reason, we must specify the schema manually.



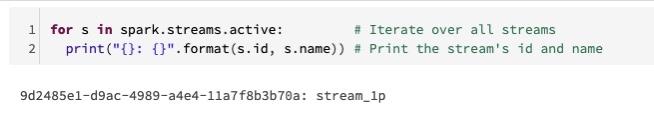
**#Question67**:  
Writing Streaming Data in Spark ?  
  
Writing a Stream  
  
The method **DataFrame.writeStream** returns a DataStreamWriter used to configure the output of the stream.  
  
There are a number of parameters to the DataStreamWriter configuration:  
  
Query's name (optional) - This name must be unique among all the currently active queries in the associated SQLContext.  
  
Trigger (optional) - Default value is ProcessingTime(0) and it will run the query as fast as possible.  
  
Checkpointing directory (optional for pup/sub sinks)  
  
Output mode  
  
Output sink  
  
Configuration specific to the output sink, such as:  
  
1. The host, port and topic of the receiving Kafka server  
  
2. The file format and final destination of files  
  
3. A custom sink via **writeStream.foreach**(...)  
  
Once the configuration is completed, we can trigger the job with a call to .start()



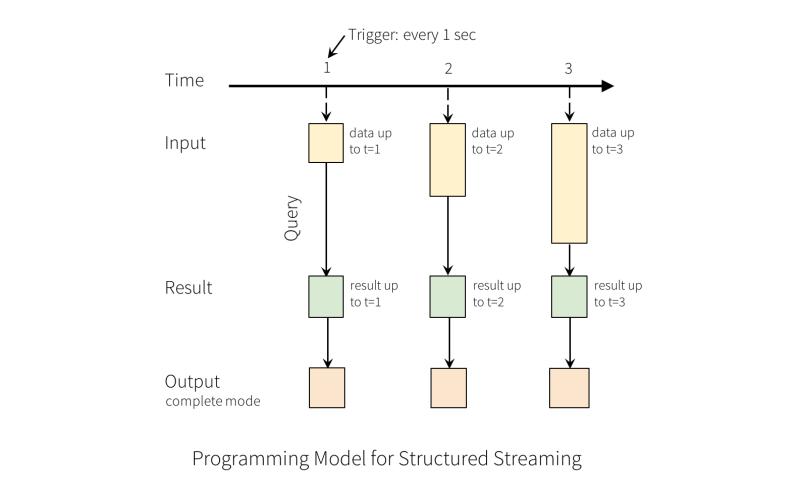
**#Question68**:  
Explain CHECKPOINTING in Structured Streaming in Spark ?  
  
Checkpointing  
  
A checkpoint stores the current state of your streaming job to a reliable storage system such as Azure Blob Storage or HDFS.  
It does not store the state of your streaming job to the local file system of any node in your cluster.  
  
Together with write ahead logs, a terminated stream can be restarted and it will continue from where it left off.  
  
To enable this feature, you only need to specify the location of a checkpoint directory:  
.option("checkpointLocation", checkpointPath)  
  
Note the below points carefully:  
  
1. If you do not have a checkpoint directory, when the streaming job stops, you lose all state around your streaming job and upon restart, you start from scratch.  
  
2. For some sinks, you will get an error if you do not specify a checkpoint directory:  
analysisException: 'checkpointLocation must be specified either through option("checkpointLocation", ...)..  
  
3. Also note that every streaming job should have its own checkpoint directory: No sharing.



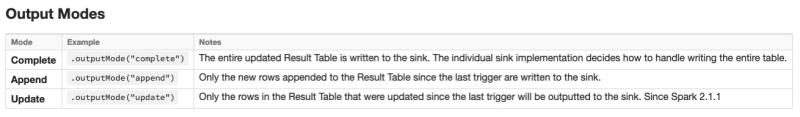
**#Question69**:  
How do we view the list of active streams in Spark ?  
  
We can use - **spark.streams.active.**



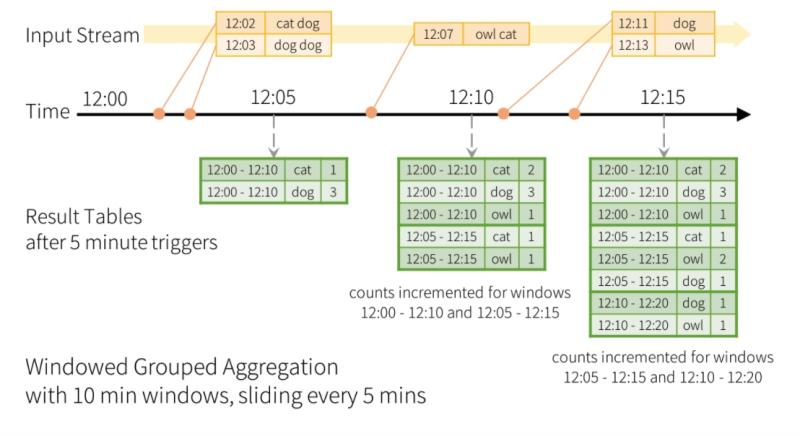
**#Question70**:  
End-to-end Fault Tolerance in Spark Structured Streaming ?  
  
Structured Streaming ensures end-to-end exactly-once fault-tolerance guarantees through checkpointing and Write Ahead Logs.  
  
Structured Streaming sources, sinks, and the underlying execution engine work together to track the progress of stream processing.  
  
If a failure occurs, the streaming engine attempts to restart and/or reprocess the data.  
  
This approach only works if the streaming source is replayable. To ensure fault-tolerance, Structured Streaming assumes that every streaming source has offsets, akin to:  
  
Kafka message offsets  
Event Hubs offsets  
  
At a high level, the underlying streaming mechanism relies on a couple approaches:  
  
1. Structured Streaming uses checkpointing and write-ahead logs to record the offset range of data being processed during each trigger interval.  
  
2. The streaming sinks are designed to be idempotent—that is, multiple writes of the same data (as identified by the offset) do not result in duplicates being written to the sink.  
  
Taken together, replayable data sources and idempotent sinks allow Structured Streaming to ensure end-to-end, exactly-once semantics under any failure condition.



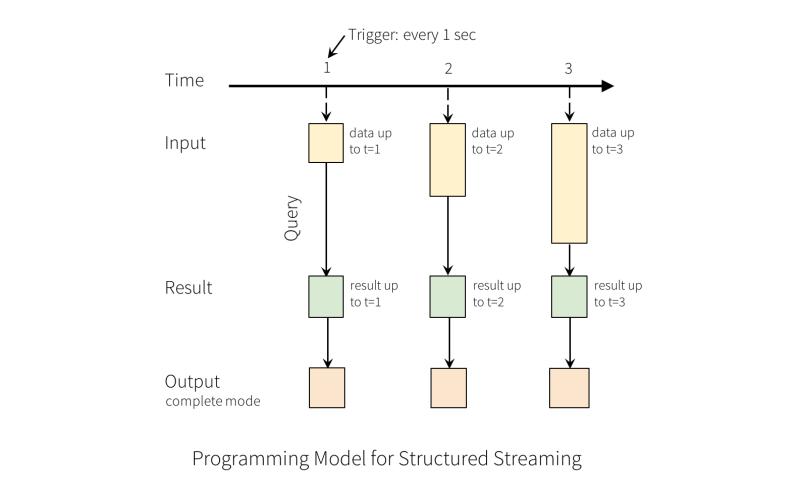
**#Question71**:  
When We do a write stream command, what does this option do - outputMode("append") ?  
  
This option takes on the following values and their respective meanings:  
  
append: add only new records to output sink  
  
complete: rewrite full output - applicable to aggregations operations  
  
update: update changed records in place



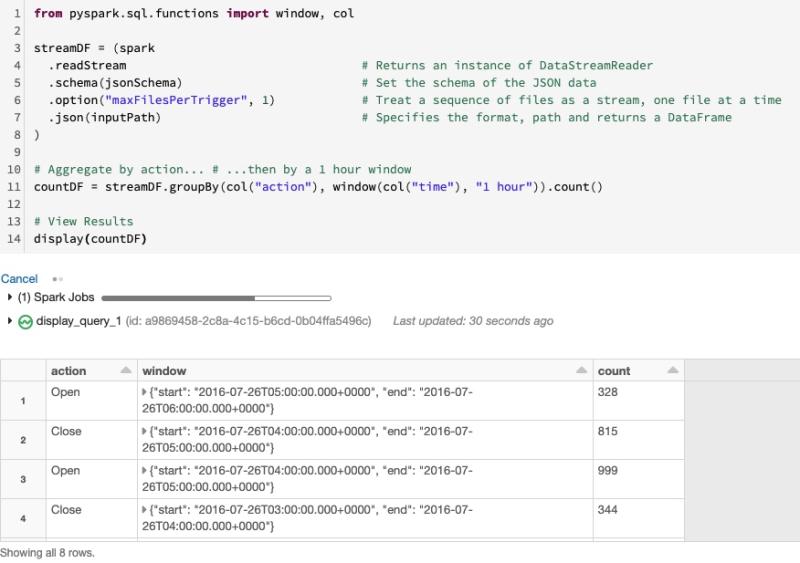
**#Question72**:  
Windowing in Spark Structured Streaming ?  
  
Windowing  
  
If we were using a static DataFrame to produce an aggregate count, we could use groupBy() and count().  
  
Instead we accumulate counts within a sliding window, answering questions like "How many records are we getting every second?"  
  
Sliding windows:  
  
The windows overlap and a single event may be aggregated into multiple windows.  
  
Tumbling Windows:  
  
The windows do not overlap and a single event will be aggregated into only one window.



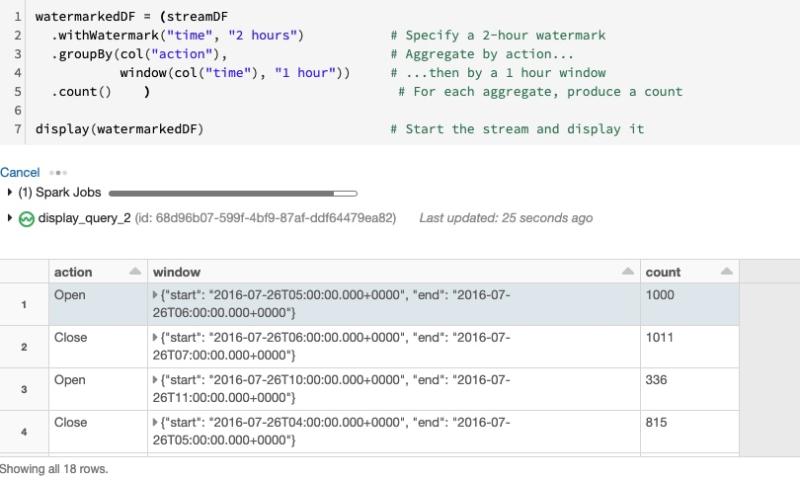
**#Question73**:  
Event Time vs Receipt Time in Spark Structured Streaming ?  
  
