Practice exams

Which of the following is the deepest level in Spark’s execution hierarchy?

Ans: Task.

Explanation:

The hierarchy is, from top to bottom: Job, Stage, Task.

Executors and slots facilitate the execution of tasks, but they are not directly part of the hierarchy. Executors are launched by the driver on worker nodes for the purpose of running a specific Spark

application. Slots help Spark parallelize work. An executor can have multiple slots which enable it to process multiple tasks in parallel.

- Here is a great tip on how to differentiate actions from transformations: If an operation returns a DataFrame, Dataset, or an RDD, it is a transformation. Otherwise, it is an action. Actions do not send results to the driver, while transformations do.

Q1: Which of the following describes a narrow transformation?

Ans: **A narrow transformation is an operation in which no data is exchanged across the cluster.**

Q2: Which of the following statements about stages is correct?

Ans: **Tasks in a stage may be executed by multiple machines at the same time.**

Q3: Which of the following describes tasks?

Ans: **Tasks get assigned to the executors by the driver.**

Q4: Which of the following describes a difference between Spark's cluster and client execution modes?

Ans: **In cluster mode, the driver resides on a worker node, while it resides on an edge node in client mode.**

Q4: Which of the following describes Spark's standalone deployment mode?

Ans: **Standalone mode uses only a single executor per worker per application.f**

Q5: Which of the following describes properties of a shuffle?

Ans: **In a shuffle, Spark writes data to disk.**

Q6: Which of the following statements about the differences between actions and transformations is correct?

Ans: **Actions can trigger Adaptive Query Execution, while transformation cannot.**

Q7: Which of the following is a characteristic of the cluster manager?

Ans: **The cluster manager receives input from the driver through the `SparkContext`.**

Q8: Which of the following are valid execution modes?

Ans: **Client, Cluster, Local**

Q9: Deployment methods in spark

Ans: Deployment modes often refer to ways that Spark can be deployed in cluster mode and how it uses specific frameworks outside Spark. Valid deployment modes are standalone, Apache YARN, Apache Mesos and Kubernetes.

Q10: Which of the following describes Spark actions?

Ans: **The driver receives data upon request by actions.**

Q11: Which of the following statements about executors is correct?

Ans: **Executors stop upon application completion by default.**

Q12: Which of the following describes a valid concern about partitioning?

Ans: **A shuffle operation returns 200 partitions if not explicitly set.**

Q13: Which of the following is characteristic of using accumulators?

Ans: **Accumulator values can only be read by the driver, but not by executors.**

Q14: Which of the following statements about reducing out-of-memory errors is incorrect?

Ans: **Concatenating multiple string columns into a single column may guard against out-of-memory errors**

Q15: Which of the following statements about storage levels is incorrect?

Ans: `MEMORY_AND_DISK` **replicates cached DataFrames both on memory and disk.**

Q16: Which of the following is not a feature of Adaptive Query Execution?

Ans: **Reroute a query in case of an executor failure.**

Q17: The code block displayed below contains an error. The code block should trigger Spark to cache DataFrame `transactionsDf` in executor memory where available, writing to disk where insufficient executor memory is available, in a fault-tolerant way. Find the error.

Code block:

```
transactionsDf.persist(StorageLevel.MEMORY_AND_DISK)
```

Ans: **The storage level is inappropriate for fault-tolerant storage.**

Q18: The code block displayed below contains an error. The code block should configure Spark to split data in 20 parts when exchanging data between executors for joins or aggregations. Find the error.

Code block:

```
spark.conf.set(spark.sql.shuffle.partitions, 20)
```

Ans: Correct code block:

```
spark.conf.set("spark.sql.shuffle.partitions", 20)
```

The code block expresses the option incorrectly.

Correct! The option should be expressed as a string.

Q19: **Column `predError` should be sorted by `desc_nulls_first()` instead.**

Wrong. Since Spark's default sort order matches `asc_nulls_first()`, `null`s would have to come last when inverted.

Q20: While you might be tempted to change `unix_timestamp()` to `to_unixtime()` (in line with the `from_unixtime()` operator), this function does not exist in Spark. `unix_timestamp()` is the correct operator to use here.

Q21: `itemsDf.withColumnRenamed(col("attributes"), col("feature0"), col("supplier"), col("feature1"))`

Wrong. The `DataFrame.withColumnRenamed()` operator takes exactly two string arguments. So, in this answer both using `col()` and using four arguments is wrong.

