Conversion from one file format to other in Apache Spark

[February 24, 2018](https://datamazic.blogspot.com/2018/02/conversion-from-one-file-format-to.html)

|  |  |
| --- | --- |
| **Read -->**  **Write**  |  V | **Text file**  sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username retail\_dba --password cloudera \  --table orders \  --target-dir /user/cloudera/ReadDiffFileFormat/text \  --as-textfile  **Read:**  scala> val textFile = sc.textFile("/user/cloudera/ReadDiffFileFormat/text")  textFile: org.apache.spark.rdd.RDD[String] = /user/cloudera/ReadDiffFileFormat/text MapPartitionsRDD[279] at textFile at <console>:30 |
| **Text file** | textFile.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/textout")  Using compression  textFile.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/text/textoutput/compressed", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Sequence file** | For sequence file we need to have a key.  val textMap = textFile.map(e => (e.split(",")(0).toInt, e))  textMap.saveAsSequenceFile("/user/cloudera/ReadDiffFileFormat/sequenceout")  Using compression  textMap.saveAsSequenceFile("/user/cloudera/ReadDiffFileFormat/text/textoutput/sequenceout/compressed", Some(classOf[org.apache.hadoop.io.compress.BZip2Codec])) |
| **JSON file** | val textMapDF = textFile.map(e => (e.split(",")(0).toInt, e.split(",")(1), e.split(",")(2).toInt, e.split(",")(3))).toDF  textMapDF.toJSON.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/jsonout")  Using compression  textMapDF.toJSON.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/text/textoutput/jsonout/compressed", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Parquet** | val textMapDF = textFile.map(e => (e.split(",")(0).toInt, e.split(",")(1), e.split(",")(2).toInt, e.split(",")(3))).toDF  textMapDF.write.parquet("/user/cloudera/ReadDiffFileFormat/parquetout")  Using compression  sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip")  textMapDF.write.parquet("/user/cloudera/ReadDiffFileFormat/text/textoutput/parquetout/compressed") |
| **ORC file** | val textMapDF = textFile.map(e => (e.split(",")(0).toInt, e.split(",")(1), e.split(",")(2).toInt, e.split(",")(3))).toDF  textMapDF.write.orc("/user/cloudera/ReadDiffFileFormat/orcout")  Compression not required. |
| **Avro file** | textMapDF.write.avro("/user/cloudera/ReadDiffFileFormat/avroout")  Using compression  import com.databricks.spark.avro.\_  sqlContext.setConf("spark.sql.avro.compression.codec", "snappy")  textMapDF.write.avro("/user/cloudera/ReadDiffFileFormat/text/textoutput/avroout/compressed1") |

|  |  |
| --- | --- |
| **Read -->**  **Write**  |  V | **JSON file**  **Read:**  val jsonFile = sqlContext.read.json("file:///home/cloudera/workspace/scala/rajudata/data-master/retail\_db\_json/orders") |
| **Text file** | val jsonFileMap = jsonFile.rdd.map(e => (e.getLong(0) + "," + e.getString(1) + "," + e.getLong(2) + "," + e.getString(3)))  jsonFileMap.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/jsontotext1")  Using compression  jsonFileMap.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/json/jsontotext/compressed", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Sequence file** | val jsonFileMap = jsonFile.rdd.map(e => (e.getLong(0), (e.getLong(0) + "," + e.getString(1) + "," + e.getLong(2) + "," + e.getString(3))))  jsonFileMap.saveAsSequenceFile("/user/cloudera/problem5/ReadDiffFileFormat/jsontoSequence")  Using compression  jsonFileMap.saveAsSequenceFile("/user/cloudera/ReadDiffFileFormat/jsontoSequence/compressed", Some(classOf[org.apache.hadoop.io.compress.BZip2Codec])) |
| **JSON file** | jsonFile.toDF.write.json("/user/cloudera/problem5/ReadDiffFileFormat/jsontojson")  Using compression  jsonFile.toJSON.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/jsontojson/compressed", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Parquet** | jsonFile.toDF.write.parquet("/user/cloudera/problem5/ReadDiffFileFormat/jsontoparquet")  Using compression  sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip")  jsonFile.write.parquet("/user/cloudera/ReadDiffFileFormat/jsontoparquet/compressed") |
| **ORC file** | jsonFile.toDF.write.orc("/user/cloudera/problem5/ReadDiffFileFormat/jsontoorc")  scala> jsonFile.toDF.write.orc("/user/cloudera/problem5/ReadDiffFileFormat/jsontoorc")  java.lang.AssertionError: assertion failed: The ORC data source can only be used with HiveContext.  Lets use HiveContext  val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)  val jsonFile = hiveContext.read.json("file:///home/cloudera/workspace/scala/rajudata/data-master/retail\_db\_json/orders")  jsonFile.toDF.write.orc("/user/cloudera/problem5/ReadDiffFileFormat/jsontoorc") |
| **Avro file** | jsonFile.toDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/jsontoavro")  Using compression  import com.databricks.spark.avro.\_  sqlContext.setConf("spark.sql.avro.compression.codec", "gzip") |

