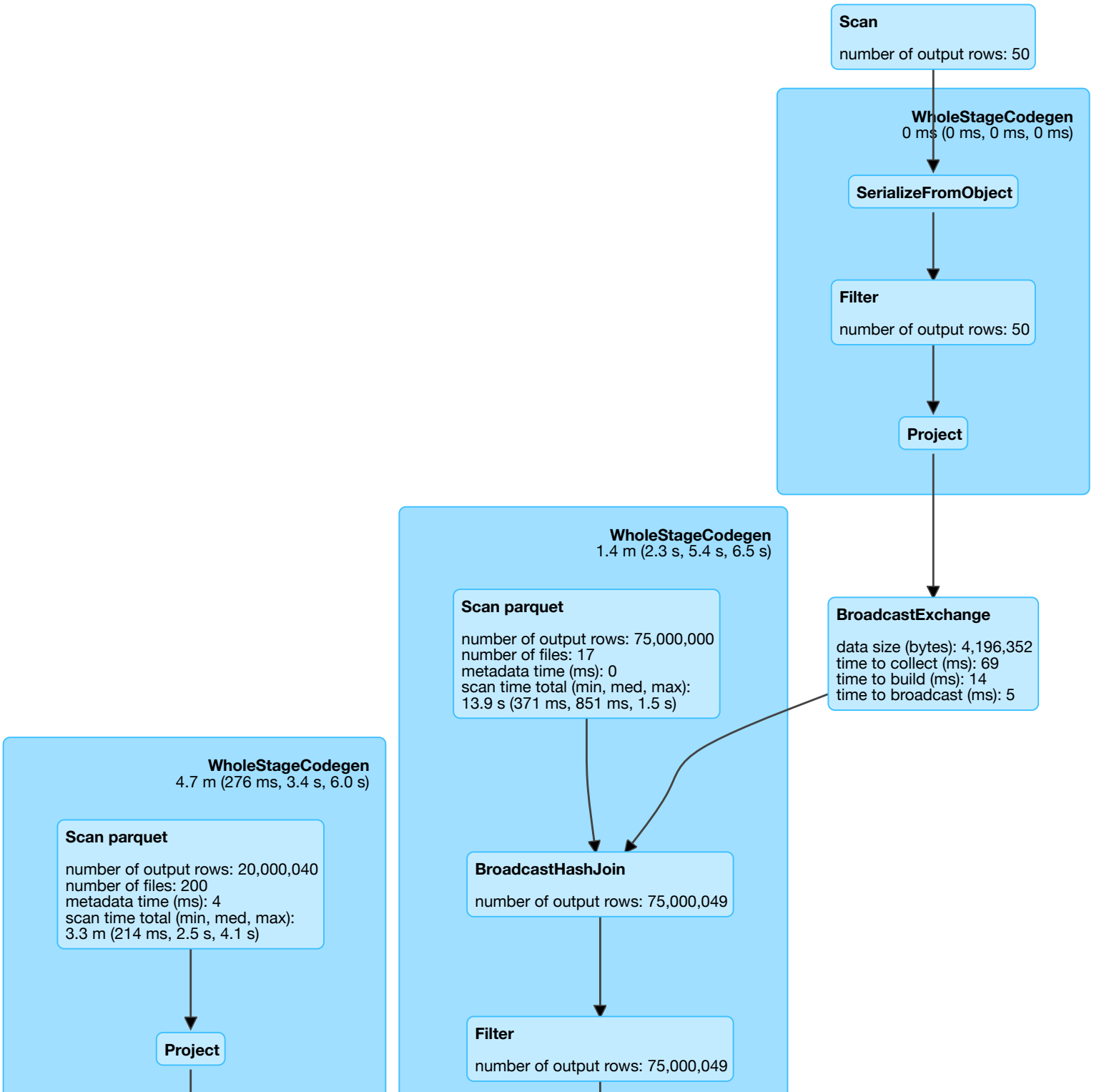
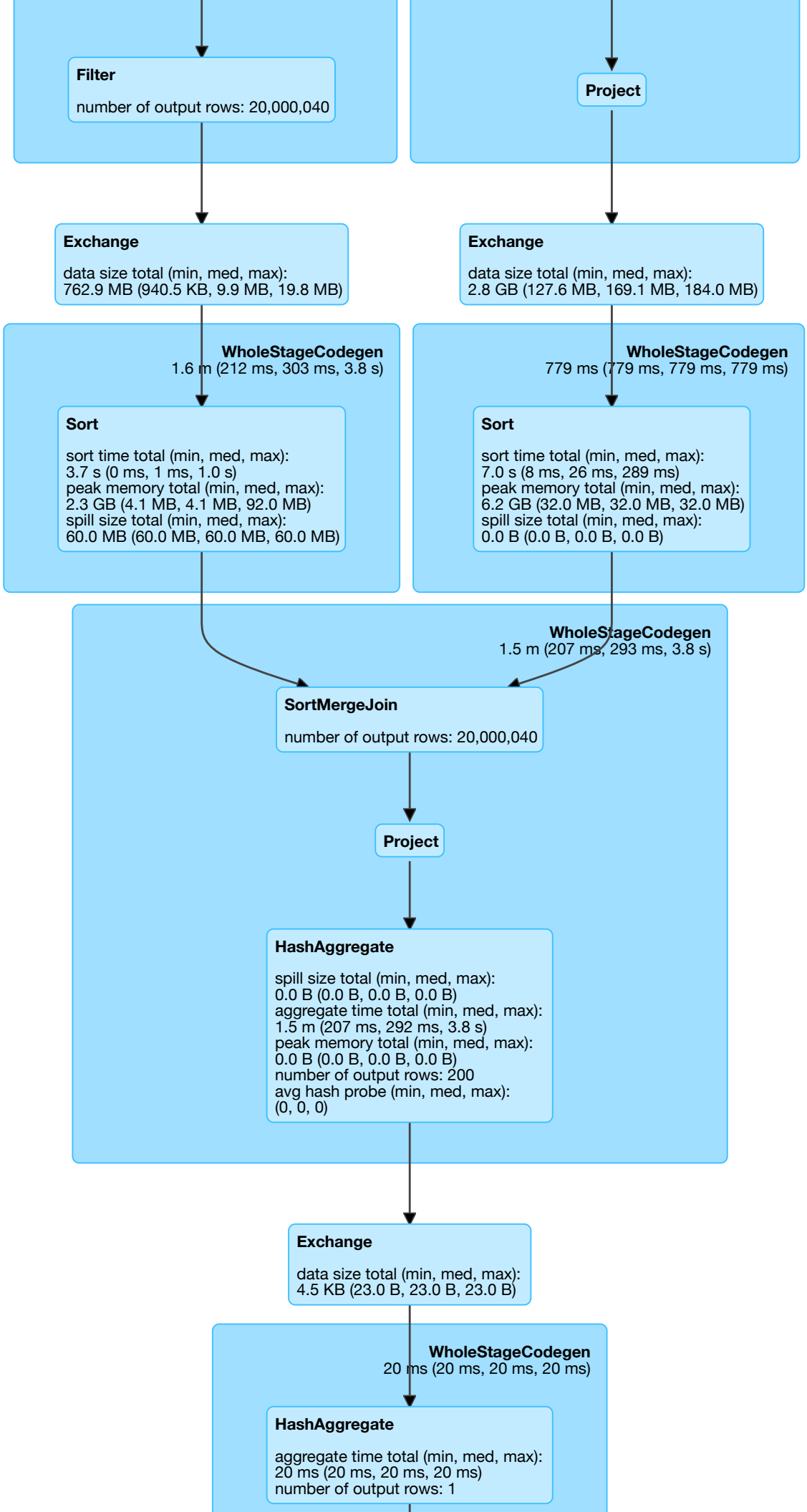
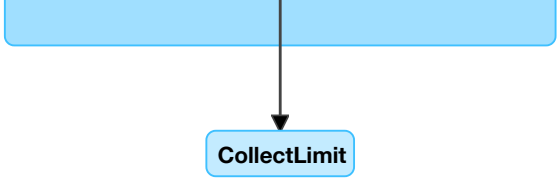
Details for Query 5

Submitted Time: 2020/05/07 11:38:45
Duration: 43 s
Succeeded Jobs: 4 (/jobs/job/?id=4) 5 (/jobs/job/?id=5)




Filter
number of output rows: 20,000,040
Project
Exchange
data size total (min, med, max):
762.9 MB (940.5 KB, 9.9 MB, 19.8 MB)
Exchange
data size total (min, med, max):
2.8 GB (127.6 MB, 169.1 MB, 184.0 MB)
WholeStageCodegen
1.6 m (212 ms, 303 ms, 3.8 s)
Sort
sort time total (min, med, max):
3.7 s (0 ms, 1 ms, 1.0 s)
peak memory total (min, med, max):
2.3 GB (4.1 MB, 4.1 MB, 92.0 MB)
spill size total (min, med, max):
60.0 MB (60.0 MB, 60.0 MB, 60.0 MB)
WholeStageCodegen
779 ms (779 ms, 779 ms, 779 ms)
Sort
sort time total (min, med, max):
7.0 s (8 ms, 26 ms, 289 ms)
peak memory total (min, med, max):
6.2 GB (32.0 MB, 32.0 MB, 32.0 MB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
WholeStageCodegen
1.5 m (207 ms, 293 ms, 3.8 s)
SortMergeJoin
number of output rows: 20,000,040
Project
HashAggregate
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
aggregate time total (min, med, max):
1.5 m (207 ms, 292 ms, 3.8 s)
peak memory total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
number of output rows: 200
avg hash probe (min, med, max):
(0, 0, 0)
Exchange
data size total (min, med, max):
4.5 KB (23.0 B, 23.0 B, 23.0 B)
WholeStageCodegen
20 ms (20 ms, 20 ms, 20 ms)
HashAggregate
aggregate time total (min, med, max):
20 ms (20 ms, 20 ms, 20 ms)
number of output rows: 1



Details

```
== Parsed Logical Plan ==
GlobalLimit 21
+- LocalLimit 21
   +- Project [cast(avg_revenue#71 as string) AS avg_revenue#75]
      +- Aggregate [avg((cast(price#2 as double) * cast(num_pieces_sold#10 as double))) AS avg_revenue#71]
         +- Join Inner, (salted_product_id#52 = salted_key1#63)
            :- SubqueryAlias `o`
            :  +- SubqueryAlias `orders_salted`
            :     +- Project [order_id#6, product_id#7, seller_id#8, date#9, num_pieces_sold#10,
bill_raw_text#11, CASE WHEN product_id#7 IN (0) THEN concat(product_id#7, -,
cast(cast(round((rand(815980494930188591) * cast(49 as double)), 0) as int) as string)) ELSE product_id#7 END AS
salted_key1#63]
            :        +- Relation[order_id#6,product_id#7,seller_id#8,date#9,num_pieces_sold#10,bill_raw_text#11]
parquet
            +- SubqueryAlias `p`
               +- SubqueryAlias `products_salted`
