Q1. What is Apache Spark SQL?

Ans. Apache Spark SQL is a module for structured data processing in Spark. Spark SQL integrates relational processing (i.e. SQL) with Spark’s functional programming using Scala, Java, etc weave SQL queries with Dataframes/Datasets based transformations.

Q2. What are Datasets in Apache Spark SQL?

Ans. A Dataset is a distributed collection of data that was introduced in Spark 1.6 that provides the benefits of RDDs plus the benefits of Spark SQL's optimized execution engine.

Q3. How do you create Datasets in Apache Spark SQL?

Ans. A Dataset can be constructed from Java objects and then manipulated using functional transformations such as map(), flatMap(), filter() etc.

Q4. How do you convert existing RDDs to Spark Datasets?

Ans. Spark SQL supports two different methods to convert existing RDDs to Spark Datasets

1. Infer the schema using Reflection - Spark SQL can automatically convert an existing RDD of JavaBeans into a DataFrame by using reflection. The bean info which is obtained using reflection, defines the schema of the table.

2. Programatically specifying the schema - Spark SQL supports programatically specifying the schema, in cases where the Java bean cannot be defined ahead of time. This can be done in three steps.

1. Create an RDD of rows using the original RDD.
2. Create the schema represented by a StrutType matching the structure of rows in the RDD.
3. Apply the schema to the RDD of rows via *createDataFrame()* method provided by *SparkSession*.

Q6. What is SparkSession?

Ans. SparkSession is the entry point to a Spark application which contains information about the application and the application configuration parameters and values. A SparkSession object can be created by using the command *builder()* on a *SparkSession* object.

```
SparkSession spark = SparkSession.builder()

.appName('Spark SQL Example')

.config('config.option','value')

.getOrCreate()
```

Q7. What are DataFrames?

Ans. A DataFrame is a Dataset organized into named columns. A DataFrame is equivalent to a Relational Database Table. DataFrames can be created from a variety of sources such as structured data files, external databases, Hive tables and Resilient Distributed Datasets.

Q8. What is the difference between Temp View and Global Temp View?

Temporary views in Spark SQL are tied to the Spark session that created the view, and will not be available once the Spark session is terminated.

Global Temporary views are not tied to a Spark session, but can be shared across multiple Spark sessions. Global Temporary views are available until the Spark application is terminated. Global Temporary view is tied to a system database global_temp, and the view must be accessed using this qualified name.

```
df.createGlobalTempView('global_temp.test');

spark.sql('SELECT * FROM global_temp.test').show();

spark.newSession().sql('select * from global_temp.test').show();
```

Q9. How are aggregations performed on DataFrames?

Ans. DataFrames has built-in functions that provide common aggregation functions such as count(), countDistinct(), avg(), max(), min() etc.

Q10. How do you create Datasets from JSON files?

Ans. You can create a Dataset from JSON files by calling the method *read().json()* on a *SparkSession* object. Spark SQL automatically infers the schema of the JSON file and loads it as a Dataset.

Q11. What libraries do Spark SQL have?
Ans.

1. Data Source API

This library has built-in support for various Datasources shown above. This library can be used with various datasources for loading and storing **structured data**. It has built-in support for Hive, Avro, JSON, JDBC, Parquet, Elastic Search, MySQL, etc.

2. Dataframe API

A DataFrame is a distributed collection of data organised into structured named column. It is equivalent to a relational table in SQL used for storing data into tables.

3. SQLInterpreter And Optimiser

SQL Interpreter and Optimizer are functional programming constructed in Scala for supporting **cost based** and **rules based optimization** to make the queries run faster than RDDs.

4. SQL Service

SQL Service is an entry point for working with structured data in Spark. It enables you to create DataFrame objects as well as the execution of SQL queries.

Q12. How will you go about enabling Hive support in Spark 2.0?

