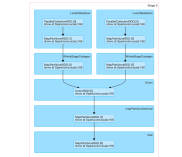
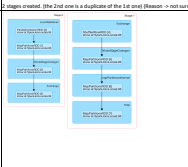
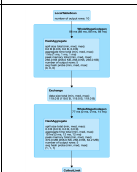
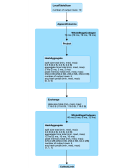
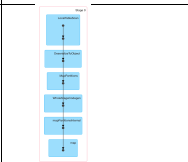
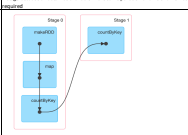
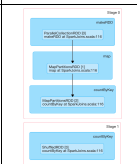
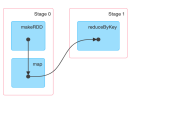
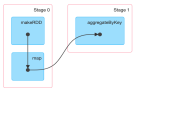
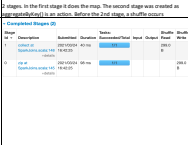
Command	Logical plan	Physical plan	Jobs	stages	Analysis	Code snippet
join	``` == Parsed Logical Plan == 'Join Inner(joinExpr,List(deptid)) :- Project [_1#52 AS empid#22, _2#53 AS empname#23, _3#59 AS empsalary#24, _4#60 AS deptid#25] : +- LocalRelation [_1#52, _2#53, _3#59, _4#60] +- Project [_1#57 AS deptid#27, _2#58 AS deptname#28, _3#5 AS deptmgr#9] +- LocalRelation [_1#57, _2#58, _3#5] == Analyzed Logical Plan == empid: int, empname: string, empsalary: int, deptname: string, deptmgr: string Project [deptid#25, empid#22, empname#23, empsalary#24, deptname#28, deptmgr#9] +- Join Inner, (deptid#25 = deptid#27) :- Project [_1#52 AS empid#22, _2#53 AS empname#23, _3#59 AS empsalary#24, _4#60 AS deptid#25] : +- LocalRelation [_1#52, _2#53, _3#59, _4#60] +- Project [_1#57 AS deptid#27, _2#58 AS deptname#28, _3#5 AS deptmgr#9] +- LocalRelation [_1#57, _2#58, _3#5] == Optimized Logical Plan == Project [deptid#25, empid#22, empname#23, empsalary#24, deptname#28, deptmgr#9] +- Join Inner, (deptid#25 = deptid#27) :- LocalRelation [empid#22, empname#23, empsalary#24, deptid#25] +- LocalRelation [deptid#27, deptname#28, deptmgr#9] ```	``` == Physical Plan == AdaptiveSparkPlan isFinalPlan=false +- SortMergeJoin [deptid#25], [deptid#27], Inner :- Sort [deptid#25, asc] : +- Exchange hashpartitioning(deptid#25, 200), ENSURE_REQUIREMENTS, [plan_id=1] : +- LocalTableScan [empid#22, empname#23, empsalary#24, deptid#25] +- Sort [deptid#27, asc] +- Exchange hashpartitioning(deptid#27, 200), ENSURE_REQUIREMENTS, [plan_id=2] +- LocalTableScan [deptid#27, deptname#28, deptmgr#9] ```	- Completed Jobs (2) Job Id ▽ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total 1 | show at SparkJoins.scala:53 | 2021/03/24 17:19:44 | 77 ms | 1/1 | 1/1 0 | show at SparkJoins.scala:53 | 2021/03/24 10:30:20 | 0.0 s | 1/1 | 1/1 2 jobs created. (the 1st one is the ThreadPoolExecutor and the 2nd one is the join). Spark jobs are related to queries with join operators that fit the definition of BroadcastHashJoin where one join side is broadcast to executors for join. That BroadcastHashJoin operator uses a ThreadPool for this asynchronous broadcasting	 2 stages. In stage 0, the BroadcastHashJoin uses the ThreadPool to transfer the data from one join side to all the other executors		``` val deptdf = Seq((1, "HR", "A"), (2, "Escalation", "B"), (3, "Engineering", "C")).toDF("deptid", "deptname", "deptmgr") val empdf = Seq((1, "A", 25000, 1), (2, "B", 30000, 2), (3, "C", 100000, 3), (4, "D", 50000, 3), (5, "E", 40000, 1)).toDF("empid", "empname", "empsalary", "deptid") val joinedDF = empdf.join(deptdf, Seq("deptid")) val joinedstatsdf = joinedDF.groupBy("deptid").agg(max("empsalary").as("maxsalary"), min("empsalary").as("minsalary"), avg("empsalary").as("avgsalary")) deptdf.show() empdf.show() joinedDF.show() joinedstatsdf.show() deptdf.explain(true) empdf.explain(true) joinedDF.explain(true) joinedstatsdf.explain(true) ```
