PySpark Interview Handbook — Batch 2

Problems 026–050
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 026: 026 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 027: 027 - Streaming (Structured): Trigger challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 028: 028 - Window Functions: 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 = clicks.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .w
```

Problem 029: 029 - Window Functions: Row_number 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 = orders.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .w
```

Problem 030: 030 - DataFrame Basics: Select challenge

Problem

DataFrame Basics

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
res = logs.select("id", "user_id", "event_type", "value") \    .withColumn("value_norm", (F.col("value") - F.mean("valu
assert "value_norm" in res.columns
```

Problem 031: 031 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 032: 032 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 033: 033 - Pivot & Crosstab: Pivot challenge

Problem

Pivot & Crosstab

Solution (PySpark)

```
from pyspark.sql import functions as F
res = clicks.groupBy("user_id").pivot("event_type").agg(F.count("*")).fillna(0)
```

Problem 034: 034 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 035: 035 - Pivot & Crosstab: Cube challenge

Problem

Pivot & Crosstab

Solution (PySpark)

```
from pyspark.sql import functions as F
res = logs.groupBy("user_id").pivot("event_type").agg(F.count("*")).fillna(0)
```

Problem 036: 036 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 037: 037 - Dates & Timestamps: From_unixtime challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 038: 038 - Pivot & Crosstab: Rollup challenge

Problem

Pivot & Crosstab

Solution (PySpark)

```
from pyspark.sql import functions as F
res = users.groupBy("user_id").pivot("event_type").agg(F.count("*")).fillna(0)
```

Problem 039: 039 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 040: 040 - UDFs & Pandas UDFs: Returntype challenge

Problem

UDFs & Pandas UDFs

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
@F.udf(DoubleType())
def score(x): return float(x)*1.1 if x is not None else None
res = events.withColumn("score", score("value"))
```

Problem 041: 041 - Misc Utilities: Broadcast joins challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 042: 042 - 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(users)
res = model.transform(users)
```

Problem 043: 043 - Joins: Semi 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")
```

Problem 044: 044 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 045: 045 - File IO & Formats: Orc challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 046: 046 - File IO & Formats: Json challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 047: 047 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 048: 048 - MLlib Basics: Logisticregression 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(transactions)
res = model.transform(transactions)
```

Problem 049: 049 - MLlib Basics: Logisticregression 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(users)
res = model.transform(users)
```

Problem 050: 050 - UDFs & Pandas UDFs: Vectorized challenge

Problem

UDFs & Pandas UDFs

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
@F.udf(DoubleType())
def score(x): return float(x)*1.1 if x is not None else None
res = logs.withColumn("score", score("value"))
```