Q22: `itemsDf.withColumnRenamed("attributes", "feature0").withColumnRenamed("supplier", "feature1")`

Correct! Spark's `DataFrame.withColumnRenamed` syntax makes it relatively easy to change the name of a column

Q23: `1. itemsDf.withColumnRenamed("attributes", "feature0")`

`2. itemsDf.withColumnRenamed("supplier", "feature1")`

No. In this answer, the returned DataFrame will only have column `supplier` be renamed, since the result of the first line is not written back to `itemsDf`.

Q24: Find the error. Code block: `transactionsDf.write.partitionOn("storeId").parquet(filePath)`

Ans: **No method `partitionOn()` exists for the `DataFrame` class, `partitionBy()` should be used instead.**

Q25: Which of the following code blocks reads in the two-partition parquet file stored at `filePath`, making sure all columns are included exactly once even though each partition has a different schema?

Ans: **`spark.read.option("mergeSchema", "true").parquet(filePath)`**

Correct. Spark's `DataFrameReader`'s `mergeSchema` option will work well here, since columns that appear in both partitions have matching data types. Note that `mergeSchema` would fail if one or more columns with the same name that appear in both partitions would have different data types.

Q26: On a side note: One answer option includes a function `str_split`. This function does not exist in pySpark. Only `split` exists.

—- — — — — — Line Break— — — — — — Test 1— — — — — —

Q1: Which of the following options describes the responsibility of the executors in Spark?

Ans: **The executors accept tasks from the driver, execute those tasks, and return results to the driver.**

Q2: Which of the following describes the role of tasks in the Spark execution hierarchy?

Ans: Tasks are the smallest element in the execution hierarchy

Q3: Which of the following describes the role of the cluster manager?

Ans: **The cluster manager allocates resources to Spark applications and maintains the executor processes in client mode.**

Q4: Which of the following is the idea behind dynamic partition pruning in Spark?

Ans: **Dynamic partition pruning is intended to skip over the data you do not need in the results of a query.**

Q5: Which of the following is one of the big performance advantages that Spark has over Hadoop?

Ans: Spark achieves great performance by storing data and performing computation in memory, whereas large jobs in Hadoop requrie a large amount of relatively slow disk I/O operations.

Q6: Which of the following is the deepest level in Spark's execution hierarchy?

Ans: Task. The hierarchy is, from top to bottom: Job, Stage, Task.

Q7: Which of the following statements about garbage collection in Spark is incorrect?

Ans: **Manually persisting RDDs in Spark prevents them from being garbage collected.**

Q8: Which of the following describes characteristics of the Dataset API?

Ans: **The Dataset API is available in Scala, but it is not available in Python.**

Q9: Which of the following describes the difference between client and cluster execution modes?

Ans: In cluster mode, the driver runs on the worker nodes, while the client mode runs the driver on the client machine.

Q10: Which of the following statements about executors is correct, assuming that one can consider each of the JVMs working as executors as a pool of task execution slots?

Ans: **Tasks run in parallel via slots.**

Q11: Which of the following statements about RDDs is incorrect?

Ans: **An RDD consists of a single partition.**

Quite the opposite: Spark partitions RDDs and distributes the partitions across multiple nodes.

Q12: Which of the following describes characteristics of the Spark UI?

Ans: **There is a place in the Spark UI that shows the property `spark.executor.memory`.**

Q13: Which of the following statements about broadcast variables is correct?

Ans: Broadcast variables are immutable

Q14: Which of the following is a viable way to improve Spark's performance when dealing with large amounts of data, given that there is only a single application running on the cluster?

Ans: In**crease values for the properties** `spark.default.parallelism` **and** `spark.sql.shuffle.partitions`

Q15: Which of the following describes a shuffle?

Ans: **A shuffle is a process that compares data between partitions.**

Q17: Which of the following describes Spark's Adaptive Query Execution?

Ans: **Adaptive Query Execution features are dynamically switching join strategies and dynamically optimizing skew joins.**

Q18: Which of the following code blocks removes all rows in the 6-column DataFrame `transactionsDf`
that have missing data in at least 3 columns?

Ans: **transactionsDf.dropna(thresh=4)**

Q19: `transactionsDf.filter(col("storeId")==25).take(5)`

Any of the options with `collect` will not work because `collect` does not take any arguments, and in both cases the argument `5` is given.

Ans: `transactionsDf.filter(col("storeId")==25).collect(5)`

Q20: `itemsDf.withColumnRenamed(col("manufacturer"), col("supplier"))`

No. Watch out – although the `col()` method works for many methods of the DataFrame API, `withColumnRenamed` is not one of them. As outlined in the documentation linked below, `withColumnRenamed` expects strings.

