SELECT A.row_num, B.col_num, SUM(A.value * B.value) AS value FROM matrixA A INNER JOIN matrixB B ON A.col_num = B.row_num GROUP BY A.row_num, B.col_num;

In [ ]:

```
#nombres de colonnes de A = nombre de lignes de B donc il faut effectuer un join sur ça
#matrices are coded as (number row, number column, value)
```

In [ ]:

```
import findspark
findspark.init()
import pyspark
sc = pyspark.SparkContext(appName="MatrixMultiplicationSQL")
```

SMALL MATRIX EXAMPLE

In [ ]:

```
def product_sum( listoftuples) :
    i = 0
    for tuple_ in listoftuples :
        i = i + tuple_[0]*tuple_[1]
    return i
```

In [ ]:

```
matrixA = sc.parallelize([(1,1,1),(1,2,2),(1,3,3),(2,1,2),(2,2,5),(2,3,7)])
matrixA.collect()
```

In [ ]:

```
matrixB = sc.parallelize([(1,1,2),(1,2,4),(1,3,8),(2,1,1),(2,2,5),(2,3,10),(3,1,3),(3,2
,6),(3,3,9)])
matrixB.collect()
```

In [ ]:

```
matrixA_temp = matrixA.map(lambda x : (x[1],x))
matrixB_temp = matrixB.map(lambda x : (x[0],x))
matrixB_temp.collect()
```

In [ ]:

```
#sql join
matrixAB = matrixA_temp.join(matrixB_temp)
#sql group by
matrixAB_to_group = matrixAB.map(lambda x : (x[1][0],x[1][1]))
matrixAB_to_group = matrixAB_to_group.map(lambda x :((x[0][0],x[1][1]),(x[0][2],x[1][2
])))
matrixAB_to_group.collect()
```

In [ ]:

```
matrixAB_grouped = matrixAB_to_group.groupByKey().map(lambda x : (x[0],list(x[1])))
matrixAB_product = matrixAB_grouped.map(lambda x : (x[0], product_sum(x[1])))
matrixAB_product.collect()
```

In [ ]:

```
matrixAB_product.count()
```

GENERATE MATRICES OF RANDOM VALUES

In [ ]:

```
import random
def random_matrix(n,m) :
    liste_random=[]
    for i in range(1,n+1):
        for j in range(1,m+1):
            liste_random.append((i,j,random.random()))
    return liste_random
```

In [ ]:

```
matrixA = sc.parallelize(random_matrix(1000000,3))
matrixA.count()
```

In [ ]:

```
matrixB = sc.parallelize(random_matrix(3,3))
matrixB.count()
```

In [ ]:

```
import time
start_time = time.time()
#initial algorithm
matrixA_temp = matrixA.map(lambda x : (x[1],x))
matrixB_temp = matrixB.map(lambda x : (x[0],x))
#sql join
matrixAB = matrixA_temp.join(matrixB_temp)
#sql group by
matrixAB_to_group = matrixAB.map(lambda x : (x[1][0],x[1][1]))
matrixAB_to_group = matrixAB_to_group.map(lambda x :((x[0][0],x[1][1]),(x[0][2],x[1][2])))
matrixAB_grouped = matrixAB_to_group.groupByKey().map(lambda x : (x[0],list(x[1])))
matrixAB_product = matrixAB_grouped.map(lambda x : (x[0], product_sum(x[1])))
matrixAB_product.collect()
print("--- %s seconds ---" % (time.time() - start_time))
```

In [ ]:

```
matrixAB_product.count()
```

BETTER PERFORMANCE ALGORITHM USING COGROUP INSTEAD OF JOIN

In [ ]:

```
start_time = time.time()
#better performance algorithm : replace inner join with a cogroup + broadcast the small
er matrix
def J(x):
    j=[]
    k=x[0]
    if x[1][0]!=[] and x[1][1]!=[]:
        for l in x[1][0]:
            for r in x[1][1]:
                j.append((k,(l,r)))
    return j

#matrixA = sc.parallelize(random_matrix(100000,3))
#matrixA.count()
#matrixB = sc.broadcast(random_matrix(3,3)) #matrixB.value to access values of the broa
dcast variable
#start matrix multiplication
#matrices preparation
matrixA_temp = matrixA.map(lambda x : (x[1],x))
matrixB_temp = matrixB.map(lambda x : (x[0],x))
#sql join
cg = matrixA_temp.cogroup(matrixB_temp).map(lambda x :(x[0], ( list(x[1][0]), list(x[1]
[1]))))
matrixAB = cg.flatMap(lambda x: J(x))
#sql group by
matrixAB_to_group = matrixAB.map(lambda x : (x[1][0],x[1][1]))
matrixAB_to_group = matrixAB_to_group.map(lambda x :((x[0][0],x[1][1]),(x[0][2],x[1][2
])))
matrixAB_grouped = matrixAB_to_group.groupByKey().map(lambda x : (x[0],list(x[1])))
#matrix product computation
matrixAB_product = matrixAB_grouped.map(lambda x : (x[0], product_sum(x[1])))
matrixAB_product.collect()
print("--- %s seconds ---" % (time.time() - start_time))
```