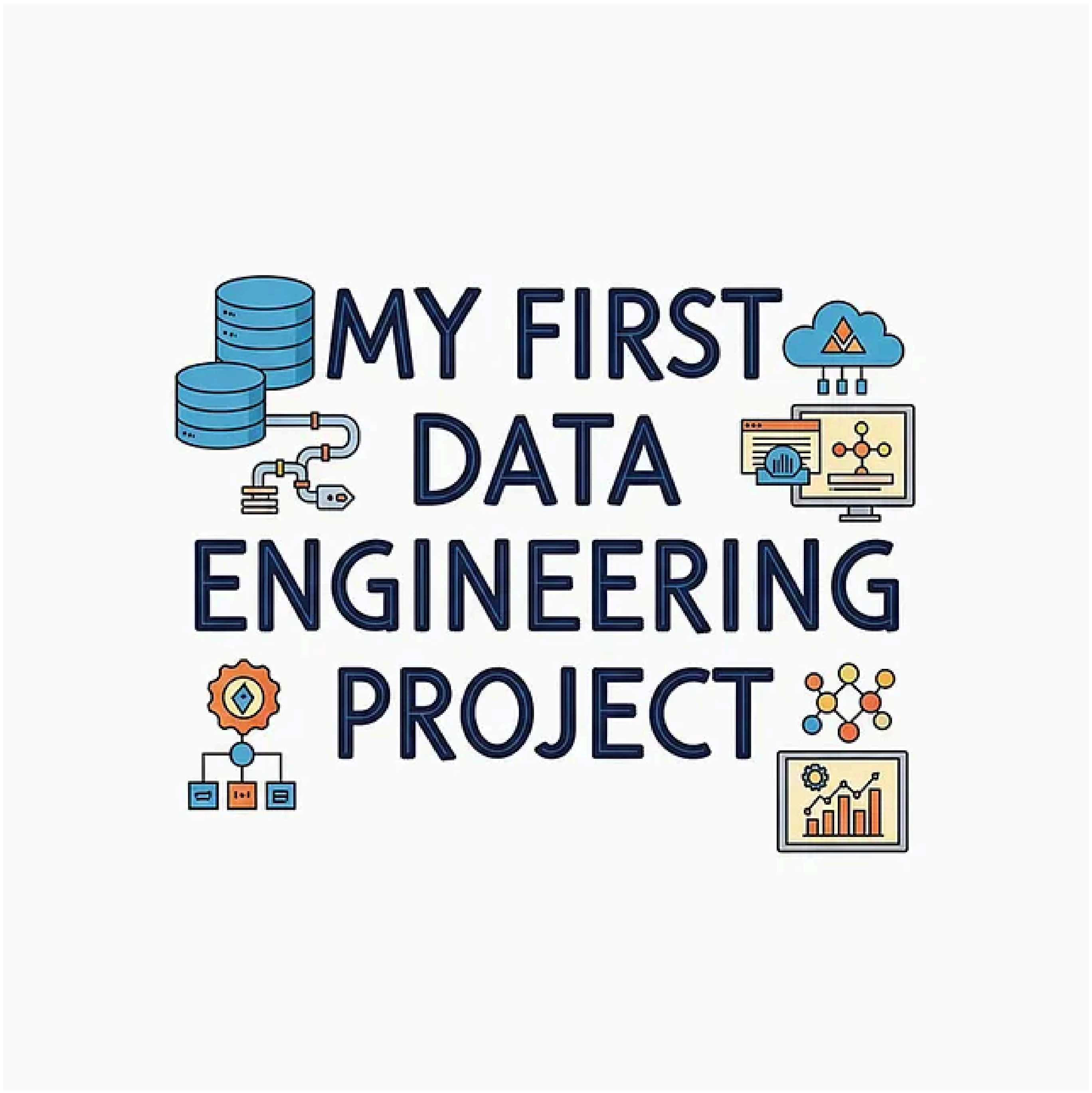
Open in app ↗

Medium Search Write 3

Get unlimited access to the best of Medium for less than $1/week. Become a member