Event Time is the time at which the event occurred in the real world.  
  
Event Time is NOT something maintained by the Structured Streaming framework.  
  
Examples of Event Time:  
  
1. The timestamp recorded in each record of a log file  
2. The instant at which an IoT device took a measurement  
3. The moment a REST API received a request  
  
Examples of Receipt Time:  
  
1. A timestamp added to a DataFrame the moment it was processed by Spark  
2. The timestamp extracted from an hourly log file's file name  
  
Problem with using Receipt Time is going to be with accuracy. For example:  
  
1. The time between when an IoT device takes a measurement vs when it is reported can be off by several minutes.  
  
This could have significant ramifications to security and health devices, for example  
The timestamp embedded in an hourly log file can be off by up to one hour making correlations to other events extremely difficult  
  
2. The timestamp added by Spark as part of a DataFrame transformation can be off by hours to weeks to months depending on when the event occurred and when the job ran



**#Question74**:  
Windowed Streaming Example in Spark Structured Streaming?  
  
With the schema defined, we can create the initial DataFrame streamDF and then countsDF which represents our aggregation:  
  
To view the results of our query, pass the DataFrame countsDF to the display() function.



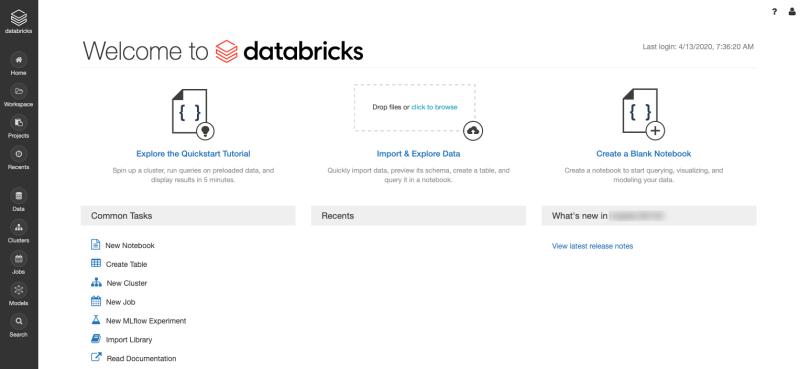
**#Question75**:  
Watermarking in Spark Structured Streaming?  
  
Problem with Generating Many Windows:  
  
In the last example, We are generating a window for every 1 hour aggregate.  
  
Every window has to be separately persisted and maintained.  
  
Over time, this aggregated data will build up in the driver.  
  
The end result being a massive slowdown if not an OOM Error.  
  
How do we fix that problem?  
  
One simple solution is to increase the size of our window (say, to 2 hours).  
  
That way, we're generating fewer windows.  
  
But if the job runs for a long time, we're still building up an unbounded set of windows.  
  
Eventually, we could hit resource limits.  
  
Watermarking  
  
A better solution to the problem is to define a cut-off.  
  
A point after which Structured Streaming will commit windowed data to sink, or throw it away if the sink is console or memory  
  
Example Details  
In the example below:  
1. Data received 2 hour past the watermark will be dropped.  
2. Data received within 2 hours of the watermark will never be dropped.  
  
More specifically, any data less than 2 hours behind the latest data processed till then is guaranteed to be aggregated.



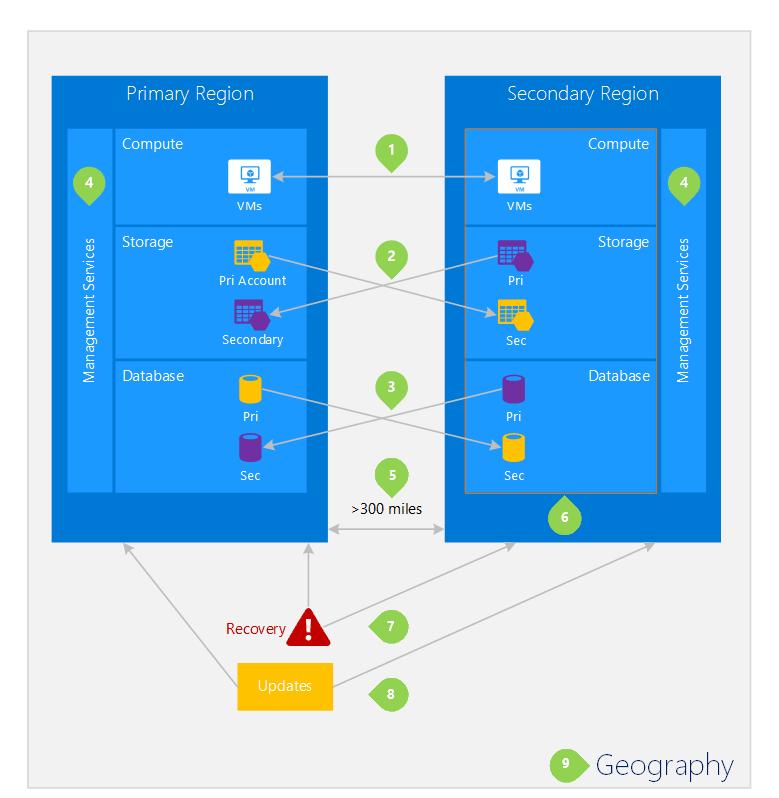
**#Question76**:  
STOP all streams in Spark Structured Streaming?  
  
We can use .stop() to stop all the active streams



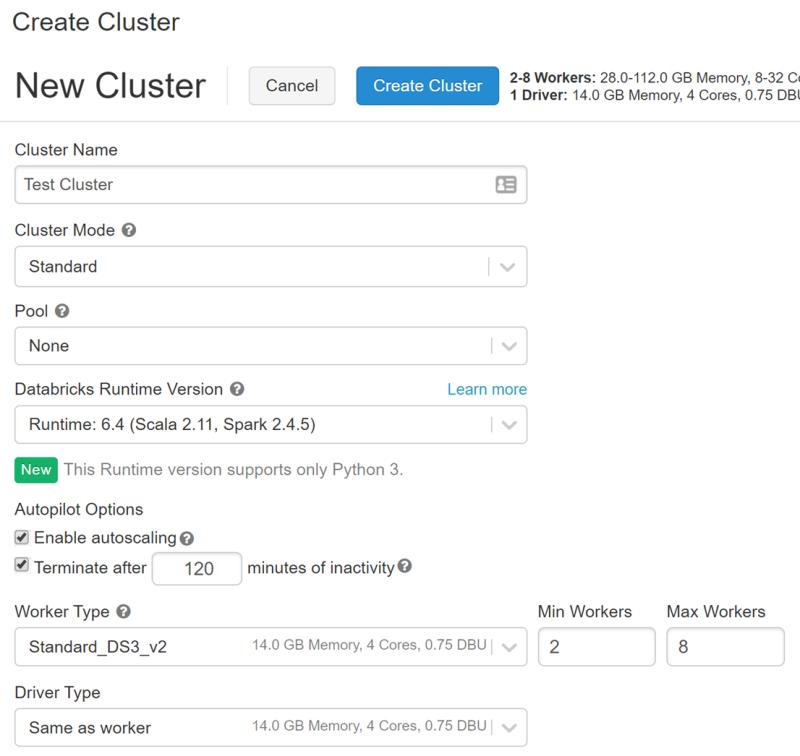
**#Question77**:  
Databricks workspace limits ?  
  
Key workspace limits are:  
  
1. The maximum number of jobs that a workspace can create in an hour is 1000  
  
2. At any time, you cannot have more than 150 jobs simultaneously running in a workspace  
  
3. There can be a maximum of 150 notebooks or execution contexts attached to a cluster  
  
4. There can be a maximum of 1500 Azure Databricks API calls/hour



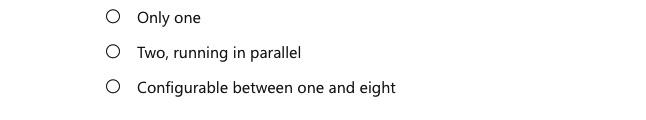
**#Question78**:  
High availability / Disaster recovery (HA/DR) in Databricks ?  
  
Within each subscription, consider the following best practices for HA/DR:  
  
1. Deploy Azure Databricks in two paired Azure regions, ideally mapped to different control plane regions.  
  
For example, East US2 and West US2 will map to different control planes  
  
Whereas West and North Europe will map to same control plane  
  
2. Use Azure Traffic Manager to load balance and distribute API requests between two deployments, when the platform is primarily being used in a backend non-interactive mode.



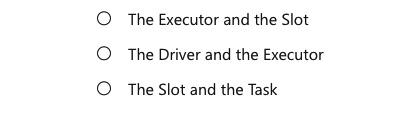
**#Question79**:  
CLUSTER in Databricks ?  
  
There are two types of Databricks clusters, according to how they are created.  
  
Clusters created using UI and Clusters API are called Interactive Clusters, whereas those created using Jobs API are called Jobs Clusters.  
  
Further, each cluster can be of two modes:  
  
1. Standard  
2. High Concurrency.  
  
Regardless of types or mode, all clusters in Azure Databricks can automatically scale to match the workload, using a feature known as Autoscaling.



