Systems for Data Science - HW10

SQL Queries Using Spark

Shubham Shetty

March 27 2021

1 Background

SQL query executed using Spark is -

```
SELECT DISTINCT a.student_name
FROM df_student a
INNER JOIN df_attend b
ON a.student_id = b.student_id
INNER JOIN df_course c
ON b.course_id = c.course_id
WHERE c.course_name = <Subject>
```

Here the value "Subject" is passed by user as a command line argument.

Input files taken for this assignment in CSV form were -

1. *course.csv*

Course ID	Course Name
1	OOPS
2	D&OS
3	ML
4	Systems
5	AI
6	BIA

2. *student.csv*

Student ID	Student Name
1	Aaron
2	Bobby
3	Darren
4	Eric
5	Francis
6	George
7	Henry
8	Jack
9	Killian
10	Leo

3. *attend.csv*

Student ID	Course ID
1	2
1	4
1	5
2	1
2	2
2	3
3	2
3	5
3	6
4	1
4	5
4	6
5	1
5	5
5	6
6	2
6	3
6	4
7	1
7	4
7	6
8	2
8	3
8	4
9	1
9	5
9	6
10	2
10	4
10	5

2 Physical Plan

Calling the SparkSQL EXPLAIN command (with "FORMATTED" option) after running the query results in following output -

```
|== Physical Plan ==
* HashAggregate (16)
+- Exchange (15)
   +- * HashAggregate (14)
      +- * Project (13)
         +- * BroadcastHashJoin Inner BuildRight (12)
            :- * Project (7)
            :  +- * BroadcastHashJoin Inner BuildLeft (6)
            :     :- BroadcastExchange (3)
            :     :  +- * Filter (2)
            :     :     +- Scan csv  (1)
            :     +- * Filter (5)
            :        +- Scan csv  (4)
            +- BroadcastExchange (11)
               +- * Project (10)
                  +- * Filter (9)
                     +- Scan csv  (8)


(1) Scan csv
Output [2]: [student_id#16, student_name#17]
Batched: false
Location: InMemoryFileIndex [file:/Users/shubhamshetty/
   Documents/UMass/532/HW10/student.csv]
PushedFilters: [IsNotNull(student_id)]
ReadSchema: struct<student_id:string,student_name:string>

(2) Filter [codegen id : 1]
Input [2]: [student_id#16, student_name#17]
Condition : isnotnull(student_id#16)

(3) BroadcastExchange
Input [2]: [student_id#16, student_name#17]
Arguments: HashedRelationBroadcastMode(List(input[0, string,
   false]),false), [id=#135]

(4) Scan csv
Output [2]: [student_id#56, course_id#57]
```

```
Batched: false
Location: InMemoryFileIndex [file:/Users/shubhamshetty/
   Documents/UMass/532/HW10/attend.csv]
PushedFilters: [IsNotNull(student_id), IsNotNull(course_id)]
ReadSchema: struct<student_id:string,course_id:string>

(5) Filter
Input [2]: [student_id#56, course_id#57]
Condition : (isnotnull(student_id#56) AND isnotnull(course_id
   #57))

(6) BroadcastHashJoin [codegen id : 3]
Left keys [1]: [student_id#16]
Right keys [1]: [student_id#56]
Join condition: None

(7) Project [codegen id : 3]
Output [2]: [student_name#17, course_id#57]
Input [4]: [student_id#16, student_name#17, student_id#56,
   course_id#57]

(8) Scan csv
Output [2]: [course_id#36, course_name#37]
Batched: false
Location: InMemoryFileIndex [file:/Users/shubhamshetty/
   Documents/UMass/532/HW10/course.csv]
PushedFilters: [IsNotNull(course_name), EqualTo(course_name,
   OOPS), IsNotNull(course_id)]
ReadSchema: struct<course_id:string,course_name:string>

(9) Filter [codegen id : 2]
Input [2]: [course_id#36, course_name#37]
Condition : ((isnotnull(course_name#37) AND (course_name#37 =
   OOPS)) AND isnotnull(course_id#36))

(10) Project [codegen id : 2]
Output [1]: [course_id#36]
Input [2]: [course_id#36, course_name#37]

(11) BroadcastExchange
Input [1]: [course_id#36]
Arguments: HashedRelationBroadcastMode(List(input[0, string,
   true]),false), [id=#145]
```

```
(12) BroadcastHashJoin [codegen id : 3]
Left keys [1]: [course_id#57]
Right keys [1]: [course_id#36]
Join condition: None

(13) Project [codegen id : 3]
Output [1]: [student_name#17]
Input [3]: [student_name#17, course_id#57, course_id#36]

(14) HashAggregate [codegen id : 3]
Input [1]: [student_name#17]
Keys [1]: [student_name#17]
Functions: []
Aggregate Attributes: []
Results [1]: [student_name#17]

(15) Exchange
Input [1]: [student_name#17]
Arguments: hashpartitioning(student_name#17, 200),
   ENSURE_REQUIREMENTS, [id=#151]

(16) HashAggregate [codegen id : 4]
Input [1]: [student_name#17]
Keys [1]: [student_name#17]
Functions: []
Aggregate Attributes: []
Results [1]: [student_name#17]
```

Interpretation of Physical Plan

From the formatted ouput of SparkSQL EXPLAIN command, following can be inferred-

1. *Scan csv*: student.csv is being loaded into dataframe.
2. *Filter*: "isnotnull" condition is being checked for primary key "student_id".
3. *BroadcastExchange*: Columns from student dataframe being broadcast.
4. *Scan csv*: attend.csv is being loaded into dataframe.
5. *Filter*: "isnotnull" condition is being checked for foreign keys "student_id" and "course_id".

6. *BroadcastHashJoin*: Hash equi-join on key=student_id is performed and broadcast.

7. *Project*: Retain relevant columns only (student_name and course_id).

8. *Scan csv*: course.csv is being loaded into dataframe.

9. *Filter*: "isnotnull" condition is being checked for primary key "course_id", and filtered for selected "course_name".

10. *Project*: Take relevant column only.

11. *BroadcastExchange*: Columns from student dataframe being broadcast.

12. *BroadcastHashJoin*: Hash equi-join on key=course_id is performed and broadcast.

13. *Project*: Take relevant columns only.

14. *HashAggregate*: Taking distinct student_name column by performing HashAggregate function

15. *Exchange*: Exchange hash partitions for column student_name

16. *HashAggregate*: Perform HashAggregate on student_name column.

Implementation Using RDDs

For implementing code to mimic output of the same query, following functions of RDD would have to be used[1] -

1. *map*: Return a new distributed dataset formed by passing each element of the source through a function func.

2. *filter*: Return a new dataset formed by selecting those elements of the source on which func returns true.

3. *distinct*: Return a new dataset that contains the distinct elements of the source dataset

4. *join*: When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key.

5. *collect*: Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

[1]definitions for RDD functions taken from official Spark documentation (`https://spark.apache.org/docs/latest/rdd-programming-guide.html`)