                  +- Project [product_id#0, product_name#1, price#2, salted_product_id#52]
                     +- Project [product_id#0, product_name#1, price#2, original_key#33, salt#34, CASE WHEN
isnull(salt#34) THEN product_id#0 ELSE concat(product_id#0, -, salt#34) END AS salted_product_id#52]
                        +- Join LeftOuter, (product_id#0 = original_key#33)
                           :- Relation[product_id#0,product_name#1,price#2] parquet
                           +- ResolvedHint (broadcast)
                              +- Project [_1#29 AS original_key#33, _2#30 AS salt#34]
                                 +- SerializeFromObject [staticinvoke(class
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(assertnotnull(input[0,
scala.Tuple2, true]))._1, true, false) AS _1#29, staticinvoke(class org.apache.spark.unsafe.types.UTF8String,
StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._2, true, false) AS _2#30]
                                    +- ExternalRDD [obj#28]

== Analyzed Logical Plan ==
avg_revenue: string
GlobalLimit 21
+- LocalLimit 21
   +- Project [cast(avg_revenue#71 as string) AS avg_revenue#75]
      +- Aggregate [avg((cast(price#2 as double) * cast(num_pieces_sold#10 as double))) AS avg_revenue#71]
         +- Join Inner, (salted_product_id#52 = salted_key1#63)
            :- SubqueryAlias `o`
            :  +- SubqueryAlias `orders_salted`
            :     +- Project [order_id#6, product_id#7, seller_id#8, date#9, num_pieces_sold#10,
bill_raw_text#11, CASE WHEN product_id#7 IN (0) THEN concat(product_id#7, -,
cast(cast(round((rand(815980494930188591) * cast(49 as double)), 0) as int) as string)) ELSE product_id#7 END AS
salted_key1#63]
            :        +- Relation[order_id#6,product_id#7,seller_id#8,date#9,num_pieces_sold#10,bill_raw_text#11]
parquet
            +- SubqueryAlias `p`
               +- SubqueryAlias `products_salted`
                  +- Project [product_id#0, product_name#1, price#2, salted_product_id#52]
                     +- Project [product_id#0, product_name#1, price#2, original_key#33, salt#34, CASE WHEN
isnull(salt#34) THEN product_id#0 ELSE concat(product_id#0, -, salt#34) END AS salted_product_id#52]
                        +- Join LeftOuter, (product_id#0 = original_key#33)
                           :- Relation[product_id#0,product_name#1,price#2] parquet
                           +- ResolvedHint (broadcast)
                              +- Project [_1#29 AS original_key#33, _2#30 AS salt#34]