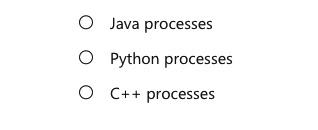
**#Question80**:  
How many drivers does a Databricks Cluster have ?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**#Question81**:  
Spark is a distributed computing environment. Therefore, work is parallelized across executors. At which two levels does this parallelization occur?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



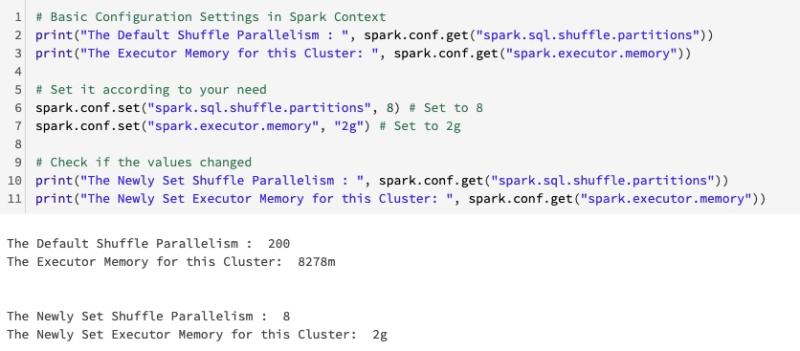
**#Question82**:  
What type of process are the driver and the executors?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



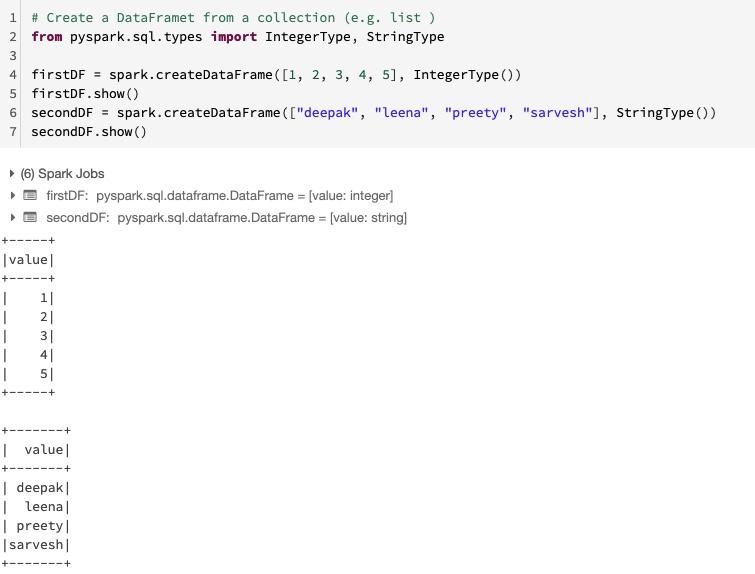
**#Question83**:  
What are the new features in Apache Spark 3.0?  
  
Here are the biggest new features in Spark 3.0:  
  
1. 2x performance improvement on TPC-DS over Spark 2.4, enabled by adaptive query execution ( AQE ), dynamic partition pruning ( DPP) and other optimizations  
  
2. ANSI SQL compliance  
  
3. Significant improvements in pandas APIs, including Python type hints and additional pandas UDFs  
  
4. Better Python error handling, simplifying PySpark exceptions  
  
5. New UI for structured streaming



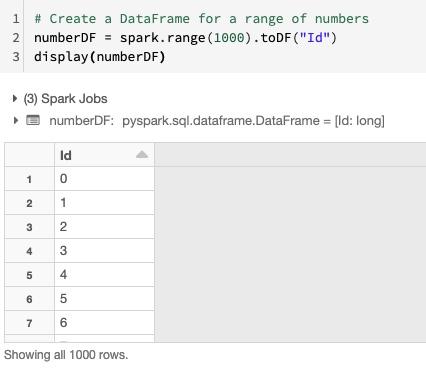
**#Question84**:  
How to get and set Basic Configuration Settings in Spark Context ?  
  
We can get / set configuration variables :  
  
**spark.sql.shuffle.partitions**  
**spark.executor.memory**



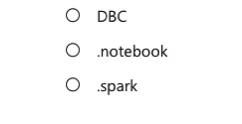
**#Question85**:  
Create a Spark DataFrame from a collection (e.g. list ) ?  
  
We can create Dataframe from the function -  
createDataFrame



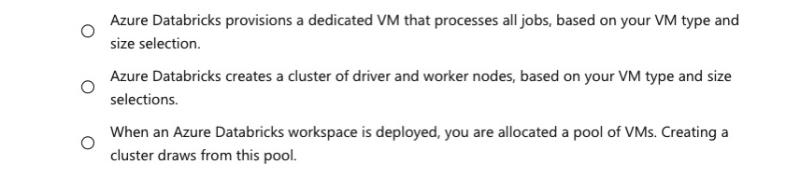
**#Question86**:  
Create a DataFrame for a range of numbers ?  
  
We can create Dataframe from the function -  
range()



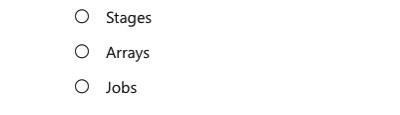
**#Question87**:  
Which notebook format is used in Databricks ?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**#Question88**:  
When creating a new cluster in the Azure Databricks workspace, what happens behind the scenes ?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



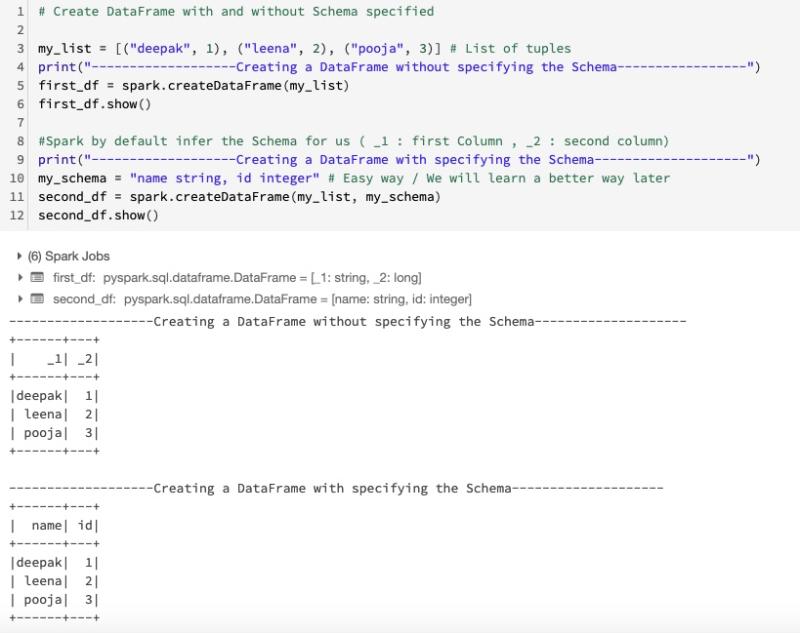
**#Question89**:  
To parallelize work, the unit of distribution is a Spark Cluster. Every Cluster has a Driver and one or more executors. Work submitted to the Cluster is split into what type of object ?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



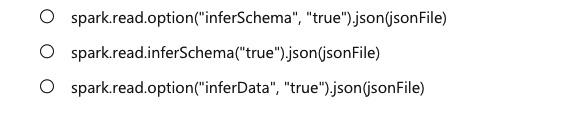
**#Question90**:  
Create DataFrame from Row Object ?  
  
1. Import - from **pyspark.sql** import Row  
2. Create Row objects and use function - createDataFrame



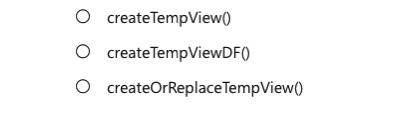
**#Question91**:  
Create DataFrame with and without Schema Specified ?  
  
1. Create rows froms Tuples  
2. use createDataFrame function without specifying the schema  
3. use createDataFrame function with specifying the schema



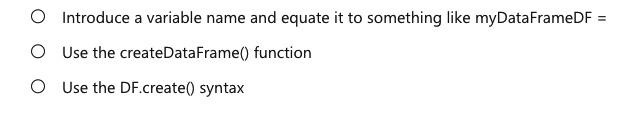
**#Question92**:  
How do you infer the data types and column names when you read a JSON file?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



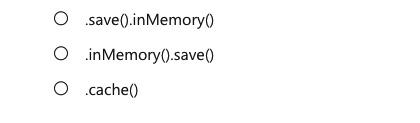
**#Question93**:  
Which DataFrame method do you use to create a temporary view?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



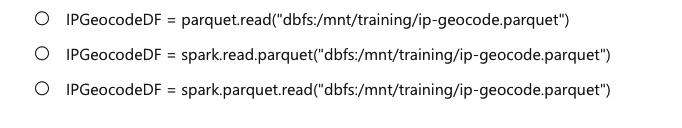
**#Question94**:  
How do you create a DataFrame object?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



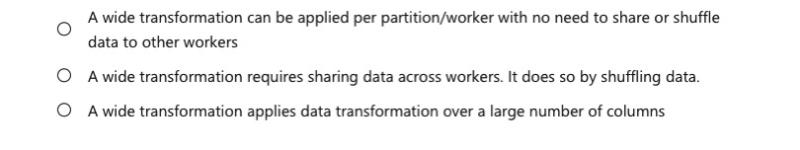
**#Question95**:  
How do you cache data into the memory of the local executor for instant access?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



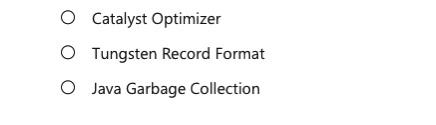
**#Question96**:  
What is the Python syntax for defining a DataFrame in Spark from an existing Parquet file in DBFS?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**#Question97**:  
Which of the following statements describes a wide transformations?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



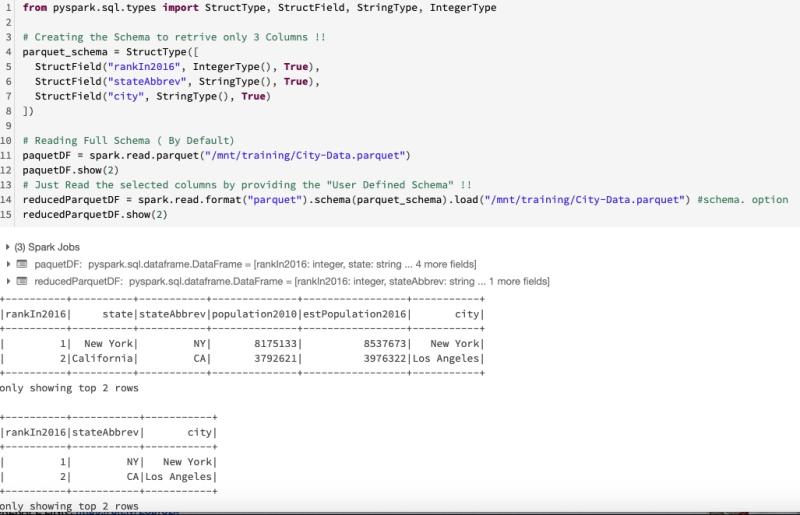
**#Question98**:  
Which feature of Spark determines how your code is executed?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



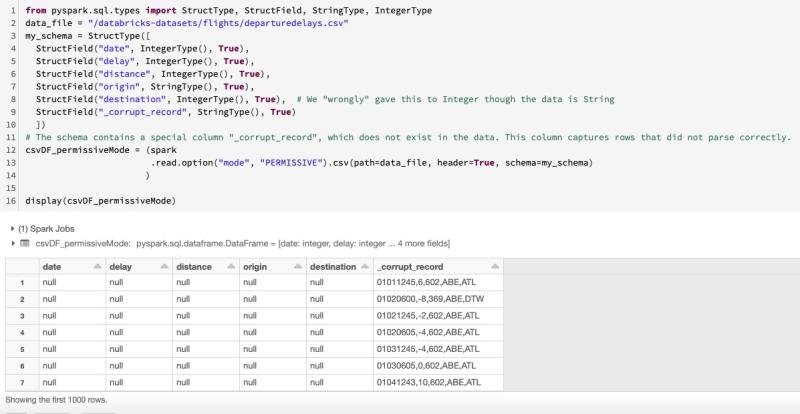
**#Question99**:  
If you create a DataFrame that will read some data from Azure Blob Storage, and then you create another DataFrame by filtering the initial DataFrame. What feature of Spark causes these transformation to be analyzed?  
  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**#Question100**:  
Reading PARQUET file in Spark - Read all the COLUMNS & Read few selected COLUMNS?  
  
