

Above are the two approaches to define schema

Dataframe does not have datatypes, they have spark types

We also learned that every Spark application does three things.

1. Load
2. Process
3. Write

However, contrary to a large number of sources and destinations, we have just three options to hold the data in Apache Spark.

1. Spark Dataframe
2. Spark DataSet
3. Spark RDD

Spark Dataframes

A Spark data Frame is a distributed collection of structured data. Since they hold structured data, you can think of them as a database table with a schema attached to it

## Spark Session

A Spark session is the entry point to programming Spark with Data Frame APIs

he most critical Spark Session API is the read method. It returns a Data Frame Reader

## Spark DataFrameReader

Following other options

1. quote -> Quote is the character used to enclose the string values. Quoting your string value is critical if you have a field that contains a comma. The default value is the double quote character, and hence we can rely on the default value for our example.

2. inferSchema -> Infer schema will automatically guess the data types for each field. If we set this option to TRUE, the API will read some sample records from the file to infer the schema. If we want to set this value to false, we must specify a schema explicitly.

3. nullValue -> The null value is used to define the string that represents a null.

4. timestampFormat -> This one is to declare the timestamp format used in your CSV file.

5. mode -> This one is crucial. It defines the method for dealing with a corrupt record. There are three supported modes.  
PERMISSIVE, DROPMALFORMED, and FAILFAST.  
The first two options allow you to continue loading even if some rows are corrupt. The last one throws an exception when it meets a corrupted record. We will be using the last one in our example because we do not want to proceed in case of data errors.

|  |
| --- |
| Val df = spark.read |
|  | .format("csv") |
|  | .option("header", "true") |
|  | .option("inferSchema", "true") |
|  | .option("nullValue", "NA") |
|  | .option("timestampFormat", "yyyy-MM-dd'T'HH:mm?:ss") |
|  | .option("mode", "failfast") |
|  | .option("path", "/home/prashant/spark-data/survey.csv") |
|  | .load() |

ou have already seen some transformation code earlier.We used the read API to load the data from a CSV file.

|  |  |
| --- | --- |
|  | val df = spark.read |
|  | .format("csv") |
|  | .option("header", "true") |
|  | .option("inferSchema", "true") |
|  | .option("nullValue", "NA") |
|  | .option("timestampFormat", "yyyy-MM-dd'T'HH:mm:ss") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/survey.csv") |
|  |  |
|  | //Then we applied a select transformation and a filter condition. |
|  | val sel = |
|  | df.select("Timestamp", "Age", "remote\_work", "leave").filter("Age > 30") |

If you are a database developer, you will see the above transformation as an SQL expression.

|  |  |
| --- | --- |
|  | select timestamp, age,remote\_work,leave |
|  | from survey\_tbl |
|  | where age > 30; |

**To make this SQL work, all you need is a table and an SQL execution engine.The good news is that the Spark offers you both of these things.**

In the previous document we wrote some scala code now that can converted to SQL query

|  |
| --- |
| select gender, sum(all\_yes), sum(all\_nos) |
|  | from (select case when lower(trim(gender)) in ('male','m','male-ish','maile','mal', |
|  | 'male (cis)','make','male','man','msle', |
|  | 'mail', 'malr','cis man', 'cis male') |
|  | then 'Male' |
|  | when lower(trim(gender)) in ('cis female','f','female','woman', |
|  | 'femake','female ','cis-female/femme', |
|  | 'female (cis)','femail') |
|  | then 'Female' |
|  | else 'Transgender' |
|  | end as gender, |
|  | case when treatment == 'Yes' then 1 else 0 end as all\_yes, |
|  | case when treatment == 'No' then 1 else 0 end as all\_nos |
|  | from survey\_tbl) |
|  | where gender != 'Transgender' |
|  | group by gender |

## Spark SQL requires Schema

A schema is nothing more than a definition for the column names and their data types. In our earlier example, we allowed the API to infer the schema. However, there are two approaches to handle schema.

1. Let the data source define the schema, and we infer it from the source.
2. Define a schema explicitly in your program and read the data using your schema definition.

## When your source system offers a well-defined schema, schema inference is a reasonable choice. However, it is a good idea to define your schema manually while working with untyped sources such as CSV and JSON. In our current example, we are loading data from a CSV file. So, the recommendation is to define the schema instead of using the inferSchema.

## Spark Types to define Schema

Dataframe does not have datatypes, they have spark types

## Data frames do not use Scala types or Python types. No matter which language are you using for your code, A Spark data frame API always uses Spark types.You can get the list of Spark Types in [org.apache.spark.sql.types](https://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.sql.types.package" \t "_blank) package.