                                 +- SerializeFromObject [staticinvoke(class
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(assertnotnull(input[0,
scala.Tuple2, true]))._1, true, false) AS _1#29, staticinvoke(class org.apache.spark.unsafe.types.UTF8String,
StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._2, true, false) AS _2#30]
                                    +- ExternalRDD [obj#28]

== Optimized Logical Plan ==
GlobalLimit 21
```

```
+- LocalLimit 21
   +- Aggregate [cast(avg((cast(price#2 as double) * cast(num_pieces_sold#10 as double))) as string) AS
avg_revenue#75]
      +- Project [num_pieces_sold#10, price#2]
         +- Join Inner, (salted_product_id#52 = salted_key1#63)
            :- Filter isnotnull(salted_key1#63)
            :  +- Project [num_pieces_sold#10, CASE WHEN (product_id#7 = 0) THEN concat(product_id#7, -,
cast(cast(round((rand(8159804949301885 91) * 49.0), 0) as int) as string)) ELSE product_id#7 END AS
salted_key1#63]
            :     +- Relation[order_id#6,product_id#7,seller_id#8,date#9,num_pieces_sold#10,bill_raw_text#11]
parquet
            +- Project [price#2, CASE WHEN isnull(salt#34) THEN product_id#0 ELSE concat(product_id#0, -,
salt#34) END AS salted_product_id#52]
               +- Filter isnotnull(CASE WHEN isnull(salt#34) THEN product_id#0 ELSE concat(product_id#0, -,
salt#34) END)
                  +- Join LeftOuter, (product_id#0 = original_key#33)
                     :- Project [product_id#0, price#2]
                     :  +- Relation[product_id#0,product_name#1,price#2] parquet
                     +- ResolvedHint (broadcast)
                        +- Project [_1#29 AS original_key#33, _2#30 AS salt#34]
                           +- Filter isnotnull(_1#29)
                              +- SerializeFromObject [staticinvoke(class
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(input[0, scala.Tuple2,
true])._1, true, false) AS _1#29, staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType,
fromString, assertnotnull(input[0, scala.Tuple2, true])._2, true, false) AS _2#30]
                                 +- ExternalRDD [obj#28]

== Physical Plan ==
CollectLimit 21
+- *(7) HashAggregate(keys=[], functions=[avg((cast(price#2 as double) * cast(num_pieces_sold#10 as double)))],
output=[avg_revenue#75])
   +- Exchange SinglePartition
      +- *(6) HashAggregate(keys=[], functions=[partial_avg((cast(price#2 as double) * cast(num_pieces_sold#10
as double)))], output=[sum#79, count#80L])
         +- *(6) Project [num_pieces_sold#10, price#2]