  
1. read the parquet file from - **spark.read.parquet**  
2. Create the schema for reduced Dataframe  
3. use the .schema option



**#Question101**:  
PERMISSIVE mode (default) in Spark ?  
  
When reading CSV files with a specified schema, it is possible that the actual data in the files does not match the specified schema.  
  
For example, a field containing name of the city will not parse as an integer. The consequences depend on the mode that the parser runs in:  
  
PERMISSIVE (default): nulls are inserted for fields that could not be parsed correctly  
  
In the PERMISSIVE mode it is possible to inspect the rows that could not be parsed correctly. To do that, you can add \_corrupt\_record column to the schema.



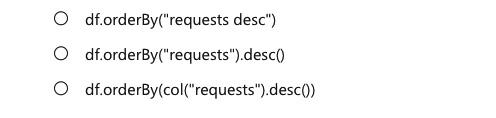
**#Question102**:  
DROPMALFORMED mode in Spark ?  
  
When reading CSV files with a specified schema, it is possible that the actual data in the files does not match the specified schema.  
  
For example, a field containing name of the city will not parse as an integer. The consequences depend on the mode that the parser runs in:  
  
DROPMALFORMED: drops lines that contain fields that could not be parsed



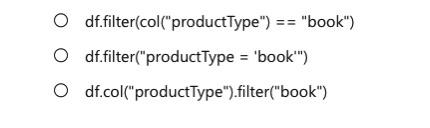
**#Question103**:  
FAILFAST mode in Spark ?  
  
When reading CSV files with a specified schema, it is possible that the actual data in the files does not match the specified schema.  
  
For example, a field containing name of the city will not parse as an integer. The consequences depend on the mode that the parser runs in:  
  
FAILFAST: aborts the reading if any malformed data is found



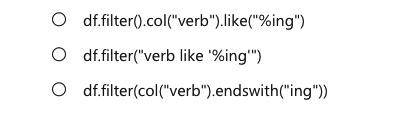
**#Question104**:  
Which command orders by a column in descending order?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



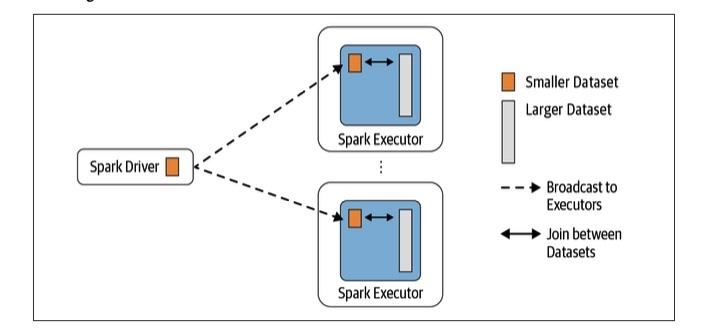
**#Question105**:  
Which command specifies a column value in a DataFrame's filter? Specifically, filter by a productType column where the value is equal to book ?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



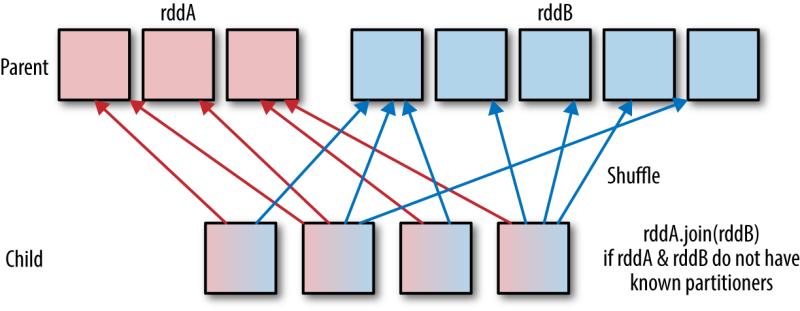
**#Question106**:  
When using the Column Class, which command filters based on the end of a column value? For example, a column named verb and filtered by words ending with "ing" ?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



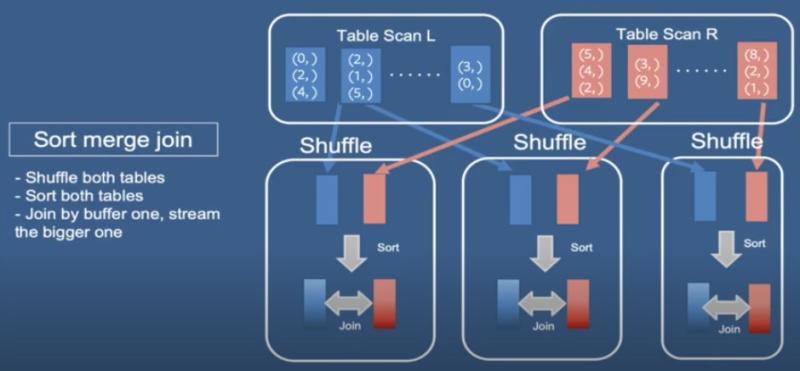
**#Question107**:  
Broadcast Hash Join in Spark ?  
  
When one of the dataset is small enough to fit in the memory, it is broadcasted over to all the executors where the larger dataset resides and a hash join is performed.  
  
There are two phases in it:  
  
1. Broadcast: Using a Spark broadcast variable, The smaller dataset is broadcasted across the executors in the cluster where the larger table is located.  
  
2. Hash Join: A standard hash join is performed on each executor.  
  
By default Spark will use a broadcast join if the smaller dataset is less than 10 MB ( 10485760 ) . This configuration is set in  
  
spark.sql.autoBroadcastJoinThreshold  
  
You can decrease or increase the size depending on how much memory you have on each executor and in the driver.  
  
If you are confident that you have enough memory you can use a broadcast join with DataFrames larger than 10 MB (even up to 100 MB).



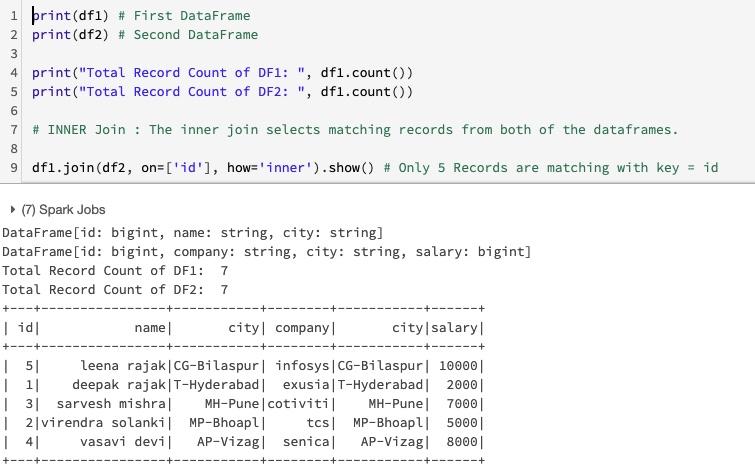
**#Question108**:  
Shuffle Hash Join in Spark ?  
  
A Shuffle Hash Join is the most basic type of join. Data from each side of the join is shuffled into partitions by key, but no sort if required. This join strategy can be used with large tables, but you must monitor for data skew as it may lead to out-of-memory errors.  
  
There are two phases in it:  
  
1. Shuffle Phase : In this phase spark shuffles data across the partitions. The idea here is that if two datasets have the same keys, they end up in the same partition so that the data required for join is available in the same partition.  
  
2. Hash Join: A standard hash join is performed on each executor.  
  
Here is the thing to note that is Shuffle Hash join will be used as the join strategy only when **spark.sql.join.preferSortMergeJoin** is set to false.  
  
and the cost to build a hash map is less than sorting the data. By default, sort merge join is preffered over Shuffle Hash Join.



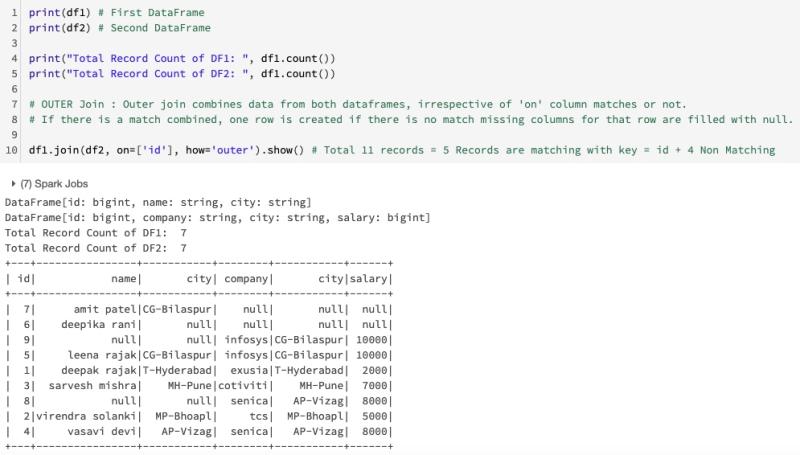
**#Question109**:  
Shuffle Sort Merge Join in Spark ?  
  
A Sort Merge Join is the most robust join strategy. It can be used with data of any size. It requires that data be shuffled and sorted, which is a costly operation because it requires data to move across executors. If your tables are small, performance may be slower than with other implementations.  
  
There are two phases in it:  
  
1. Sort Phase : The sort phase sorts each data set by its desired join key.  
  
2. Merge Phase : the merge phase iterates over each key in the row from each data set and merges the rows if the two keys match.  
  
By default, the SortMergeJoin is enabled via **spark.sql.join.preferSortMerge** Join.



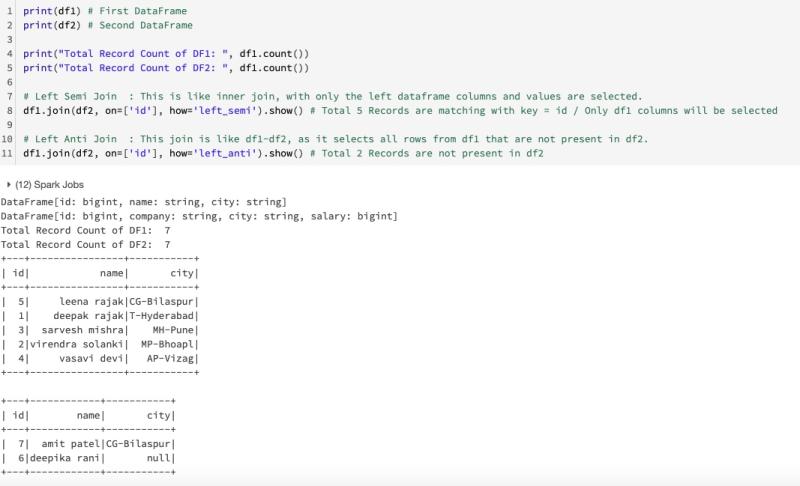
**#Question110**:  
INNER Join in Spark ?  
  
