# PySpark Interview Handbook — Batch 7

Problems 151–175
Generated: 2025-09-13 04:05:06Z (UTC)

## Problem 151: 151 - Complex Types: Maps challenge

## Problem

Complex Types

## Solution (PySpark)

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

## Problem 152: 152 - Misc Utilities: Mappartitions challenge

### Problem

Misc Utilities

### Solution (PySpark)

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

## Problem 153: 153 - Performance & Tuning: Coalesce challenge

### Problem

Performance & Tuning

### Solution (PySpark)

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

# Problem 154: 154 - Strings & Regex: Split challenge

## Problem

Strings & Regex

## Solution (PySpark)

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

## Problem 155: 155 - Pivot & Crosstab: Pivot challenge

## Problem

Pivot & Crosstab

## Solution (PySpark)

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

## Problem 156: 156 - File IO & Formats: Delta-like challenge

### Problem

File IO & Formats

### Solution (PySpark)

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

## Problem 157: 157 - File IO & Formats: Orc challenge

### Problem

File IO & Formats

### Solution (PySpark)

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

## Problem 158: 158 - 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 = clicks.join(F.broadcast(users), "user_id", "left")
```

## Problem 159: 159 - Complex Types: Maps challenge

### Problem

Complex Types

### Solution (PySpark)

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

## Problem 160: 160 - 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 161: 161 - File IO & Formats: Orc challenge

### Problem

File IO & Formats

### Solution (PySpark)

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

## Problem 162: 162 - Dates & Timestamps: Date_trunc challenge

### Problem

Dates & Timestamps

### Solution (PySpark)

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

## Problem 163: 163 - Complex Types: Arrays challenge

### Problem

Complex Types

### Solution (PySpark)

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

## Problem 164: 164 - 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(orders)
res = model.transform(orders)
```

## Problem 165: 165 - DataFrame Basics: Select challenge

### Problem

DataFrame Basics

### Solution (PySpark)

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

## Problem 166: 166 - Spark SQL: Create temp view challenge

### Problem

Spark SQL

### Solution (PySpark)

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

# Problem 167: 167 - Aggregations & GroupBy: Agg challenge

## Problem

Aggregations & GroupBy

## Solution (PySpark)

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

## Problem 168: 168 - Spark SQL: Sql queries challenge

### Problem

Spark SQL

### Solution (PySpark)

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

## Problem 169: 169 - File IO & Formats: Parquet challenge

### Problem

File IO & Formats

### Solution (PySpark)

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

## Problem 170: 170 - Complex Types: Maps challenge

### Problem

Complex Types

### Solution (PySpark)

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

# Problem 171: 171 - Dates & Timestamps: From_unixtime challenge

## Problem

Dates & Timestamps

## Solution (PySpark)

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

# Problem 172: 172 - Misc Utilities: Accumulators (concept) challenge

## Problem

Misc Utilities

## Solution (PySpark)

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

## Problem 173: 173 - Streaming (Structured): Watermark challenge

### Problem

Streaming (Structured)

### Solution (PySpark)

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

## Problem 174: 174 - DataFrame Basics: Drop challenge

### Problem

DataFrame Basics

### Solution (PySpark)

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

## Problem 175: 175 - Strings & Regex: Concat_ws challenge

## Problem

Strings & Regex

## Solution (PySpark)

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