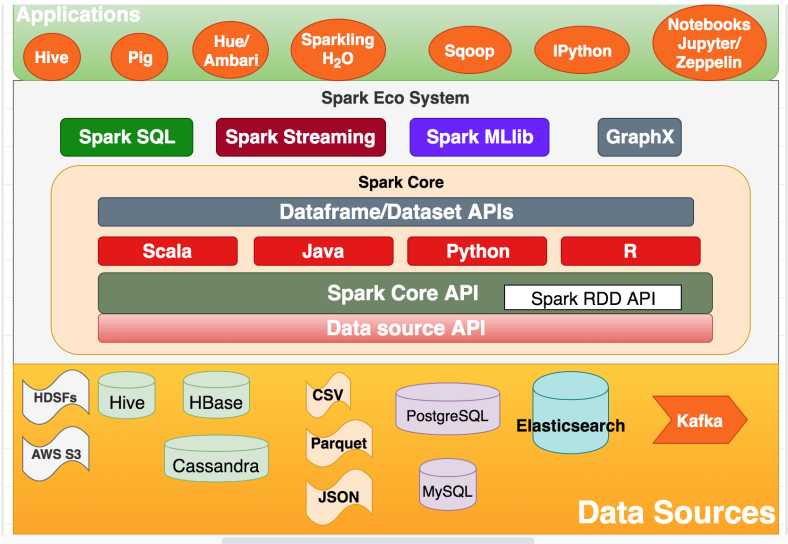
10 Spark SQL Interview Q&As

Posted on November 17, 2019

Q1. What is Spark SQL?
A1. 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. It provides support for various data sources as shown below:



Spark SQL in Spark Eco System

Q2. What libraries do Spark SQL have?
A2.

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.

300+ Java Interview FAQs

300+ Java FAQs 🔥
16+ Java Key Areas Q&As
150+ Java Architect FAQs
80+ Java Code Quality Q&As
150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥
FAQs Data - 01: SQL
FAQs Data - 02: Data Modelling
FAQs Data - 02: Data Warehouse
FAQs Data - 03: Big Data
FAQs Data - 04: Hadoop (HDFS)
FAQs Data - 05: MapReduce
FAQs Data - 06: Hive
FAQs Data - 07: Impala
FAQs Data - 08: Spark
FAQs Data - 09: Spark SQL
FAQs Data - 10: Apache Kafka
FAQs Data - 11: Data Governance
FAQs Data - 11: NoSQL
FAQs Data - 12: Data security
FAQs Data - 13: Analytics & Science
FAQs Data - 14: AWS
FAQs Data - 15: Sqoop & Nifi
FAQs Data - 16: Yarn, Zookeeper
FAQs Data - 40: Scala 140+ FAQs

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.

Q3. How will you go about enabling Hive support in Spark 2.0?
A3.

```
 //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 r

//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)

```

Q4. How will you go about using Spark SQL with Spark 2.0 SparkSession?
A4.

```
// 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)

```

FAQs Data - 41: 100+ Python FAQs
Tutorials - Big Data

800+ Java Interview Q&As

300+ Core Java Q&As
300+ Enterprise Java Q&As
150+ Java Frameworks Q&As
120+ Companion Tech Q&As
Tutorials - Enterprise Java

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()

```

Q5. How will you go about saving & reading from Hive table with SparkSession?

A5.

```
 //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

//read from the hive table
val resultsHiveDF = spark.sql("SELECT name, state, age FROM my_hive_
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")

```

Q6. How will you display the number of employees at different age groups?

A6.

```
df.groupBy("age").count().show()

```

Q7. How will you create a temporary view of a DataFrame?

A7.

```
df.createOrReplaceTempView("tmp_employee")
val sqlDF = spark.sql("SELECT * FROM tmp_employee")
sqlDF.show()

```

Q8. How will you use a DataSet API with Spark SQL?

A8.

```

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

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

```

Q9. How will you be reading json & parquet files?

A9.

json

```
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
youngsterNamesDF.show()

```

Parquet

```
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
youngsterNamesDF.show()

```

Q10. What is a Spatk SQL's UDF?

A10. Spark SQL's User-Defined Functions (UDFs) define new Column-based functions that **extend the vocabulary of Spark SQL's DSL** for transforming Datasets.

```

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.toUpperCas
spark.catalog.listFunctions.filter('name like "%Upper%").show(false)


```

‹ Common Table Expressions (i.e. CTE) in SQL using the "WITH" clause

10+ Key Microservices Interview Questions Answered ›

Arulkumaran

Mechanical Engineer to self-taught Java engineer within 2 years & a **freelancer** within 3 years. Freelancing since 2003. Preparation empowered me to **attend 190+ job interviews** & choose from **150+ job offers** with sought-after contract rates. Author of the book “**Java/J2EE job interview companion**“, which sold **35K+ copies** & superseded by this site with 2,050+ registered users. Amazon.com profile | **Reviews** | LinkedIn | LinkedIn Group | YouTube

Contact us: java-interview@hotmail.com

Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty ltd. The EmpoweringTech pty ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances into consideration. The EmpoweringTech pty ltd will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. Privacy Policy