## How to define a Spark Schema

|  |
| --- |
| //You can create a Schema for survey data set using below code |
|  | import org.apache.spark.sql.types.\_ |
|  | val surveySchema = StructType( |
|  | Array( |
|  | StructField("timestamp", TimestampType, true), |
|  | StructField("age", LongType, true), |
|  | StructField("gender", StringType, true), |
|  | StructField("country", StringType, true), |
|  | StructField("state", StringType, true), |
|  | StructField("self\_employed", StringType, true), |
|  | StructField("family\_history", StringType, true), |
|  | StructField("treatment", StringType, true), |
|  | StructField("work\_interfere", StringType, true), |
|  | StructField("no\_employees", StringType, true), |
|  | StructField("remote\_work", StringType, true), |
|  | StructField("tech\_company", StringType, true), |
|  | StructField("benefits", StringType, true), |
|  | StructField("care\_options", StringType, true), |
|  | StructField("wellness\_program", StringType, true), |
|  | StructField("seek\_help", StringType, true), |
|  | StructField("anonymity", StringType, true), |
|  | StructField("leave", StringType, true), |
|  | StructField("mental\_health\_consequence", StringType, true), |
|  | StructField("phys\_health\_consequence", StringType, true), |
|  | StructField("coworkers", StringType, true), |
|  | StructField("supervisor", StringType, true), |
|  | StructField("mental\_health\_interview", StringType, true), |
|  | StructField("phys\_health\_interview", StringType, true), |
|  | StructField("mental\_vs\_physical", StringType, true), |
|  | StructField("obs\_consequence", StringType, true), |
|  | StructField("comments", StringType, true) |
|  | ) |
|  | ) |
|  |  |
|  | //You can load the data using above schema |
|  | val df = spark.read |
|  | .format("csv") |
|  | .schema(surveySchema) |
|  | .option("header", "true") |
|  | .option("nullValue", "NA") |
|  | .option("timestampFormat", "yyyy-MM-dd'T'HH:mm:ss") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/survey.csv") |

Spark data frame schema is a *StructType* that contains a set of *StructFields*. Each *StructField* defines a column. The *[StructField](https://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.sql.types.StructField" \t "_blank)*is a serializable class under Scala *AnyRef*.  
The S *StructField* constructor can take four values.

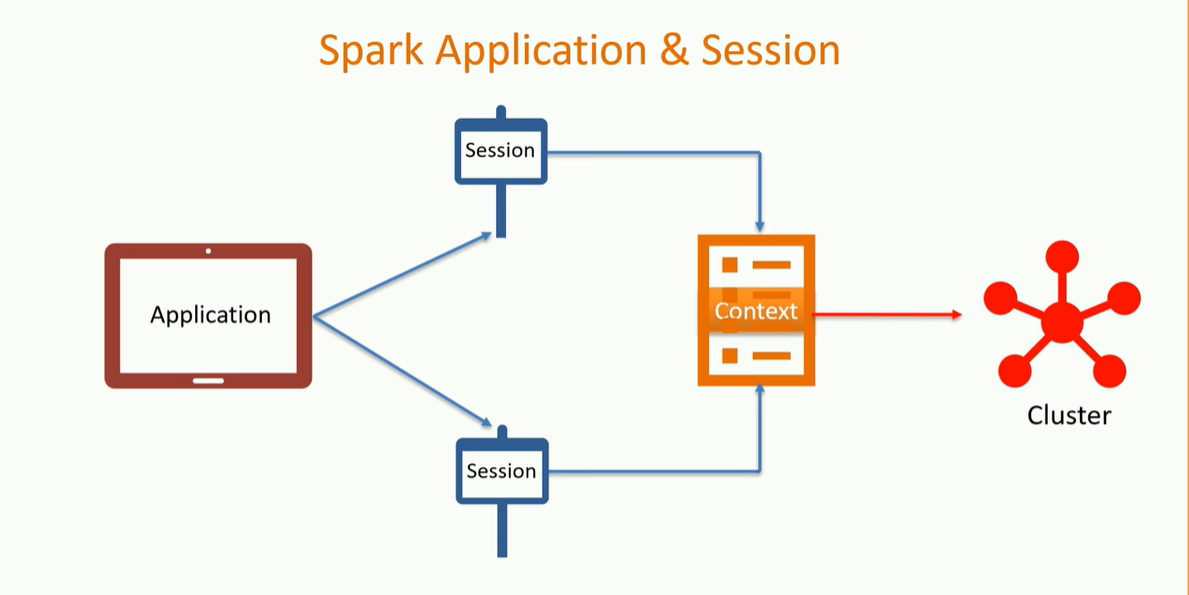
1. The name of the column
2. The data type of the column.
3. A boolean that tells if the field is nullable. This parameter defaults to true.
4. You can also supply some metadata for each column. The metadata is nothing but a map of key-value pairs. The default value is empty.

The *[StructType](https://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.sql.types.StructType" \t "_blank)*is also a class that holds an array of *StructFields*. If you are using Python, both of those structs are same in Python as well. However, the Python *StructType* is a list of *StructFields* whereas Scala *StrcutType* is an array of *StructField*.

Spark Temporary View

Apache Spark allows you to create a temporary view using a data frame. It is just like a view in a database. Once you have a view, you can execute SQL on that view. Spark offers four data frame methods to create a view.