INNER Join : The inner join selects matching records from both of the dataframes.  
  
on= [key columns]  
how= [join type]



**#Question111**:  
OUTER Join in Spark ?  
  
OUTER Join : Outer join combines data from both dataframes, irrespective of 'on' column matches or not.  
  
If there is a match combined, one row is created if there is no match missing columns for that row are filled with null.  
  
on= [key columns]  
how= [join type]



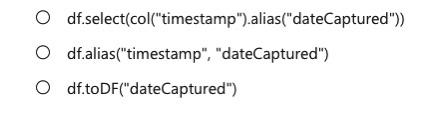
**#Question112**:  
LEFT SEMI & LEFT ANTI Join in Spark ?  
  
Left Semi Join : This is like inner join, with only the left dataframe columns and values are selected.  
  
Left Anti Join : This join is like df1-df2, as it selects all rows from df1 that are not present in df2.  
  
on= [key columns]  
how= [join type]



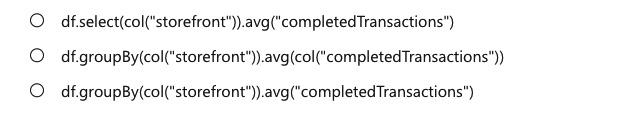
**#Question113**:  
REPARTITON & COALESCE in Spark ?  
  
REPARTITON  
The repartition method can be used to either increase or decrease the number of partitions in a DataFrame.  
  
COALESCE  
The coalesce method reduces the number of partitions in a DataFrame.



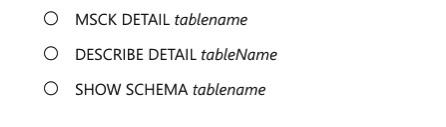
**#Question114**:  
Which method for renaming a DataFrame's column is incorrect?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



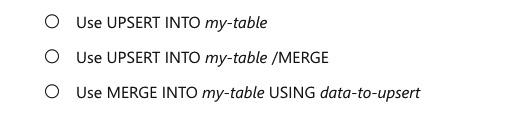
**#Question115**:  
You need to find the average of sales transactions by storefront. Which of the following aggregates would you use?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



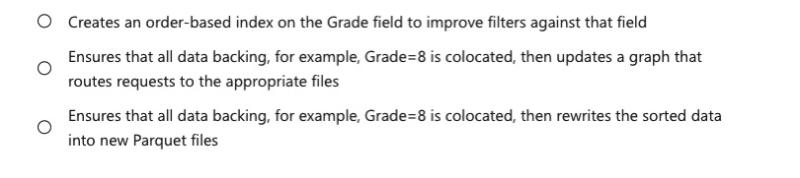
**#Question116**:  
What is the Databricks Delta command to display metadata?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



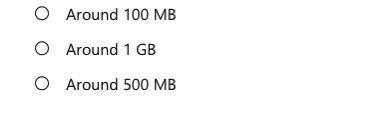
**#Question117**:  
How do you perform UPSERT in a Delta dataset?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



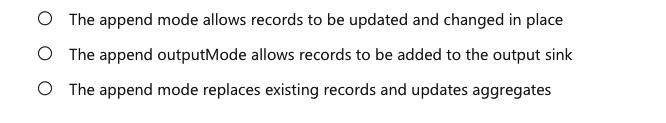
**#Question118**:  
What optimization does the following command perform: OPTIMIZE Students ZORDER BY Grade?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



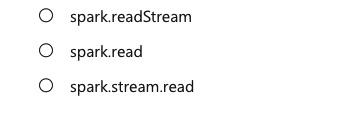
**#Question119**:  
What size does OPTIMIZE compact small files to?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



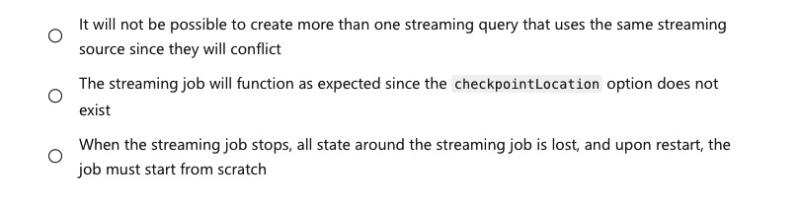
**#Question120**:  
When doing a write stream command, what does the outputMode("append") option do?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



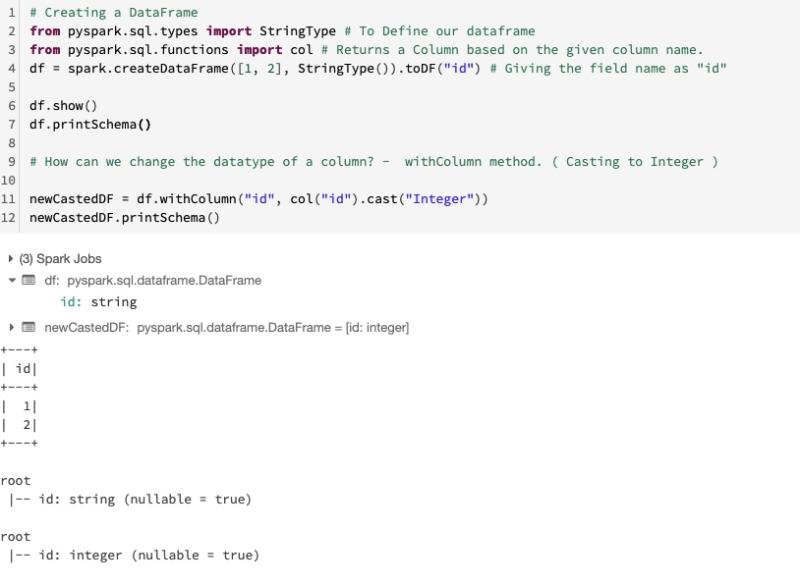
**#Question121**:  
In Spark Structured Streaming, what method should be used to read streaming data into a DataFrame?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**#Question122**:  
What happens if the command option("checkpointLocation", pointer-to-checkpoint directory) is not specified?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**#Question123**:  
How can we change the datatype of a column in Spark ?  
  
  
1. Using withColumn method  
2. cast() method of col() function  
2. Casting it to Integer or vice-versa



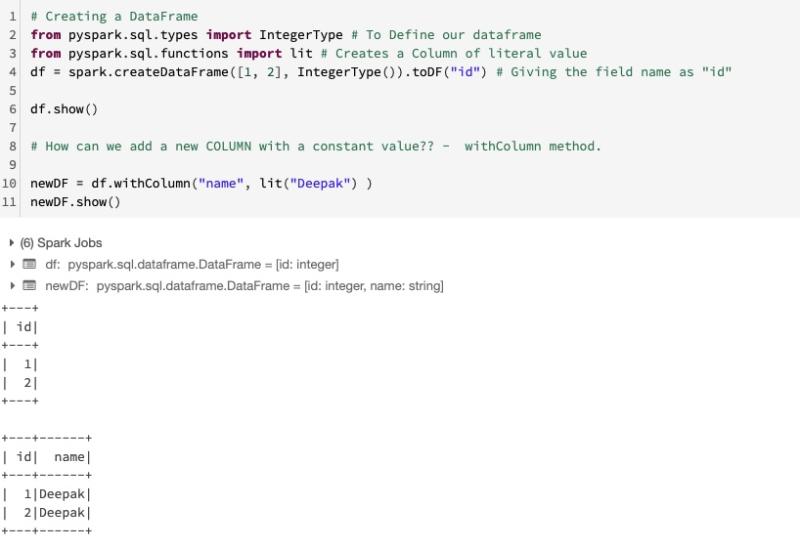
**#Question124**:  
How can we change the value of the existing column ?  
  
  
1. Using withColumn method.  
2. We can use any supported operation  
3. Here, We have used Multiplication



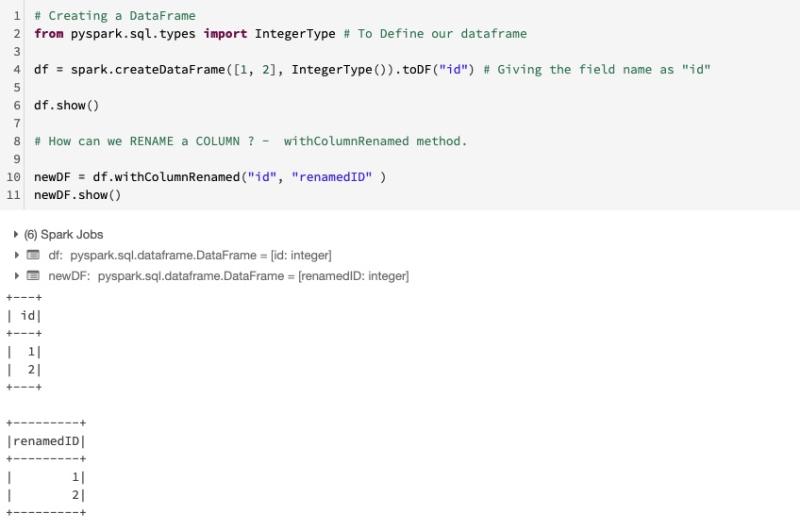
**#Question125**:  
How can we derived new COLUMN from an existing?  
  
  
1. Using withColumn method.  
2. First parameter - New Column Name  
3. Second parameter - Existing Column ( with or without any operations )  
4. We can use any supported operation  
5. Here, We have used Multiplication



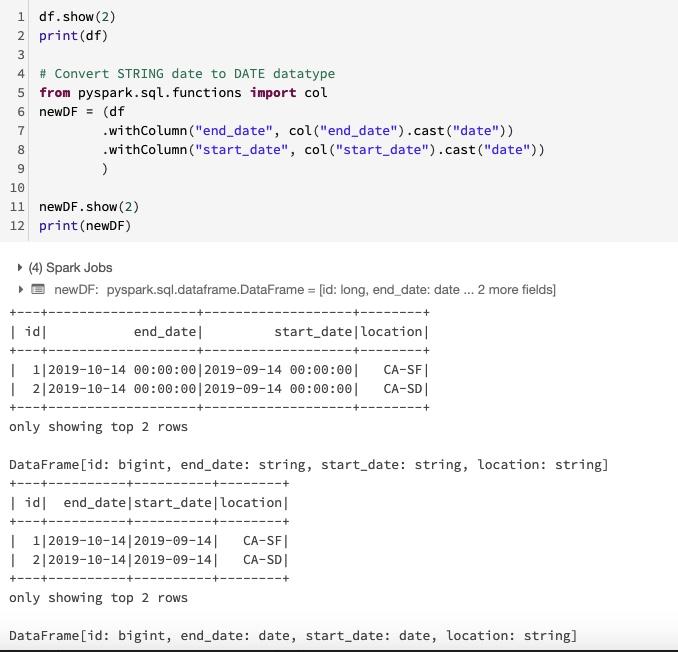
**#Question126**:  
How can we add a new COLUMN with a constant value?  
  