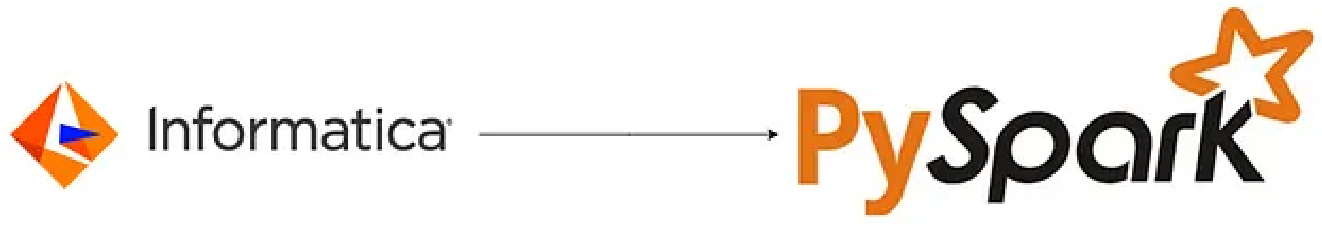
# My First Data Engineering Project: Phase 3— Migration of Informatica and Oracle Stored Procedures to PySpark

Anurag Sharma 6 min read · Jun 12, 2025

The third chapter of my data engineering journey! The third phase of a transformative project for a major US-based multi-investment company. After migrating a 300 TB Oracle data warehouse to Cloudera's HDFS (Part 1) and converting 112 DataStage ETL jobs to PySpark (Part 2), we faced the most challenging phase: migrating Informatica workflows and Oracle stored procedures to PySpark. This phase involved transforming complex report generation processes and intricate business logic, all while processing data from HDFS to HDFS.

Phase 1: https://medium.com/@one.step.analytics.on.data/my-first-data-engineering-project-phase-1-migrating-a-massive-data-warehouse-from-oracle-to-02c2b6391ddc

Phase 2: https://medium.com/@one.step.analytics.on.data/my-first-data-engineering-project-phase-2-migrating-datastage-etl-jobs-to-pyspark-161a8b4e5f18



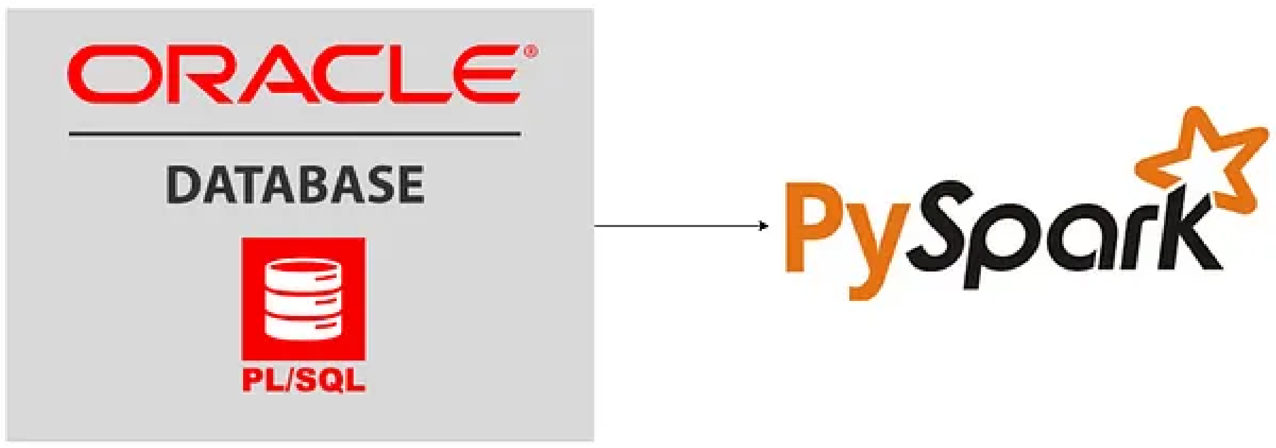
**The Project Context:**
Our client's data ecosystem powered a financial powerhouse, featuring an operational Oracle database (OLTP) for transactional applications and a data warehouse for business intelligence (BI) reporting via OBIEE.