map	``` == Parsed Logical Plan == Project [_1#9.A0 AS key#12, _2#10.A0 valA#13] +- SerializeFromObject [staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)).._1 AS _1#9, staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)._2 AS _2#10] +- MapElements <function1>, interface org.apache.spark.sql.Row, [StructField(key,IntegerType,false)], obj#8: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, StructFields(key,IntegerType,false)), obj#7: org.apache.spark.sql.Row +- Project [value#1.B0 AS key#3] +- LocalRelation [value#1] == Analyzed Logical Plan == key: int, valA: int Project [_1#9.A0 AS key#12, _2#10.A0 valA#13] +- SerializeFromObject [staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)).._1 AS _1#9, staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)._2 AS _2#10] +- MapElements <function1>, obj#8: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, StructFields(key,IntegerType,false)), obj#7: org.apache.spark.sql.Row +- Project [value#1.B0 AS key#3] +- LocalRelation [value#1] == Optimized Logical Plan == key: int, valA: int Project [_1#9.A0 AS key#12, _2#10.A0 valA#13] +- SerializeFromObject [staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)).._1 AS _1#9, staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)._2 AS _2#10] +- MapElements <function1>, obj#8: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, StructFields(key,IntegerType,false)), obj#7: org.apache.spark.sql.Row +- LocalRelation [key#3] ```	``` == Physical Plan == AdaptiveSparkPlan isFinalPlan=false +- Project [_1#9.A0 key#12, _2#10.A0 valA#13] +- SerializeFromObject [staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)).._1 AS _1#9, staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ScalaTuple2, true)._2 AS _2#10] +- MapElements <function1>, obj#8: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, StructFields(key,IntegerType,false)), obj#7: org.apache.spark.sql.Row +- LocalTableScan [key#3] ```	- Completed Jobs (1) Job Id ▽ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total 0 | show at SparkJoins.scala:67 show at SparkJoins.scala:67 | 2021/03/24 10:09:17 | 0.0 s | 1/1 | 1/1 1 job because there is only one action (here it is the show() function)	 1 stage because all data was available in the same partition and no shuffle was required		``` val seq1 = Seq(1, 2, 3, 1, 1, 5, 2, 3, 3, 4).toDF("key") val mappedtestdf = testdf.map(x => (x.getInt(0), 1).toDF("key", "val") mappedtestdf.show() mappedtestdf.explain(true) ```
mapPartitions						
union	``` == Parsed Logical Plan == 'Union :- Project [_1#56 AS empid#26, _2#57 AS empname#21, _3#58 AS empsalary#22] : +- LocalRelation [_1#56, _2#57, _3#58] +- Project [_1#63 AS deptid#7, _2#64 AS deptname#8, _3#5 AS deptmgr#9] +- LocalRelation [_1#63, _2#64, _3#5] == Analyzed Logical Plan == empid: int, empname: string, empsalary: string Union :- Project [cast(empid#26 as int) AS empid#24] : +- Project [_1#56 AS empid#26, _2#57 AS empname#21, _3#58 AS empsalary#22] : +- LocalRelation [_1#56, _2#57, _3#58] +- Project [cast(deptid#7, deptname#8, deptmgr#9)] +- Project [_1#63 AS deptid#7, _2#64 AS deptname#8, _3#5 AS deptmgr#9] +- LocalRelation [_1#63, _2#64, _3#5] == Optimized Logical Plan == Union :- LocalRelation [empid#26, empname#21, empsalary#44] +- LocalRelation [deptid#7, deptname#8, deptmgr#9] ```	``` == Physical Plan == Union :- LocalTableScan [empid#26, empname#21, empsalary#44] +- LocalTableScan [deptid#7, deptname#8, deptmgr#9] ```	- Completed Jobs (2) Job