            +- *(6) SortMergeJoin [salted_key1#63], [salted_product_id#52], Inner
               :- *(2) Sort [salted_key1#63 ASC NULLS FIRST], false, 0
               :  +- Exchange hashpartitioning(salted_key1#63, 200)
               :     +- *(1) Filter isnotnull(salted_key1#63)
               :        +- *(1) Project [num_pieces_sold#10, CASE WHEN (product_id#7 = 0) THEN
concat(product_id#7, -, cast(cast(round((rand(8159804949301885 91) * 49.0), 0) as int) as string)) ELSE
product_id#7 END AS salted_key1#63]
               :           +- *(1) FileScan parquet
[order_id#6,product_id#7,seller_id#8,date#9,num_pieces_sold#10,bill_raw_text#11] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[file:/Users/o60774/Downloads/product-sales/sales_parquet/part-00102-e651a798-93...,
PartitionFilters: [], PushedFilters: [], ReadSchema:
struct<order_id:string,product_id:string,seller_id:string,date:string,num_pieces_sold:string,bill...
               +- *(5) Sort [salted_product_id#52 ASC NULLS FIRST], false, 0
                  +- Exchange hashpartitioning(salted_product_id#52, 200)
                     +- *(4) Project [price#2, CASE WHEN isnull(salt#34) THEN product_id#0 ELSE
concat(product_id#0, -, salt#34) END AS salted_product_id#52]
                        +- *(4) Filter isnotnull(CASE WHEN isnull(salt#34) THEN product_id#0 ELSE
concat(product_id#0, -, salt#34) END)
                           +- *(4) BroadcastHashJoin [product_id#0], [original_key#33], LeftOuter, BuildRight
                              :- *(4) FileScan parquet [product_id#0,price#2] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[file:/Users/o60774/Downloads/product-sales/products_parquet/part-00015-0f978a44...,
PartitionFilters: [], PushedFilters: [], ReadSchema: struct<product_id:string,price:string>
                              +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true]))
                                 +- *(3) Project [_1#29 AS original_key#33, _2#30 AS salt#34]
                                    +- *(3) Filter isnotnull(_1#29)
                                       +- *(3) SerializeFromObject [staticinvoke(class
```

```
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(input[0, scala.Tuple2,
true])._1, true, false) AS _1#29, staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType,
fromString, assertnotnull(input[0, scala.Tuple2, true])._2, true, false) AS _2#30]
                                          +- Scan[obj#28]
```