The third phase targeted two critical components:
**1. Informatica Workflows:** These handled daily, weekly, monthly, quarterly, semi-annual, and annual report generation, joining dimension tables with fact tables to produce aggregated metrics for BI dashboards and customer reports.
**2. Oracle Stored Procedures:** These PL/SQL procedures joined dimension tables to create static, pre-aggregated tables, which Informatica later combined with fact tables for reporting.
The goal was to migrate both to PySpark, processing data entirely within HDFS. This shift from Oracle's relational environment to Hadoop's distributed framework introduced significant challenges, as we needed to replicate complex transformations and ensure the accuracy of business-critical reports.

**My Roles and Responsibilities:**

• **Workflow and Procedure Analysis:** Analyzed Informatica workflows and Oracle stored procedures to map their logic and dependencies.
• **PySpark Development:** Designed PySpark scripts to replicate Informatica transformations and PL/SQL logic, handling HDFS-to-HDFS processing.

• **UDF and Complex Logic Implementation:** Developed custom User-Defined Functions (UDFs) to handle intricate business rules not natively supported by Spark.
• **Performance Optimization:** Tuned PySpark jobs for scalability and efficiency on the Hadoop cluster.
• **Validation and Testing:** Led validation efforts to ensure PySpark outputs matched legacy results, maintaining report accuracy.
• **Documentation and Collaboration:** Documented migration processes and collaborated with BI analysts to validate business logic, leveraging my prior experience with metadata-driven solution.



**The Challenge: Replicating Informatica and Stored Procedures in PySpark**
This phase was the toughest due to the complexity of Informatica workflows and PL/SQL procedures. Informatica handled report generation by joining dimension and fact tables, applying filters, aggregations, and transformations to produce time-based reports (e.g., monthly portfolio performance). Oracle stored procedures pre-process dimension tables, creating static tables with business logic like hierarchical roll-ups or time-based aggregations.

Replicating these in PySpark required:

• **HDFS-to-HDFS Processing:** Unlike the DataStage migration (Oracle-to-Oracle), this phase involved reading and writing within HDFS, leveraging Hive tables created in Part 1.

• **Complex Logic Translation:** PL/SQL's procedural logic and Informatica's visual mappings didn't align easily with Spark's DataFrame API.

• **Report Accuracy:** Any discrepancy in report outputs could erode trust in the new system, given its visibility to customers and stakeholders.

**Step-by-Step Migration Process:**

**1. Analysis and Mapping:**

◦ **Informatica Workflows:** I analyzed Informatica PowerCenter mappings, identifying sources (Hive dimension and fact tables), transformations (joins, aggregations, lookups), and targets (report tables, CSV files). I documented dependencies to preserve execution order.

◦ **Stored Procedures:** I reviewed PL/SQL code, focusing on logic like nested loops, conditional statements, and hierarchical aggregations (e.g., rolling up investment categories by region). I extracted input/output schemas and business rules.

**2. PySpark Script Development:**

◦ **Informatica to PySpark:**

▪ Translated Informatica transformations into PySpark DataFrame operations. For example:

▪ **Joins:** Informatica's joiner transformations became df.join() with conditions.
▪ **Aggregations:** Group-by operations mapped to df.groupBy().agg().
▪ **Lookups:** Replaced Informatica lookups with Spark joins or broadcast joins for small dimension tables.
▪ Handled time-based report logic (e.g., monthly aggregations) using window functions (pyspark.sql.functions.window).
◦ **Stored Procedures to PySpark:**
▪ Converted PL/SQL's procedural logic into PySpark's functional approach. For instance, a PL/SQL loop aggregating investments by category was rewritten as:

```
from pyspark.sql.functions import sum, col
 df = df.groupBy("investment_category").agg(sum("amount").alias("total_investmen
```

▪ For complex logic (e.g., recursive calculations for hierarchical roll-ups), I used iterative DataFrame operations or temporary views to mimic PL/SQL's step-by-step processing.

