PySpark Interview Handbook — Batch 3

Problems 051–075
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 051: 051 - Window Functions: Dense_rank challenge

Problem

Window Functions

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
w = Window.partitionBy("user_id").orderBy("ts")
res = logs.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .wit
```

Problem 052: 052 - Aggregations & GroupBy: Agg challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

```
from pyspark.sql import functions as F
user_stats = users.groupBy("user_id").agg(F.count("*").alias("cnt"), F.sum("value").alias("sum_value"), F.avg("value").
global_distinct = users.select(F.countDistinct("user_id").alias("distinct_users"))
res = user_stats
```

Problem 053: 053 - Aggregations & GroupBy: Agg challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

```
from pyspark.sql import functions as F
user_stats = sessions.groupBy("user_id").agg(F.count("*").alias("cnt"), F.sum("value").alias("sum_value"), F.avg("value
global_distinct = sessions.select(F.countDistinct("user_id").alias("distinct_users"))
res = user_stats
```

Problem 054: 054 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

```
res = sessions.hint("broadcast")
```

Problem 055: 055 - Performance & Tuning: Repartition challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = clicks.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 056: 056 - MLlib Basics: Pipeline challenge

Problem

MLlib Basics

Solution (PySpark)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
label_indexer = StringIndexer(inputCol="event_type", outputCol="label", handleInvalid="skip")
assembler = VectorAssembler(inputCols=["value"], outputCol="features")
lr = LogisticRegression(maxIter=10)
model = Pipeline(stages=[label_indexer, assembler, lr]).fit(logs)
res = model.transform(logs)
```

Problem 057: 057 - Performance & Tuning: Checkpoint challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = clicks.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 058: 058 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

```
# streaming example would use readStream; here batch placeholder
res = events
```

Problem 059: 059 - Joins: Right challenge

Problem

Joins

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
users = spark.createDataFrame([("u1","US"),("u2","IN")], ["user_id","country"])
res = sessions.join(F.broadcast(users), "user_id", "left")
```

Problem 060: 060 - File IO & Formats: Delta-like challenge

Problem

File IO & Formats

Solution (PySpark)

```
# Example write (commented):
# transactions.write.mode("overwrite").partitionBy("country").parquet("/path/out")
res = transactions
```

Problem 061: 061 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

```
# streaming example would use readStream; here batch placeholder
res = events
```

Problem 062: 062 - Joins: Right challenge

Problem

Joins

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
users = spark.createDataFrame([("u1","US"),("u2","IN")], ["user_id","country"])
res = events.join(F.broadcast(users), "user_id", "left")
```

Problem 063: 063 - Performance & Tuning: Cache challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = clicks.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 064: 064 - Performance & Tuning: Skew challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = sessions.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 065: 065 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

```
res = products.hint("broadcast")
```

Problem 066: 066 - Complex Types: Explode challenge

Problem

Complex Types

Solution (PySpark)

```
from pyspark.sql import functions as F
res = logs
if "tags" in res.columns:
    res = res.withColumn("tag", F.explode_outer("tags"))
```

Problem 067: 067 - MLlib Basics: Vectorassembler challenge

Problem

MLlib Basics

Solution (PySpark)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
label_indexer = StringIndexer(inputCol="event_type", outputCol="label", handleInvalid="skip")
assembler = VectorAssembler(inputCols=["value"], outputCol="features")
lr = LogisticRegression(maxIter=10)
model = Pipeline(stages=[label_indexer, assembler, lr]).fit(logs)
res = model.transform(logs)
```

Problem 068: 068 - Window Functions: Dense_rank challenge

Problem

Window Functions

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
w = Window.partitionBy("user_id").orderBy("ts")
res = users.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .wi
```

Problem 069: 069 - Dates & Timestamps: From_unixtime challenge

Problem

Dates & Timestamps

Solution (PySpark)

```
from pyspark.sql import functions as F
res = users.withColumn("day", F.to_date("ts")).groupBy("day").agg(F.count("*").alias("events"), F.avg("value").alias("a
```

Problem 070: 070 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

```
transactions.createOrReplaceTempView("transactions_view")
res = spark.sql("SELECT user_id, COUNT(*) AS cnt FROM transactions_view GROUP BY user_id")
```

Problem 071: 071 - Strings & Regex: Regexp_replace challenge

Problem

Strings & Regex

Solution (PySpark)

```
from pyspark.sql import functions as F
res = transactions
if "email" in res.columns:
    res = res.withColumn("domain", F.regexp_extract("email", "@(.*)$", 1))
```

Problem 072: 072 - Performance & Tuning: Coalesce challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = sessions.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 073: 073 - Aggregations & GroupBy: Agg challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

```
from pyspark.sql import functions as F
user_stats = transactions.groupBy("user_id").agg(F.count("*").alias("cnt"), F.sum("value").alias("sum_value"), F.avg("v
global_distinct = transactions.select(F.countDistinct("user_id").alias("distinct_users"))
res = user_stats
```

Problem 074: 074 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

```
events.createOrReplaceTempView("events_view")
res = spark.sql("SELECT user_id, COUNT(*) AS cnt FROM events_view GROUP BY user_id")
```

Problem 075: 075 - Joins: Anti challenge

Problem

Joins

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
users = spark.createDataFrame([("u1","US"),("u2","IN")], ["user_id","country"])
res = logs.join(F.broadcast(users), "user_id", "left")
```