|  |  |
| --- | --- |
| **Read -->**  **Write**  |  V | **Parquetfile**  sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username retail\_dba --password cloudera \  --table orders \  --target-dir /user/cloudera/ReadDiffFileFormat/parquet \  --as-parquetfile  Read: Read parquet file  scala> val parquetFile = sqlContext.read.parquet("/user/cloudera/ReadDiffFileFormat/parquet")  parquetFile: org.apache.spark.sql.DataFrame = [order\_id: int, order\_date: bigint, order\_customer\_id: int, order\_status: string] |
| **Text file** | val parquetFileMap = parquetFile.map(e => (e.getInt(0) + "," + e.getLong(1) + "," + e.getInt(2) + "," + e.getString(3)))  parquetFileMap.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/parquettotext")  Using compression  parquetFileMap.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/parquettotext", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Sequence file** | val parquetFileMap = parquetFile.map(e => (e.getInt(0), (e.getInt(0) + "," + e.getLong(1) + "," + e.getInt(2) + "," + e.getString(3))))  parquetFileMap.saveAsSequenceFile("/user/cloudera/ReadDiffFileFormat/parquettotext")  parquetFileMap.saveAsSequenceFile("/user/cloudera/ReadDiffFileFormat/parquettotext", Some(classOf[org.apache.hadoop.io.compress.BZip2Codec])) |
| **JSON file** | val parquetFileDF = parquetFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) ,  e.getString(3))).toDF  parquetFileDF.toJSON.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/parquettojson")  parquetFileDF.toJSON.saveAsTextFile("/user/cloudera/ReadDiffFileFormat/parquettojson", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Parquet** | val parquetFileDF = parquetFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) ,  e.getString(3))).toDF  parquetFileDF.write.parquet("/user/cloudera/problem5/ReadDiffFileFormat/parquettoparquet")  Using compression  sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip")  parquetFileDF.write.parquet("/user/cloudera/ReadDiffFileFormat/jsontoparquet/compressed") |
| **ORC file** | val parquetFileDF = parquetFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) ,  e.getString(3))).toDF  parquetFileDF.write.orc("/user/cloudera/problem5/ReadDiffFileFormat/parquettoorc") |
| **Avro file** | import com.databricks.spark.avro.\_  val parquetFileDF = parquetFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) ,  e.getString(3))).toDF  parquetFileDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/parquettoavro")  Using compression  import com.databricks.spark.avro.\_  sqlContext.setConf("spark.sql.avro.compression.codec", "gzip")  parquetFileDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/parquettoavro/compressed") |

