Group **A**

Please fill in your name and registration number (Matrikelnr.) **immediately**.

Exam 2 ON	**SOLUTION KEY**	29.05.2019
	Advanced Database Systems (**184.780**)	**GROUP A**
Matrikelnr.	Last Name	First Name

Duration: 75 minutes. Provide the solutions on the designated pages. **Good Luck!**

Question 1: (4)

For each of the statements below, decide if it is true or false and tick the corresponding circle. You get +1 credit for each correct answer, −1 credit for each wrong answer and 0 credit if you leave the answer open. In total, you always get ≥ 0 credits on the entire exam question 1.

1. The optimal block size in HDFS is between 4 and 64 kilobytes. true ◯ false ⊗
2. Consider a table T in Hive for which neither partitioning nor bucketing is defined, and suppose that T is stored in HDFS. Then T is represented in HDFS as a directory with one or more files. true ⊗ false ◯
3. Consider a one-round MapReduce program without combiners for multiplying two $n \times n$ matrices with n^2 reducers, such that each reducer computes one entry of the resulting matrix. Then the program can be written in such a way that each reducer receives at most $2n$ values. true ⊗ false ◯
4. In Spark, several wide transformations can be grouped into one stage. true ◯ false ⊗

Question 2: (4)

You are faced with the problem of implementing a join of two relations R and S in MapReduce in such a way that the communication cost is as low as possible. By surfing on the internet you have found an interesting web site, where it is stated that the semi-broadcast join is unbeatable. Please comment on this statement as follows:

a) Explain the required MapReduce rounds of the semi-broadcast join and the communication cost of each round. [3 credits]

see slides 46/7 of the slide set II.1: MapReduce

b) Choose *one* alternative join method and explain how it works and what its communication cost is. [1 credit]

choose 1 out of the following 3:

- Repartition Join, slide 44
- Broadcast Join, slide 45
- Directed Join, slide 48: don't forget to mention that this method is only applicabe if R and S are partitioned in the same way.

Question 3: (4)

In the form of relational algebra implemented in SQL, relations are not sets, but bags; that is, tuples are allowed to appear more than once. Consequently, there are extended definitions of union, intersection, and difference for bags. Below we define the bag intersection. Sketch out a MapReduce algorithm for computing the following operation on bags R and S:

- **Bag Intersection**, defined to be the bag of tuples in which tuple t appears the minimum of the numbers of times it appears in R *and* S. Example: if t occurs 4 times in R and 2 times in S, then t should occur 2 times in the bag intersection of R and S.

In addition to this, also identify the communication cost of your algorithm, as a function of the input size.

Your Answer:

Mapper

For each input t from R produce a tuple $(t, \text{'R'})$ and for each input t from S produce a tuple $(t, \text{'S'})$

Reducer

Count the occurrences of 'R', called r, and occurences of 'S', called s, in the value list of t. Then emit the tuple t exactly $min(s, r)$ many times.

Communication cost

The mapper emits for every tuple from R and every tuple from S one output record, therefore the communication cost is essentially $R + S$.

Question 4: (4)

a) Evaluation of Spark/Scala [2 credits]
For this part of the question, you are given some Spark/Scala code and are supposed to evaluate the results. Don't worry about the "syntax" of your output, it is only important that it is clear that you understood the underlying semantics of Spark/Scala.

```
val fibs8 = List(0, 1, 1, 2, 3, 5, 8, 13)
val result = fibs8.filter(x => (x-1) % 3 == 0 && (x+2) % 2 == 1)
```

```
result: List(1, 1, 13)
```

Note: '%' is the *modulo* operator (i.e., the rest after division).

```
val primes4 = List(2, 3, 5, 7)
def foo(num:Int) = num > 1 && ((2 to num-1 toList) forall (x => num % x != 0))
val result = primes4.flatMap(x => x :: (x to x+1 toList).filter(x => !foo(x)))
```

```
result: List(2, 3, 4, 5, 6, 7, 8)
```

Note: '`x :: xs`' prepends an element `x` to a list `xs`, thus producing a new list and '`(x to y toList)`' generates a list ranging from x to y (inclusively).