Q21: `transactionsDf.desc_nulls_last("predError")`

Wrong, this is invalid syntax. There is no method `DataFrame.desc_nulls_last()` in the Spark API. There is a Spark function `desc_nulls_last()` however.

`transactionsDf.orderBy("predError").asc_nulls_last()`

Incorrect. There is no method `DataFrame.asc_nulls_last()` in the Spark API (see above).

Q22: Correct code block:

`transactionsDf.drop("predError", "productId", "value")`

The `select` operator should be replaced by the `drop` operator and the arguments to the `drop` operator should be column names `predError`, `productId` and `value` as strings.

Correct! It is important to know that the `drop` operator expects column names to be passed as string arguments if multiple columns should be removed.

Q23: The code block displayed below contains an error. When the code block below has executed, it should have divided DataFrame `transactionsDf` into 14 parts, based on columns `storeId` and `transactionDate` (in this order). Find the error.

Code block:

```
transactionsDf.coalesce(14, ("storeId", "transactionDate"))
```

Ans: Correct code block:

```
transactionsDf.repartition(14, "storeId", "transactionDate").count()
```

In the Spark documentation, the call structure for `repartition` is shown like this: `DataFrame.repartition(numPartitions, *cols)`. The `*` operator means that any argument after `numPartitions` will be interpreted as column. Therefore, the brackets need to be removed.

Finally, the question specifies that after the execution the DataFrame should be divided. So, indirectly this question is asking us to append an action to the code block. Since `.select()` is a transformation. the only possible choice here is `.count()`.

Q24: Correct code block: `transactionsDf.select((col("storeId").between(20, 30)) & (col("productId")==2))`

Ans: Another riddle here is how to chain the two conditions. The only valid answer here is `&`. Operators like `&&` or `and` are not valid. Other boolean operators that would be valid in Spark are `|` and `~`.

Q25: Which of the following code blocks returns only rows from DataFrame `transactionsDf` in which values in column `productId` are unique?

Ans: transactionsDf.dropDuplicates(subset=[”productId”] In the documentation for `dropDuplicates`, the examples show that `subset` should be used with a list.

Q26: **The code block displayed below contains an error. The code block should return a DataFrame where all entries in column `supplier` contain the letter**

combination `et` in this order. Find the error.
Code block: `itemsDf.filter(Column('supplier').isin('et'))`

Ans: Correct code block: `itemsDf.filter(col('supplier').contains('et'))`

—- — — — — — Line Break— — — — — — Test 2— — — — — —

Q1: Which of the following statements about Spark's execution hierarchy is correct?

Ans: **In Spark's execution hierarchy, a job may reach over multiple stage boundaries.**

Q2: Which of the following describes slots?

Ans: A Java Virtual Machine (JVM) working as an executor can be considered as a pool of slots for task execution

Q3: Which of the following describes the conversion of a computational query into an execution plan in Spark?

Ans: **The executed physical plan depends on a cost optimization from a previous stage.**

Q4: Which of the following describes characteristics of the Spark driver?

Ans: The Spark driver’s responsibility includes scheduling queries for execution on worker nodes

Q5: **Which of the following describes executors?**
Ans: **Executors are responsible for carrying out work that they get assigned by the driver.**

Q6: Which of the following statements about DAGs is correct?

Ans: **DAGs can be decomposed into tasks that are executed in parallel.**

Q7: Which of the following statements about lazy evaluation is incorrect?

Ans: **Execution is triggered by transformations.**

Q8: **Which of the following describes how Spark achieves fault tolerance?**

Ans: **If an executor on a worker node fails while calculating an RDD, that RDD can be recomputed by another executor using the lineage.**

Q9: Which of the following describes Spark's way of managing memory?

Ans: Storage memory is used for caching partitions derived from DataFrames.

Q10: Which of the following statements about Spark's DataFrames is incorrect?

Ans: **Spark's DataFrames are equal to Python's or R's DataFrames.**

Incorrect. They are only similar. A major difference between Spark and Python is that Spark's DataFrames are distributed, whereby Python's are not.

Q11: Which of the following statements about Spark's configuration properties is incorrect?

Ans: **The default number of partitions to use when shuffling data for joins or aggregations is 300.**

No, the default value of the applicable property `spark.sql.shuffle.partitions` is 200.

Q12: Which of the following describes a way for resizing a DataFrame from 16 to 8 partitions in the most efficient way?