Ans.

```
//Spark 2.0 builder pattern to create the Spark session
val hiveLocation = "location/spark-warehouse"
val spark = SparkSession
   .builder()
   .appName("SparkSessionZipsExample")
   .config("spark.sql.warehouse.dir", hiveLocation)
   .enableHiveSupport()
   .getOrCreate()

//Once the SparkSession is instantiated, you can configure Spark's runtime config properties.

//set new runtime options
spark.conf.set("spark.sql.shuffle.partitions", 6)
spark.conf.set("spark.executor.memory", "2g")
//get all settings
val configMap:Map[String, String] = spark.conf.getAll()

//access Hive catalog metadata
spark.catalog.listDatabases.show(false)
spark.catalog.listTables.show(false)
```

Q13. How will you go about using Spark SQL with Spark 2.0 SparkSession?

Ans.

```
// read the csv file and create the dataframe
val csvFile = args(0)
val employeeDF = spark.read.csv(csvFile)

// Now create an SQL table and issue SQL queries against it without
// using the sqlContext but through the SparkSession object.
// Creates a temporary view of the DataFrame

mydataDF.createOrReplaceTempView("my_table")
mydataDF.cache()
val resultsDF = spark.sql("SELECT name, state, age FROM my_table")
resultsDF.show(10)

```

You can also "printSchema" and perform transformations

```
// read the csv file and create the dataframe
val csvFile = args(0)
val mydataDF = spark.read.csv(csvFile)

df.printSchema()
df.select("city").show()

//increment age by 1
df.select($"name", $"age" + 1).show()
df.filter($"age" > 18).show()

```

Q14. How will you go about saving & reading from Hive table with SparkSession?

Ans.

```
 //drop if the table already exists
spark.sql("DROP TABLE IF EXISTS my_employee_table")

//saving to a hive table
spark.table("my_employee_table").write.saveAsTable("my_hive_employee_table")

//read from the hive table
val resultsHiveDF = spark.sql("SELECT name, state, age FROM my_hive_employee_table WHERE age > 18")
resultsHiveDF.show(10)

```

If you use the Scala implicits, you do not need to prefix with "spark" as in "spark.sql("……")"

```
import org.apache.spark.sql.Row
import org.apache.spark.sql.SparkSession
import spark.implicits._
import spark.sql
//.....
sql("DROP TABLE IF EXISTS my_employee_table")

```

Q15. How will you use a DataSet API with Spark SQL?
Ans.

```

case class Employee(name: String, state: String, age: Long)
val caseClassDataSet = Seq(Employee("John", "NSW", 25), Employee("Peter", "VIC", 17)).toDS()
caseClassDS.show()

val path = arg(0)
val employeeDS = spark.read.csv(path).as[Employee]
employeeDS.show()

```

Q16. How will you be reading json & parquet files?

Ans.

json

```scala
val path = arg(0) . //path to json file
val employeeDF = spark.read.json(path)
employeeDF.printSchema()
employeeDF.createOrReplaceTempView("tbl_employee")
val youngEmployees = spark.sql("SELECT * FROM tbl_employee WHERE age < 18")
youngsterNamesDF.show()
```

Parquet

```scala
val path = arg(0) . //path to parquet file
val employeeDF = spark.read.parquet(path)
employeeDF.printSchema()
employeeDF.createOrReplaceTempView("tbl_employee")
val youngEmployees = spark.sql("SELECT * FROM tbl_employee WHERE age < 18")
youngsterNamesDF.show()
```

Q17. What is a Spatk SQL’s UDF?
Ans. Spark SQL’s User-Defined Functions (UDFs) define new Column-based functions that **extend the vocabulary of Spark SQL’s DSL** for transforming Datasets.

```scala

val input = Seq((0, "John"),(1, "Peter")).toDF("id","name")

//uppercase function
val upper: String => String = _.toUpperCase

import org.apache.spark.sql.functions.udf
val upperUDF = udf(upper)
input.withColumn("upper", upperUDF('name')).show(false)

//you can register this function

spark.udf.register("customUpper", (input:String) => input.toUpperCase )
spark.catalog.listFunctions.filter('name like "%Upper%").show(false)

```