b) Writing Spark/Scala functions [2 credits]
You are given a list of random words, for instance:
`val words = List("automaton", "language", "logic", "closure")`
Produce a single list with every word duplicated (i.e. appearing twice), unless that word is shorter than 6 characters.

```
val duplicatedWords = words.flatMap( x => if (x.length < 6) {List(x)} else {List(x,x)})
duplicatedWords:  List(automaton, automaton, language, language, logic, closure, closure)
```

Note: You can use the "length" method to produce the length of any string. *You are expected to write a programmatic solution, writing down a static list of strings will get 0 credits.*

Question 5: (4)

You are given a database for a smart home lighting system with the following relational schema.
Note: underlines represent primary keys, *italics* represent foreign keys. We don't state the type of attributes here, just assume some reasonable defaults (string, int, float, ...)

manufacturer	(name, ceo, main_product)
hub	(id,name, model, version, made_by: *manufacturer.name*)
room	(id,name, created_on, lies_in: *hub.id*)
bulb	(id,name, model, version, made_by: *manufacturer.name*, part_of: *room.id*)
compat	(bulb: *bulb.id*, hub: *hub.id*, degree)
routines	(id, room: *room.id*, created_on, duration)

Assume the dataset is loaded and the corresponding views have already been created in Spark SQL. For the dataframes, simply use the name of each table appended with "DF" (e.g.: hubDF, bulbDF, etc.). You don't need to generate these dataframes, just assume they are already loaded and usable.

You are now given a number of queries, either in Spark SQL or in the Dataframe API. Your task will be to state an equivalent query of the other type, i.e. give a Spark SQL equivalent if the Dataframe API version is given, or vice-versa.

```
val query1SQL = spark.sql("SELECT   m.name, COUNT(b.id)
                           FROM     manufacturer m, bulb b
                           WHERE    b.made_by = m.name
                           GROUP BY m.name")
```

```
val query1DF = manufacturerDF.join(bulbDF,manufacturerDF("name") === bulbDF("made_by")).groupBy(manufacturerDF("name")).agg(count(bulbDF("id")))
```

```
val query2DF = routinesDF.join(roomDF,roomDF("id") === routinesDF("room"))
                         .groupBy(roomDF("id"))
                         .agg(max("duration").as("maxDuration"))
                         .select(roomDF("id"), $"maxDuration")
```

```
val query2SQL = spark.sql("SELECT r.id, max(duration) as maxDuration FROM room r, routines r2 WHERE r.id = r2.room GROUP BY r.id")
```

```
val query3DF = roomDF.join(hubDF, roomDF("lies_in") === hubDF("id"))
                     .join(routinesDF, roomDF("id") === routinesDF("room"))
                     .groupBy( roomDF("name"), hubDF("name"))
                     .agg(count("room").as("numRoutines"))
                     .select(hubDF("name").as("Hub"),roomDF("name").as("Room"), $"numRoutines")
                     .orderBy(desc("numRoutines"))
```

```
val query3SQL = spark.sql("SELECT h.name as Hub,r.name as Room, count(r2.id) as
numRoutines FROM hub h, room r, routines r2 WHERE r.lies_in = h.id AND r2.room =
r.id GROUP BY r.name, h.name ORDER BY numRoutines DESC")
```

```
val query4SQL = spark.sql("SELECT    b.name,b.model, AVG(c.degree)
                           FROM      bulb b, room r, hub h, compat c
                           WHERE     b.part_of = r.id AND r.lies_in = h.id AND c.bulb = b.id
                                     AND c.hub = h.id AND c.degree > '0.5'
                                     AND r.created_on > '2019-05-20'
                                     AND b.made_by != h.made_by
                           GROUP BY  b.name, b.model")
```

```
val query4DF = bulbDF.join(roomDF,roomDF("id") === bulbDF("part_of")).join(hubDF,
roomDF("lies_in") === hubDF("id")).join(compatDF,compatDF("bulb") === bulbDF("id")
&& compatDF("hub") === hubDF("id") ).filter("created_on > '2019-05-20'").filter("
degree > '0.5' ").filter(hubDF("made_by") !== bulbDF("made_by") ).groupBy(bulbDF(
"name"),bulbDF("model")).agg(avg(compatDF("degree")))
```

Overall: 20 points