Id ▽ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total 1 | show at SparkJoins.scala:100 | 2021/03/24 18:10:41 | 13 ms | 1/1 | 1/1 0 | show at SparkJoins.scala:100 | 2021/03/24 18:10:41 | 0.0 s | 1/1 | 1/1 2 jobs created. (the 2nd one is a duplicate of the 1st one) (Reason -> not sure)	 2 stages created. (the 2nd one is a duplicate of the 1st one) (Reason -> not sure)		``` val deptdf = Seq((1, "HR", "A"), (2, "Escalation", "B"), (3, "Engineering", "C")).toDF("deptid", "deptname", "deptmgr") val empdf = Seq((1, "A", 25000), (2, "B", 30000), (3, "C", 100000), (4, "D", 50000), (5, "E", 40000)).toDF("empid", "empname", "empsalary") val unionDDF = empdf.union(deptdf) unionDDF.show() unionDDF.explain(true) ```
groupBy	``` == Parsed Logical Plan == 'Aggregate [key#31], [key#31, count(1) AS count#7L] +- Project [value#1.A0 AS key#3] +- LocalRelation [value#1] == Analyzed Logical Plan == key: int, count: bigint Aggregate [key#31], [key#31, count(1) AS count#7L] +- Project [value#1.A0 AS key#3] +- LocalRelation [value#1] == Optimized Logical Plan == Aggregate [key#31], [key#31, count(1) AS count#7L] +- LocalRelation [key#31] ```	``` == Physical Plan == AdaptiveSparkPlan isFinalPlan=false +- HashAggregate(keys=[key#31], functions=[count(1)], output=[key#31, count#7L]) +- Exchange hashpartitioning(key#31, 200), ENSURE_REQUIREMENTS, [plan_id=1] +- HashAggregate(keys=[key#31], functions=[partial_count(1)], output=[key#31, count#43L]) +- LocalTableScan [key#31] ```	- Completed Jobs (1) Job Id ▽ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total 0 | show at SparkJoins.scala:148 | 2021/03/24 18:10:41 | 0.0 s | 1/1 | 1/1 1 job created. Job 0 was executed. Job 1,2,3,4 were skipped			``` val testdf = Seq(1, 2, 1, 1, 1, 5, 2, 3, 3, 4).toDF("key") val groupedDF1 = testdf.groupBy("key").count() groupedDF1.show() groupedDF1.explain(true) ```
groupByKey	``` == Parsed Logical Plan == Project [value#6.A0 key#13, count(1)#10L AS count#14L] +- Aggregate [value#6], [value#6, count(1) AS count(1)#10L] +- AppendColumns <function1>, interface org.apache.spark.sql.Row, [StructField(key,IntegerType,false)], createexternalrow(key#5, StructField(key,IntegerType,false)), [input[0, int, false] AS value#6] +- Project [value#1.A0 key#5] +- LocalRelation [value#1] == Analyzed Logical Plan == key: int, count: bigint Project [value#6.A0 key#13, count(1)#10L AS count#14L] +- Aggregate [value#6], [value#6, count(1) AS count(1)#10L] +- AppendColumns <function1>, interface org.apache.spark.sql.Row, [StructField(key,IntegerType,false)], createexternalrow(key#5, StructField(key,IntegerType,false)), [input[0, int, false] AS value#6] +- Project [value#1.A0 key#5] +- LocalRelation [value#1] == Optimized Logical Plan == Aggregate [value#6], [value#6 AS key#13, count(1) AS count#14L] +- Project [value#6] +- AppendColumns <function1>, interface org.apache.spark.sql.Row, [StructField(key,IntegerType,false)], createexternalrow(key#5, StructField(key,IntegerType,false)), [input[0, int, false] AS value#6] +- LocalRelation [key#5] ```	``` == Physical Plan == AdaptiveSparkPlan isFinalPlan=false +- Project [_1#13 AS key#13, _2#14 AS count#14L] +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._1), true, false) AS _1#13, staticinvoke(class scala.Long, BoxedLongType, unboxToLong, assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._2), true, false) AS _2#14] +- MapElements <function1>, obj#12: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(lambdavariable(MapObjects_loopValue0, MapObjects_loopIsNull0, StringType, true), true), false), true), false), staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._2), true, false) AS _2#10] +- MapPartitions <function1>, obj#9: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(lambdavariable(MapObjects_loopValue0, MapObjects_loopIsNull0, StringType, true), true), false), true), false), staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(lambdavariable(MapObjects_loopValue0, MapObjects_loopIsNull0, StringType, true), true), false), true), false), StructField(Products,ArrayType(StringType,true),true), obj#11: org.apache.spark.sql.Row +- LocalTableScan [_1#1, _2#2] ```	- Completed Jobs (2) Job Id ▽ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total 1 | show at SparkJoins.scala:148 | 2021/03/24 18:42:20 | 0 ms | 1/1 | 1/1 0 | show at SparkJoins.scala:148 | 2021/03/24 18:42:20 | 0.0 s | 1/1 | 1/1 2 jobs created. Job 0 was executed. Job 1,2,3,4 were skipped			``` val testdf = Seq(1, 2, 1, 1, 1, 5, 2, 3, 3, 4).toDF("key") val groupedDF2 = testdf.groupBy(key => x.getInt(0).count().toDF("key", "count") groupedDF2.show() groupedDF2.explain(true) ```
flatMap	``` == Parsed Logical Plan == 'Project [unresolvedalias(_1#14.Company, None), _2#14 ProductsAS] +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._1), true, false) AS _1#13, staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._2), true, false) AS _2#14] +- MapPartitions <function1>, obj#12: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(lambdavariable(MapObjects_loopValue0, MapObjects_loopIsNull0, StringType, true), true), false), true), false), StructField(Company,StringType,true), StructField(Products,ArrayType(StringType,true),true), obj#11: org.apache.spark.sql.Row +- Project [_1#1.A0 Company#5, _2#1.A0 Products#6] +- LocalRelation [_1#1, _2#2] ```	``` == Physical Plan == AdaptiveSparkPlan isFinalPlan=false +- Project [_1#13 AS Company#5, _2#14 AS Products#6] +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._1), true, false) AS _1#13, staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(assertnotnull(input[0, scala.Tuple2, true])._2), true, false) AS _2#14] +- MapPartitions <function1>, obj#12: scala.Tuple2 +- DeserializeToObject staticinvoke(class com.sparkjoins.Utilsspark.SparkJoins$, ObjectType(interface scala.collection.Seq), make_array(assertnotnull(lambdavariable(MapObjects_loopValue0, MapObjects_loopIsNull0, StringType, true), true), false), true), false), StructField(Company,StringType,true), StructField(Products,ArrayType(StringType,true),true), obj#11: org.apache.spark.sql.Row +- LocalTableScan [Company#5, Products#6] ```	1 job because there is only one action (here it is the show() function)	 1 stage because all data was available in the same partition and no shuffle was required		``` val testdf = Seq(("Google", Seq("Chrome", "Gmail", "Drive", "Android", "YouTube")), ("Apple", Seq("iPhone", "iPad", "Mac", "Apple Music", "AirPods")), ("Samsung", Seq("Galaxy", "TV")),).toDF("Company", "Products") val flatmappedDF = testdf.flatMap(x => x.getSeq[String](1).map(x.getString(0), _)).toDF("Company", "Product") flatmappedDF.show() flatmappedDF.explain(true) ```