  
1. Using withColumn method.  
2. First parameter - New Column Name  
3. Using the lit() function



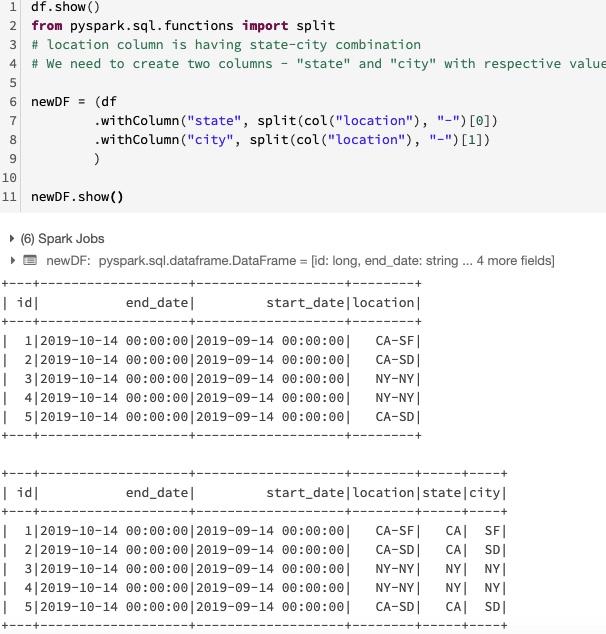
**#Question127**:  
How can we RENAME a COLUMN ?  
  
  
1. Using withColumnRenamed method.  
2. First parameter - Existing Column Name  
3. Second parameter - New Column Name



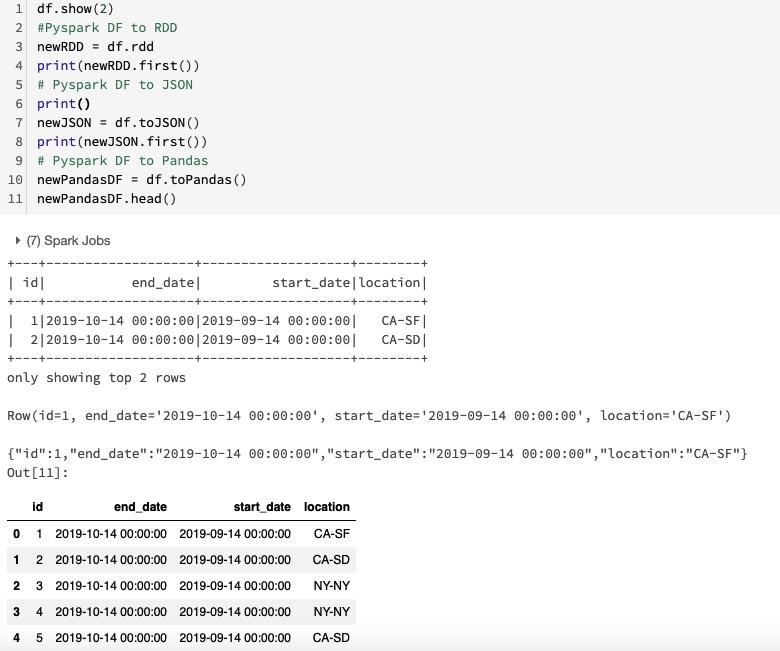
**#Question128**:  
Convert STRING date to DATE datatype?  
  
  
1. Using withColumn method.  
2. Using col() method with cast() function  
3. Alternatively we can use the to\_date() function as well



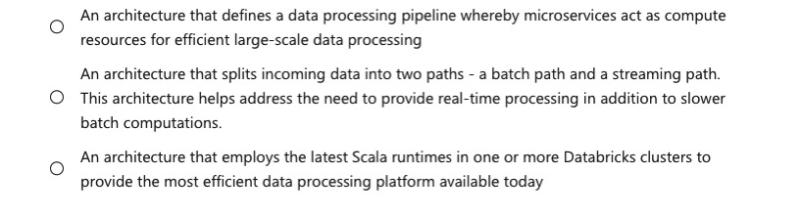
**#Question129**:  
How to SPLIT values of the COLUMN and create TWO separate columns from those values ?  
  
  
1. Using withColumn method.  
2. Using split() method  
3. get the first value via [0]  
4. get the second value via [1]



**#Question130**:  
How to CONVERT PySpark Dataframe into RDD, JSON and PANDAS dataframe ?  
  
  
1. Using .rdd method.  
2. Using .toJSON() method  
3. Using .toPandas() method



**#Question131**:  
What is a lambda architecture and what does it try to solve?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



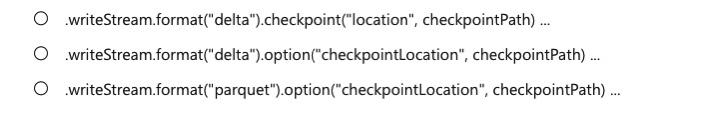
**Answer: 2**

**#Question132**:  
What command should be issued to view the list of active streams?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.



**Answer: 2**

**#Question133**:  
What is required to specify the location of a checkpoint directory when defining a Delta Lake streaming query?  
  
Let's have some Knowledge check. Please answer in the comment. I will post the correct answer after 24 Hours.

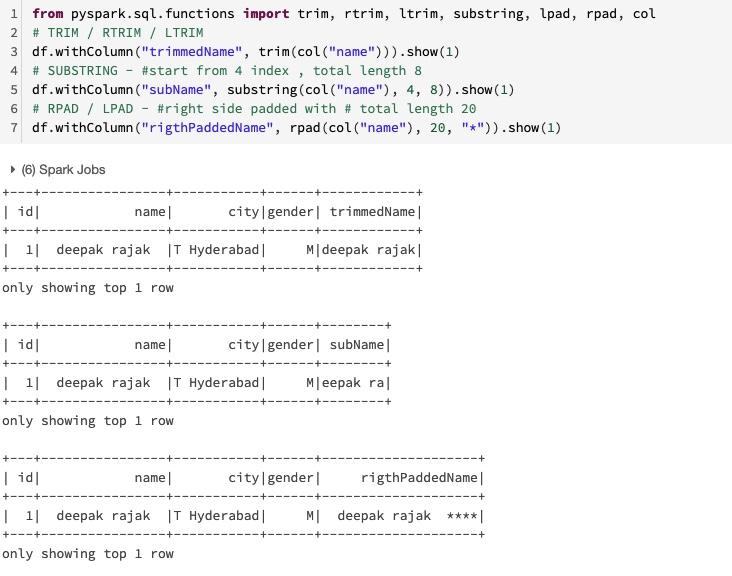


**Answer: 2**

**#Question134**:  
UPPER, LOWER, LENGTH & SPLIT function in Pyspark ?  
  
1. UPPER - To convert the value in Uppercase  
2. LOWER - To convert the value in Lowercase  
3. LENGTH - To get the length of the Column  
4. SPLIT - To split the values of the Column based on the separator ( In the below example - Blank Space is the separator )



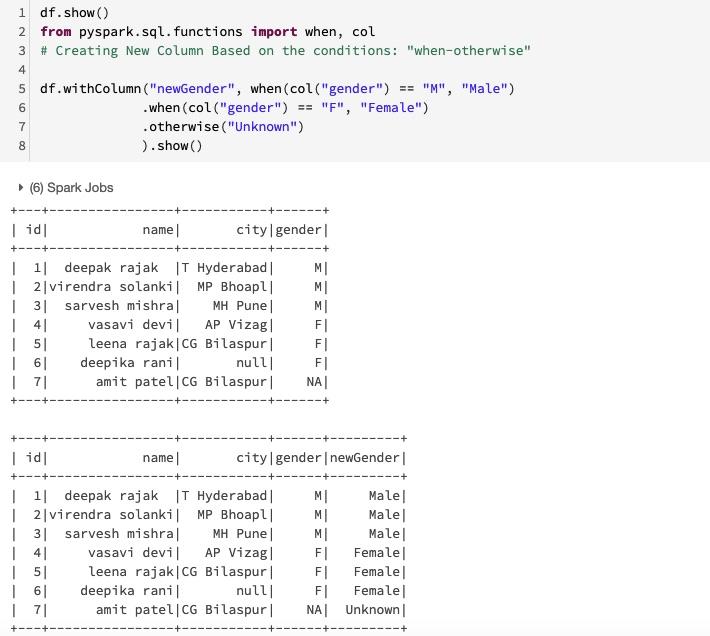
**#Question135**:  
TRIM, RTRIM, LTRIM, SUBSTRING, RPAD, LPAD function in Pyspark ?  
  
1. TRIM - leading / trailing white-space trimmed  
2. RTRIM - right side white-space trimmed  
3. LTRIM - left side white-space trimmed  
4. SUBSTRING - To get the substring from a string. start Index and total length of the resulting substring need to be passed as parameters.  
5. RPAD - For right side padding  
6. LPAD - For left side padding



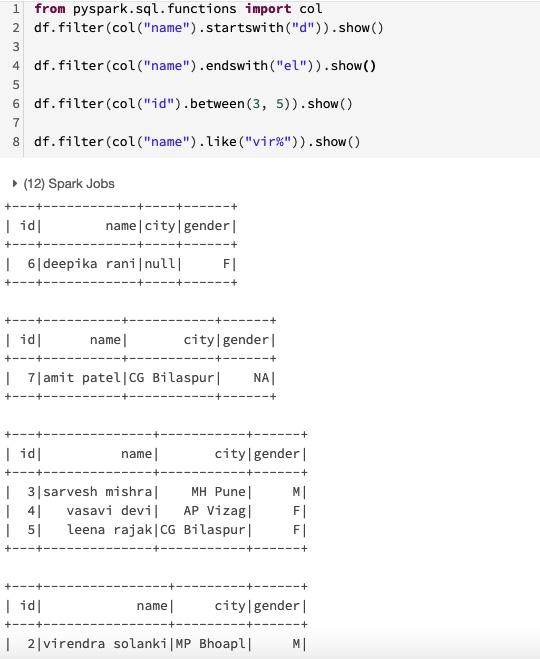
**#Question136**:  
How can we check if a field exists in a DataFrame ?  
  