|  |  |
| --- | --- |
| **Read -->**  **Write**  |  V | **ORC file**  **Read:**  val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)  val orcfile = hiveContext.read.orc("/user/cloudera/problem5/ReadDiffFileFormat/parquettoorc") |
| **Text file** | val orcFileMap = orcfile.map(e => (e.getInt(0) + "," + e.getLong(1) + "," + e.getInt(2) + "," + e.getString(3)))  orcFileMap.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/orctotext")  Using compression  orcFileMap.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/orctotext", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Sequence file** | val orcFileMap = orcfile.map(e => (e.getInt(0), (e.getInt(0) + "," + e.getLong(1) + "," + e.getInt(2) + "," + e.getString(3))))  orcFileMap.saveAsSequenceFile("/user/cloudera/problem5/ReadDiffFileFormat/orctoSequence")  Using compression  orcFileMap.saveAsSequenceFile("/user/cloudera/problem5/ReadDiffFileFormat/orctoSequence", Some(classOf[org.apache.hadoop.io.compress.BZip2Codec])) |
| **JSON file** | orcfile.toDF.toJSON.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/orctojson")  Using compression  orcfile.toDF.toJSON.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/orctojson", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Parquet** | orcfile.toDF.write.parquet("/user/cloudera/problem5/ReadDiffFileFormat/orctoparquet")  Using compression  sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip")  orcfile.toDF.write.parquet("/user/cloudera/problem5/ReadDiffFileFormat/orctoparquet") |
| **ORC file** | orcfile.toDF.write.orc("/user/cloudera/problem5/ReadDiffFileFormat/orctoorc") |
| **Avro file** | orcfile.toDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/orctoavro")  Using compression  import com.databricks.spark.avro.\_  sqlContext.setConf("spark.sql.avro.compression.codec", "gzip")  orcfile.toDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/orctoavro") |

|  |  |
| --- | --- |
| **Read -->**  **Write**  |  V | **Avro file**  sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username retail\_dba --password cloudera \  --table orders \  --target-dir /user/cloudera/problem5/ReadDiffFileFormat/avro \  --as-avrodatafile  Read:  val avroFile = sqlContext.read.avro("/user/cloudera/problem5/ReadDiffFileFormat/avro")  scala> avroFile.take(2)  res90: Array[org.apache.spark.sql.Row] = Array([1,1512321804000,11599,CLOSED], [2,1512321804000,256,CLOSED]) |
| **Text file** | val avroFileToText = avroFile.map(e => (e.getInt(0) + "," + e.getLong(1) + "," + e.getInt(2) + "," + e.getString(3)))   avroFileToText.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/avrototext")  Using compression   avroFileToText.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/avrototext", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Sequence file** | val avroFileToSequence = avroFile.map(e => (e.getInt(0), (e.getInt(0) + "," + e.getLong(1) + "," + e.getInt(2) + "," + e.getString(3))))  avroFileToSequence.saveAsSequenceFile("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToSequence")  Using compression  avroFileToSequence.saveAsSequenceFile("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToSequence", Some(classOf[org.apache.hadoop.io.compress.BZip2Codec])) |
| **JSON file** | val avroFileMap = avroFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) , e.getString(3)))  avroFileMap.toDF.toJSON.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/avroFileTojson1")  Using compression  avroFileMap.toDF.toJSON.saveAsTextFile("/user/cloudera/problem5/ReadDiffFileFormat/avroFileTojson1", classOf[org.apache.hadoop.io.compress.BZip2Codec]) |
| **Parquet** | val avroFileMap = avroFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) , e.getString(3)))  avroFileMap.toDF.write.parquet("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToparquet1")  Using compression  sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip")  avroFileMap.toDF.write.parquet("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToparquet1") |
| **ORC file** | val avroFileMap = avroFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) , e.getString(3)))  avroFileMap.toDF.write.orc("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToorc") |
| **Avro file** | val avroFileMap = avroFile.map(e => (e.getInt(0) , e.getLong(1) , e.getInt(2) , e.getString(3)))  avroFileMap.toDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToavro")  Using compression  import com.databricks.spark.avro.\_  sqlContext.setConf("spark.sql.avro.compression.codec", "gzip")  avroFileMap.toDF.write.avro("/user/cloudera/problem5/ReadDiffFileFormat/avroFileToavro") |