1. createGlobalTempView
2. createOrReplaceGlobalTempView
3. createOrReplaceTempView
4. createTempView



## Global vs Local Temp view

The local temporary view is only visible to the current spark session. However, a Global temporary view is visible to the current spark application across the sessions.  
Wait a minute. Do you mean a SparkSession and a Spark Application are two different things?  
Yes. We normally start a Spark Application by creating a Spark session. To a beginner, it appears that a Spark Application can have a single session. However, that is not true. You can have multiple sessions in a single Spark application. The Spark session internally creates a Spark context. A SparkContext represents the connection to a Spark cluster. It also keeps track of all the RDDs, cached data as well as the configurations.  
You cannot have more than one Spark Context in a single JVM. That means, one instance of an application can have only one connection to the cluster and hence a single Spark context. You cannot have more than one Spark context. However, your application can create multiple Spark Sessions. All those sessions will point to the same context, but you can have multiple sessions.

local temporary views, they are only visible to the current session. However, global temporary views are visible across the spark sessions within the same application.  
In all this discussion, one thing is crystal clear. None of them are visible to other applications. So, you create a global temporary view or a local temporary view, they are always local to your application, and they live only till your application is alive

let me create a local temporary view.

|  |  |
| --- | --- |
|  | df.createOrReplaceTempView("survey\_tbl") |
|  |  |

The method takes the name of the view as an argument. The above statement must have created a temporary table or a view. Where can you find it?  
Well, a temporary view is maintained by the Spark session. So, let's check the Spark session.

|  |  |
| --- | --- |
|  | spark.catalog.listTables.show |

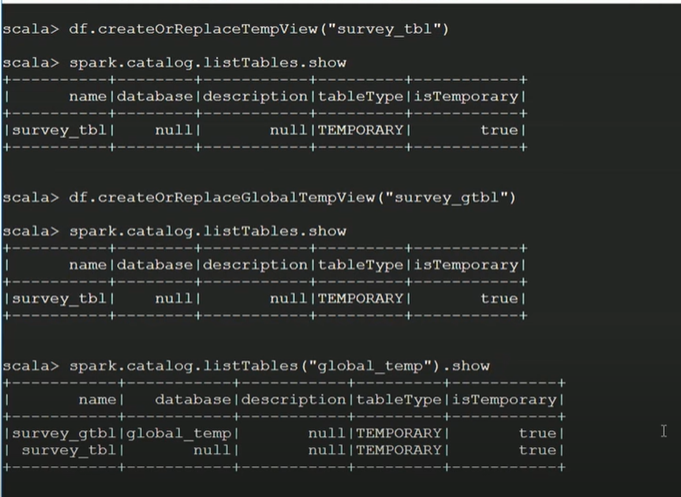
Spark session offers you a catalog. A catalog is an interface that allows you to create, drop, alter or query underlying databases, tables, and functions

The above statement is using the *listTables* method in the catalog. If you check the output of the above statement, you can see that the view that we created is a temporary table that doesn't belong to any databases.  
Let's create a global temporary table and see if we can list that as well.

|  |  |
| --- | --- |
|  | df.createOrReplaceGlobalTempView("survey\_gtbl") |

We used the appropriate method to create a Global temporary view on our data frame. I named the view as *survey\_gtbl*. If you call the catalog *listTables* method once again. You won’t see that global table. There is a reason for that. A Global temp table belongs to a system database called *global\_temp*. So, if you want to access the global temp table, you must look into the *global\_temp* database. So, the correct method call should also specify the database name.

|  |  |
| --- | --- |
|  | spark.catalog.listTables("global\_temp").show |



The output of the above statement should list your global temp table. Once you register the temp table, executing your SQL statement is a simple thing.

|  |  |
| --- | --- |
|  | spark.sql("""select timestamp, age,remote\_work,leave |
|  | from survey\_tbl |
|  | where age > 30""") |

xecute an SQL on the spark session, and you get a data frame in return. So, if you think that the SQL is simpler to solve your problems, instead of using lengthy data frame API chains, you are free to use SQL. And surprisingly, you do not have a performance penalty. The SQL works as fast as a Data frame transformation.

|  |  |
| --- | --- |
|  | spark.sql("""select gender, sum(yes), sum(no) |
|  | from (select case when lower(trim(gender)) in ('male','m','male-ish','maile','mal', |
|  | 'male (cis)','make','male ','man','msle', |
|  | 'mail','malr','cis man','cis male') |
|  | then 'Male' |
|  | when lower(trim(gender)) in ('cis female','f','female','woman', |
|  | 'female','female ','cis-female/femme', |
|  | 'female (cis)','femail') |
|  | then 'Female' |
|  | else 'Transgender' |
|  | end as gender, |
|  | case when treatment == 'Yes' then 1 else 0 end as yes, |
|  | case when treatment == 'No' then 1 else 0 end as no |
|  | from survey\_tbl) |
|  | where gender != 'Transgender' |
|  | group by gender""").show |