1. By using **df.schema** method  
2. **df.schema.fields** prints Full schema  
3. **df.schema.names** prints only column names  
4. We can use a simple if - in to check if the field is present in our dataframe or not ?



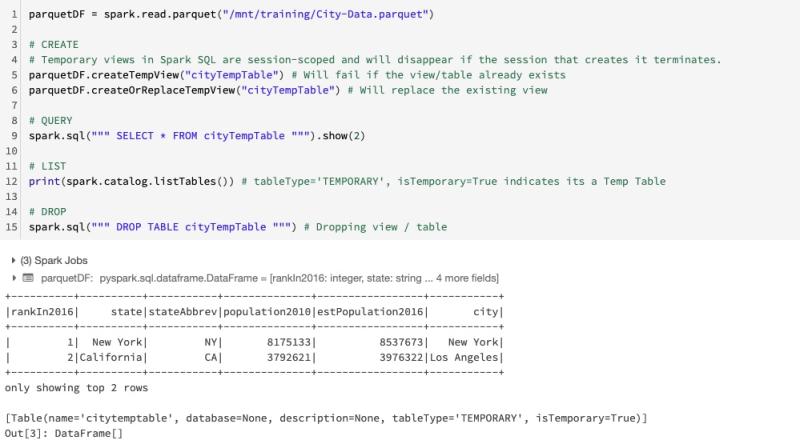
**#Question137**:  
How to create a NEW COLUMN based on certain conditions using WHEN - OTHERWISE construct.  
  
1. Using withColumn to create the new column  
2. Use WHEN to create the condition  
3. Use OTHERWISE to create the default value which doesn't match any condition.



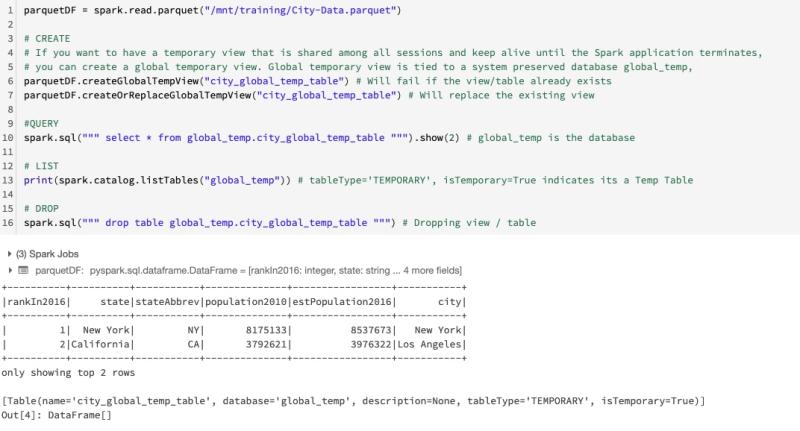
#Question138:  
COL method - STARTSWITH, ENDSWITH , BETWEEN & LIKE function  
  
1. STARTSWITH - To get the values which startswith some string.  
2. ENDSWITH - To get the values which endswith some string.  
3. BETWEEN - To get the values between 2 ranges.  
4. LIKE - To get the values which is like the given string.



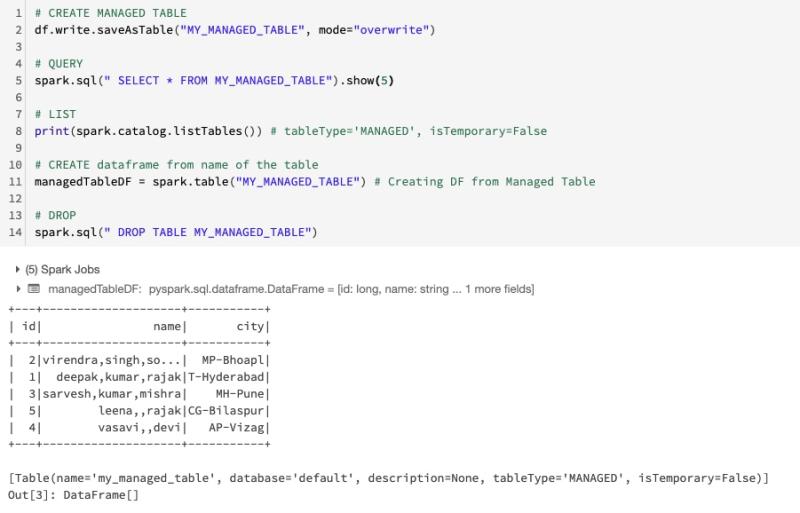
#Question139:  
What is Local Temporary View ? How we can CREATE, QUERY, LIST and DROP them ?  
  
Local Temporary views in Spark SQL are session-scoped and will disappear if the session that creates it terminates.  
  
1. CREATE - We can create the local temp view from either - "createTempView" or "createOrReplaceTempView" method  
  
2. QUERY - Same way, the way we query a table or view via spark.sql method  
  
3. LIST - Using spark.catalog.listTables() method.  
  
4. DROP - Same way, the way we drop a table or view via spark.sql method



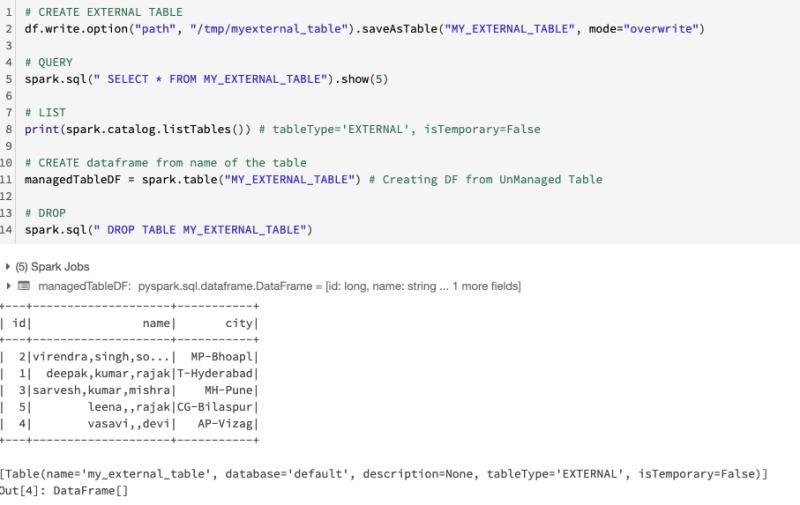
#Question140:  
What is Global Temporary View ? How we can CREATE, QUERY, LIST and DROP them ?  
  
If you want to have a temporary view that is shared among all sessions and keep alive until the Spark application terminates,  
you can create a global temporary view. Global temporary view is tied to a system preserved database global\_temp.  
  
1. CREATE - We can create the local temp view from either - "createGlobalTempView" or "createOrReplaceGlobalTempView" method  
  
2. QUERY - Same way, the way we query a table or view via spark.sql method  
  
3. LIST - Using spark.catalog.listTables("global\_temp") method.  
  
4. DROP - Same way, the way we drop a table or view via spark.sql method



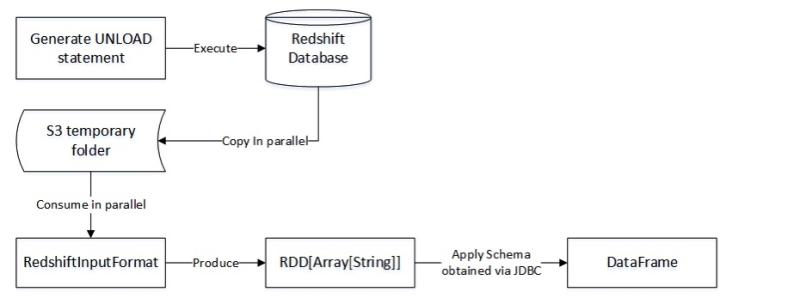
#Question141:  
How can we save the Spark DataFrame as MANAGED tables? How can we create the Dataframe from Table?  
  
saveAsTable will materialize the contents of the DataFrame and create a pointer to the data in the Hive metastore.  
  
Managed tables will also have their data deleted automatically when a table is dropped.  
  
1. CREATE - We can create the managed table from "saveAsTable' method  
  
2. QUERY - Same way, the way we query a table or view via spark.sql method  
  
3. LIST - Using spark.catalog.listTables() method.  
  
4. DROP - Same way, the way we drop a table or view via spark.sql method.  
  
5. CREATE DATAFRAME: we can use the method .table() for creating Dataframe



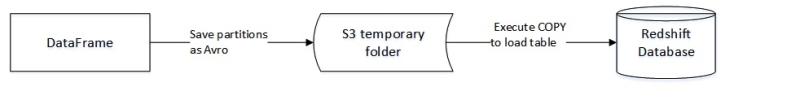
#Question142:  
How can we save the Spark DataFrame as UNMANAGED tables (External Table) ? How can we create the Dataframe from Table?  
  
saveAsTable will materialize the contents of the DataFrame and create a pointer to the data in the Hive metastore.  
  
UnManaged tables will not have their data deleted automatically when a table is dropped. We need to delete the data files separetly if we wish to delete them.  
  
1. CREATE - We can create the managed table from "saveAsTable' method. We need to supply the path as parameter.  
  
2. QUERY - Same way, the way we query a table or view via spark.sql method  
  
3. LIST - Using spark.catalog.listTables() method.  
  
4. DROP - Same way, the way we drop a table or view via spark.sql method.  
  
5. CREATE DATAFRAME: we can use the method .table() for creating Dataframe



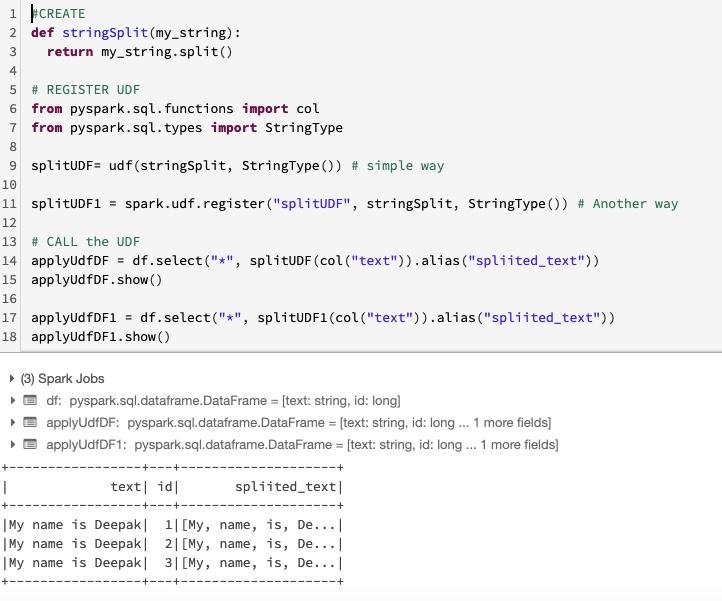
#Question143:  
How Spark reads from AWS Redshift ?  
  