countByKey	NA	``` *(1) MapPartitionsRDD[1] at map at SparkJoins.scala:117 [] | ParallelCollectionRDD[0] at makeRDD at SparkJoins.scala:116 []* ```	1 job because there is only one action	 2 stages. In the first stage it does the map and also the countByKey. The second stage was created as countByKey() is an action. The stage 1 would have more DAG, if more transformations and actions are performed in it. A stage comprises of all the transformations till an action is occured **Note: countByKey does not require a shuffle**		``` val testrdd = sc.makeRDD(Seq(1, 2, 3, 1, 1, 5, 2, 3, 3, 4)) val rdd1 = testrdd.map(x => (x, 1)) val rdd2 = rdd1.countByKey() println(rdd2.toDebugString) rdd2.foreach(println) ```
reduceByKey	NA	``` *(1) ShuffledRDD[2] at reduceByKey at SparkJoins.scala:127 [] +-(1) MapPartitionsRDD[1] at map at SparkJoins.scala:126 [] | ParallelCollectionRDD[0] at makeRDD at SparkJoins.scala:125 []* ```	1 job because there is only one action	 2 stages. In the first stage it does the map. The second stage was created as reduceByKey() is an action. Before the 2nd stage, a shuffle occurs		``` val testrdd = sc.makeRDD(Seq(1, 2, 3, 1, 1, 5, 2, 3, 3, 4)) val mappedrdd = testrdd.map(x => (x, 1)) val reducedrdd = mappedrdd.reduceByKey(_ + _) println(reducedrdd.toDebugString) reducedrdd.collect.foreach(println) ```
aggregateByKey	NA	``` *(1) ShuffledRDD[2] at aggregateByKey at SparkJoins.scala:136 [] +-(1) MapPartitionsRDD[1] at map at SparkJoins.scala:135 [] | ParallelCollectionRDD[0] at makeRDD at SparkJoins.scala:134 []* ```	1 job because there is only one action	 2 stages. In the first stage it does the map. The second stage was created as aggregateByKey() is an action. Before the 2nd stage, a shuffle occurs		``` val testrdd = sc.makeRDD(Seq(1, 2, 3, 1, 1, 5, 2, 3, 3, 4)) val mappedrdd = testrdd.map(x => (x, 1)) val aggregatedrdd = mappedrdd.aggregateByKey(0(_ + _, _ + _) println(aggregatedrdd.toDebugString) aggregatedrdd.collect.foreach(println) ```
combineByKey	NA	``` *(1) ShuffledRDD[3] at combineByKey at SparkJoins.scala:146 [] +-(1) MapPartitionsRDD[2] at map at SparkJoins.scala:145 [] | ParallelCollectionRDD[1] at makeRDD at SparkJoins.scala:144 [] | ParallelCollectionRDD[0] at makeRDD at SparkJoins.scala:143 []* ```	- Completed Jobs (1) Job Id ▽ | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write 0 | collect at SparkJoins.scala:149 collect at SparkJoins.scala:149 | 2021/03/24 18:46:20 | 40 ms | 2/2 | | | 299.0 B | 0 | zip at SparkJoins.scala:149 zip at SparkJoins.scala:149 | 2021/03/24 18:46:20 | 66 ms | 2/2 | | | | 299.0 B 2 jobs created			``` val rdd1 = sc.makeRDD(List("dog", "cat", "salmon", "python", "wolf", "bear", "turtle", "bee")) val rdd2 = sc.makeRDD(List(1, 2, 2, 3, 1, 1, 2, 4)) val rdd3 = rdd1.zip(rdd2) val combinedrdd = rdd3.combineByKey(List(_), (x: List[String], y: String) => y :: x, (x: List[String], y: List[String]) => x ::: y) println(combinedrdd.toDebugString) combinedrdd.collect.foreach(println) ```
shuffling	Which job shuffling will come? The Spark SQL shuffle is a mechanism for redistributing or re-partitioning data so that the data grouped differently across partitions Spark shuffle is a mechanism Spark uses to redistribute the data across different executors and even across machines. Shuffle is an expensive operation since it involves the following - Disk I/O - Involves data serialization and deserialization - Network I/O Spark RDD triggers shuffle for several operations like repartition(), coalesce(), groupByKey(), reduceByKey(), cogroup() and join()	NA	NA	NA	NA	NA