Ans: **Use a narrow transformation to reduce the number of partitions.**

Correct! `DataFrame.coalesce(n)` is a narrow transformation, and in fact the most efficient way to resize the DataFrame of all options listed. One would run `DataFrame.coalesce(8)` to resize the DataFrame.

Q13: Dataframe.select() is a narrow transformation

Q14: Which of the following statements about data skew is incorrect?

Ans: **To mitigate skew, Spark automatically disregards null values in keys when joining.**

Q15: Which of the following describes the characteristics of accumulators?

Ans: **If an action including an accumulator fails during execution and Spark manages to restart the action and complete it successfully, only the successful attempt will be counted in the accumulator.**

Correct, when Spark tries to rerun a failed action that includes an accumulator, it will only update the accumulator if the action succeeded.

Q16: Which of the following code blocks stores a part of the data in DataFrame `itemsDf` on executors?

Ans: itemsDf.cache().count()

Caching means storing a copy of a partition on an executor, so it can be accessed quicker by subsequent operations, instead of having to be recalculated. `cache()` is a

lazily-evaluated method of the DataFrame. Since `count()` is an action (while `filter()` is not), it triggers the caching process.

Q17: This question targets your knowledge about how to chain filtering conditions. Each filtering condition should be in parentheses. The correct operator for "or" is the pipe character (`|`) and not the word `or`. Another operator of concern is the equality operator. For the purpose of comparison, equality is expressed as two equal signs (`==`).

Q18: The code block displayed below contains an error. The code block should produce a DataFrame with `color` as the only column and three rows with `color` values of `red`, `blue`, and `green`, respectively. Find the error.

Code block: `1. spark.createDataFrame([("red",), ("blue",), ("green",)], "color")`

Ans: **The `"color"` expression needs to be wrapped in brackets, so it reads `["color"]`.(Correct)**

Correct code block:

```
spark.createDataFrame([("red",), ("blue",), ("green",)], ["color"])
```

The `createDataFrame` syntax is not exactly straightforward, but luckily the documentation (linked below) provides several examples on how to use it.

Q19: **Which of the following code blocks stores DataFrame `itemsDf` in executor memory and, if insufficient memory is available, serializes it and saves it to disk?**

- `itemsDf.persist(StorageLevel.MEMORY_ONLY)`
- `itemsDf.cache(StorageLevel.MEMORY_AND_DISK)` **(Incorrect)**
- `itemsDf.store()`
- `itemsDf.cache()` **(Correct)**
- `itemsDf.write.option('destination', 'memory').save()`

Ans: The key to solving this question is knowing (or reading in the documentation) that, by default, `cache()` stores values to memory and writes any partitions for which there is insufficient memory to disk. `persist()` can achieve the exact same behavior, however not with the `StorageLevel.MEMORY_ONLY` option listed here. It is also worth noting that `cache()` does not have any arguments.

Q20: **Which of the following code blocks creates a new one-column, two-row DataFrame `dfDates` with column `date` of type `timestamp`?**

Ans:

```
    1. dfDates = spark.createDataFrame([("23/01/2022 11:28:12",),("24/01/2022 10:58:34",)], ["date"])
```

```
2. dfDates = dfDates.withColumn("date", to_timestamp("date", "dd/MM/yyyy HH:mm:ss"))
```
(Correct)

When no schema is specified, Spark sets the `string` data type as default. So, the next line is needed.

Q21: The code block displayed below contains an error. The code block should configure Spark so that DataFrames up to a size of 20 MB will be broadcast to all worker nodes when performing a join. Find the error.

Code block:

```
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 20)
```

Ans: **Spark will only broadcast DataFrames that are much smaller than the default value.**

This is correct. The default value is 10 MB (10485760 bytes). Since the configuration for `spark.sql.autoBroadcastJoinThreshold` expects a number in bytes (and not megabytes), the code block sets the limits to merely 20 bytes, instead of the requested 20 * 1024 * 1024 (= 20971520) bytes.

Q22: **Which of the following code blocks returns a DataFrame that is an inner join of DataFrame `itemsDf` and DataFrame `transactionsDf`, on columns `itemId` and `productId`, respectively and in which every `itemId` just appears once?**

Ans: `itemsDf.join(transactionsDf, itemsDf.itemId==transactionsDf.productId).dropDuplicates(["itemId"])` **(Correct)**

Filtering out distinct rows based on columns is achieved with the `dropDuplicates` method, not the `distinct` method which does not take any arguments.

Q23: Which of the following code blocks generally causes a great amount of network traffic?

Ans: `DataFrame.collect()` sends all data in a DataFrame from executors to the driver, so this generally causes a great amount of network traffic in comparison to the other options listed.