The spark-redshift connector executes a Redshift UNLOAD command (using JDBC) which copies the Redshift table in parallel to a temporary S3 bucket provided by the user.  
  
Next it reads these S3 files in parallel using the Hadoop InputFormat API and maps it to an RDD instance.  
  
Finally, it applies the schema of the table (or query), retrieved using JDBC metadata retrieval capabilities, to the RDD generated in the prior step to create a DataFrame instance.  
  
Note: Databricks preloads the necessary libraries to use spark-redshift so you don't have to import and attach them yourself.  
  
1. Redshift Connector JAR  
2. AWS Redshift SDK



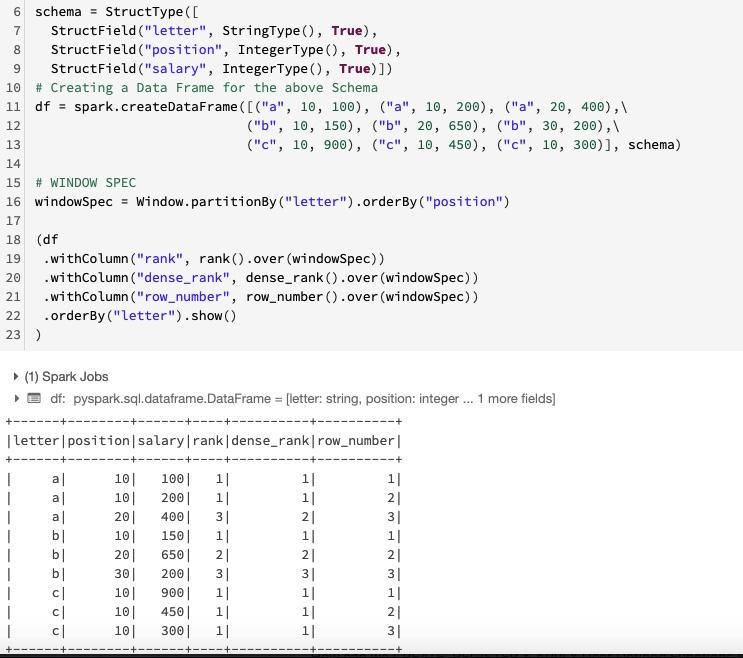
#Question144:  
How Spark writes to AWS Redshift ?  
  
The spark-redshift will first create the table in Redshift using JDBC.  
  
It then copies the partitioned RDD encapsulated by the source DataFrame instance to the temporary S3 folder.  
  
Finally, it executes the Redshift COPY command that performs a high performance distributed copy of S3 folder contents to the newly created Redshift table.  
  
Note: Databricks preloads the necessary libraries to use spark-redshift so you don't have to import and attach them yourself.  
  
1. Redshift Connector JAR  
2. AWS Redshift SDK



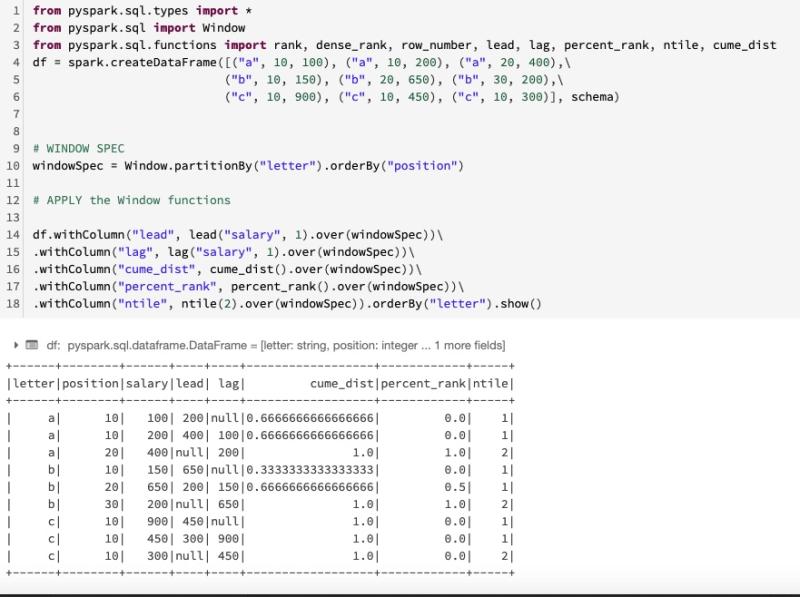
#Question145:  
How can we CREATE, REGISTER & APPLY User Defined Functions in Spark?  
  
User Defined functions can be created and used very easily in Spark.  
  
1. CREATE - We can create the UDF in any native language which is supported by Spark i.e. Scala, Python, Java etc  
  
2. Register - We can register the UDF by calling the method udf()  
  
3. APPLY - We can use the UDF wherever we need any transformation like the way we use the built-in-function provided by Spark



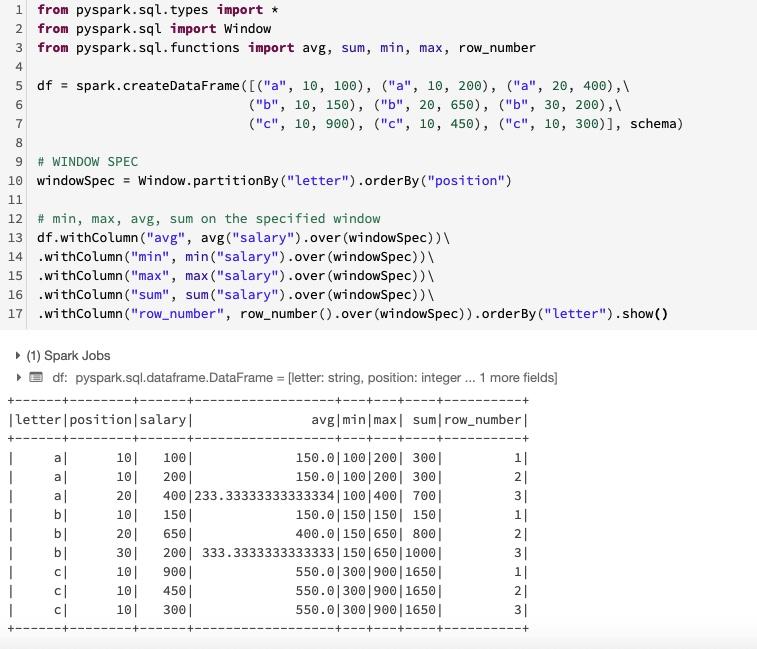
#Question146:  
WINDOW Functions in Spark - RANK(), DENSE\_RANK(), ROW\_NUMBER()  
  
Spark Window Functions have the following traits:  
  
1. Perform a calculation over a group of rows, called the Frame.  
2. A frame corresponding to the current row  
3. Return a new value to for each row by an aggregate/window function  
4. Can use SQL grammar or DataFrame API.  
  
  
1. RANK - Rank of rows. leaves gaps in rank when there are ties.  
  
2. DENSE\_RANK - dense\_rank() window function is used to get the result with rank of rows within a window partition without any gaps. This is similar to rank() function difference being rank function leaves gaps in rank when there are ties.  
  
3. ROW\_NUMBER - Sequential number for each row



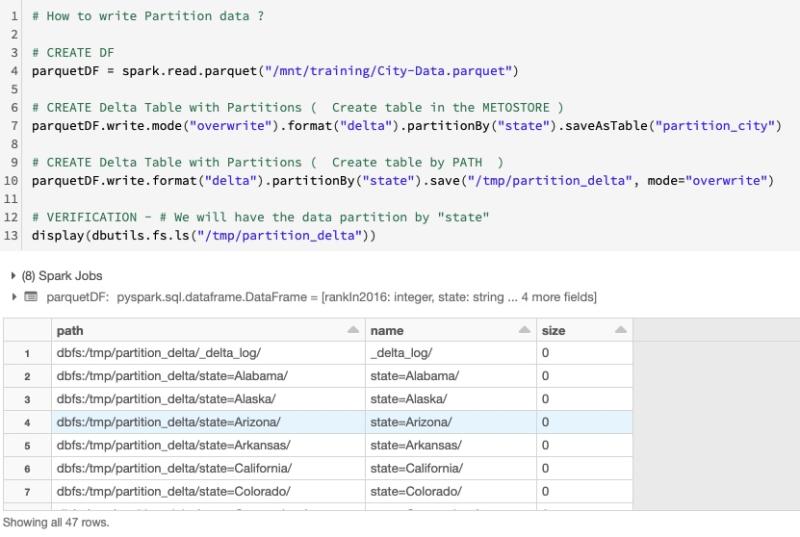
#Question147:  
WINDOW Functions in Spark - LEAD(), LAG(), CUME\_DIST(), PERCENT\_RANK() and NTILE()  
  
Spark Window Functions have the following traits:  
  
1. Perform a calculation over a group of rows, called the Frame.  
2. A frame corresponding to the current row  
3. Return a new value to for each row by an aggregate/window function  
4. Can use SQL grammar or DataFrame API.  
  
  
1. LEAD - lead means getting the value from the next row.  
  
2. LAG - lag means getting the value from the previous row.  
  
3. CUME\_DIST - cume\_dist() window function is used to get the cumulative distribution of values within a window partition.  
  
4. PERCENT\_RANK - Gives percent rank  
  
5. NTILE - ntile() window function returns the relative rank of result rows within a window partition. In below example we have used 2 as an argument to ntile hence it returns ranking between 2 values (1 and 2)



#Question148:  
WINDOW Functions in Spark - AVG(), MIN(), MAX(), SUM() and ROW\_NUMBER()  
  
Spark Window Functions have the following traits:  
  
1. Perform a calculation over a group of rows, called the Frame.  
2. A frame corresponding to the current row  
3. Return a new value to for each row by an aggregate/window function  
4. Can use SQL grammar or DataFrame API.  
  
  
1. AVG - Average within the window  
  
2. MIN - Min within the window  
  
3. MAX - Max within the window  
  
4. SUM - Sum within the window  
  
5. ROW\_NUMBER - row number for each row within the window



#Question149:  
How to WRITE PARTITIONED data in Spark?  
  
By using the - partitionBy() method.  
  
We can write the data in couple of ways:  
  
1. By Creating a Managed Table  
2. By Creating an External Table



#Question150:  
How to get HISTORY of operations in Delta Table in Spark ?  
  
By using the - describe history command.