**3. Custom UDFs for Business Logic:**
◦ Some transformations required custom logic not supported by Spark's native functions. Examples included:
▪ **Portfolio Performance Scoring:** Calculating a weighted score based on investment returns, risk levels, and market conditions.

```
from pyspark.sql.functions import udf
   from pyspark.sql.types import DoubleType
```

```
    def calculate_score(return_val, risk_level, market_factor):
    # Simplified scoring logic
    score = (return_val * 0.6) + (1 / risk_level * 0.3) + (market_factor * 0.1)
    return max(0.0, min(score, 100.0))
    score_udf = udf(calculate_score, DoubleType())
    df = df.withColumn("portfolio_score", score_udf(df["return"], df["risk"], df[
```

▪ Time-Based Roll-Ups: Aggregating semi-annual data with custom date logic (e.g., fiscal vs. calendar periods).

```
def fiscal_period(date_str, fiscal_start_month):
    from datetime import datetime
    date = datetime.strptime(date_str, "%Y-%m-%d")
    month = date.month
    year = date.year if month >= fiscal_start_month else date.year - 1
    return f"{year}-Q{(month - fiscal_start_month) // 3 + 1}"

    fiscal_udf = udf(fiscal_period, StringType())
    df = df.withColumn("fiscal_quarter", fiscal_udf(df["date"], lit(4)))
```

**4. Performance Optimization:**
◦ Optimized PySpark jobs for HDFS processing:
▪ Used partitioning (e.g., by date or region) to reduce shuffle operations.
▪ Applied broadcast joins for small dimension tables to minimize data movement.
▪ Cached intermediate results for complex, multi-step transformations.
◦ Tuned Spark configurations (e.g., spark.executor.cores, spark.sql.shuffle.partitions) to balance cluster resources.
◦ Ensured efficient Hive table access by leveraging Parquet's columnar storage and predicate pushdown.

**5. Testing and Validation:**
◦ Ran PySpark jobs in a staging environment, comparing outputs (report tables, CSVs) against Informatica and PL/SQL results.
◦ Validated key metrics (e.g., quarterly portfolio returns, aggregated investment totals) using automated Python scripts.
◦ Conducted end-to-end testing with BI analysts, ensuring reports matched legacy outputs in format and content.
◦ Used checksums and row count comparisons for static tables, ensuring data integrity.

**6. Deployment and Scheduling:**
◦ Deployed PySpark scripts to the production Hadoop cluster, integrating with Control-M (used in Parts 1 and 2) for scheduling.
◦ Configured workflows to handle report cadences (daily, monthly, etc.) and dependencies between static table creation and report generation.

**Challenges and Solutions**
The HDFS-to-HDFS processing introduced complexities, as Spark's distributed nature differed from Informatica's centralized execution and PL/SQL's procedural logic.

Key challenges included:
• **Complex Logic Translation:** PL/SQL's nested loops and cursors were cumbersome to replicate. I used iterative DataFrame operations and temporary views to mimic sequential processing, though this required careful debugging.
• **Performance Bottlenecks:** Large joins between fact and dimension tables caused shuffles. I mitigated this with broadcast joins and partitioning, reducing execution time.
• **UDF Overhead:** Complex business logic required UDFs, which slowed

performance. I optimized by rewriting logic with native Spark functions where feasible and caching intermediate results.
• **Report Accuracy:** Ensuring identical report outputs was critical. I collaborated closely with analysts, using metadata-driven validation (inspired by my Neo4j lineage experience) to trace data flow and confirm results.

**Lessons Learned**
• **Balance Functionality and Performance:** UDFs were essential but costly; native Spark functions and optimization techniques were critical for scalability.
• **Metadata Matters:** Clear documentation and lineage tracking streamlined validation and stakeholder communication.
• **Iterative Testing Is Key:** Frequent validation with analysts caught discrepancies early, ensuring business continuity.

**The Journey's End and Beyond**
Completing the Informatica and stored procedure migration marked the culmination of this project, delivering a fully modernized data pipeline on Cloudera. The client now had a scalable, cost-efficient platform for BI reporting, with PySpark handling all ETL and reporting tasks.

Data Engineering · Data Engineer · Python Programming · Automation · Big Data

**Written by Anurag Sharma**
83 followers · 3 following

Edit profile

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!

## No responses yet

Anurag Sharma him/he

What are your thoughts?

## More from Anurag Sharma