From Imperative to Declarative

Building Maintainable ETL Pipelines in Spark

Bohdan Hashchuk

Why ETL Pipelines Matter

- High-volume data requires automation, reliability, and quality

File Edit Selection View Go Run Terminal Help

Intro ×

Why ETL Pipelines Matter

- High-volume data requires automation, reliability, and quality
- Imperative Spark pipelines become large, fragile, and hard to orchestrate

File Edit Selection View Go Run Terminal Help

Intro ×

Why ETL Pipelines Matter

- High-volume data requires automation, reliability, and quality
- Imperative Spark pipelines become large, fragile, and hard to orchestrate
- Declarative pipelines solve orchestration, state, and lineage automatically

File Edit Selection View Go Run Terminal Help

Intro ×

Why ETL Pipelines Matter

- High-volume data requires automation, reliability, and quality
- Imperative Spark pipelines become large, fragile, and hard to orchestrate
- Declarative pipelines solve orchestration, state, and lineage automatically
- Databricks implements Declarative Pipelines through Delta Live Tables, but the paradigm exists independently

File Edit Selection View Go Run Terminal Help

Intro Imperative ×

Imperative Pipelines in Spark

```python
songs_raw_stream = (
    spark.readStream
         .format("cloudFiles")
         .schema(songs_schema)
         .option("cloudFiles.format", "csv")
         .option("sep", "\t")
         .load(file_path)
)

songs_raw_query = (
    songs_raw_stream.writeStream
        .format("delta")
        .outputMode("append")
        .option("checkpointLocation", checkpoint_path)
        .start(bronze_path)
)
```

File Edit Selection View Go Run Terminal Help

Intro Imperative ×

Limitations of Imperative Spark

- No built-in data quality enforcement

File Edit Selection View Go Run Terminal Help

Intro Imperative ×

Limitations of Imperative Spark

- No built-in data quality enforcement
- No auto lineage

File Edit Selection View Go Run Terminal Help

Intro Imperative ×

Limitations of Imperative Spark

- No built-in data quality enforcement
- No auto lineage
- State management is manual

File Edit Selection View Go Run Terminal Help

Intro Imperative ×

Limitations of Imperative Spark

- No built-in data quality enforcement
- No auto lineage
- State management is manual
- Hard to add more stages (bronze → silver → gold)

File Edit Selection View Go Run Terminal Help

Intro Imperative ×

Limitations of Imperative Spark

- No built-in data quality enforcement
- No auto lineage
- State management is manual
- Hard to add more stages (bronze → silver → gold)
- Pipelines require external schedulers (Airflow, cron, jobs API)

File Edit Selection View Go Run Terminal Help

Intro Imperative SDP ×

What Is a Declarative Pipeline? (SDP)

"Define what your tables should contain, not how to produce them."

```
@dp.table(comment="Raw data from a subset of the Million Song Dataset; a collection
of features and metadata for contemporary music tracks.")
def songs_raw():
    return (spark.readStream
        .format("cloudFiles")
        .schema(songs_schema)
        .option("cloudFiles.format", "csv")
        .option("sep", "\t")
        .load(file_path)
    )
```

File Edit Selection View Go Run Terminal Help

Intro Imperative SDP ×

Characteristics

- Spark infers execution order

File Edit Selection View Go Run Terminal Help

Intro Imperative SDP ×

Characteristics

- Spark infers execution order
- Spark builds table dependencies

File Edit Selection View Go Run Terminal Help

Intro Imperative SDP ×

Characteristics

- Spark infers execution order
- Spark builds table dependencies
- Automatic incremental processing

File Edit Selection View Go Run Terminal Help

Intro Imperative SDP ×

Characteristics

- Spark infers execution order
- Spark builds table dependencies
- Automatic incremental processing
- Automatic fault tolerance

Characteristics

- Spark infers execution order
- Spark builds table dependencies
- Automatic incremental processing
- Automatic fault tolerance
- Cleaner, modular transformation logic

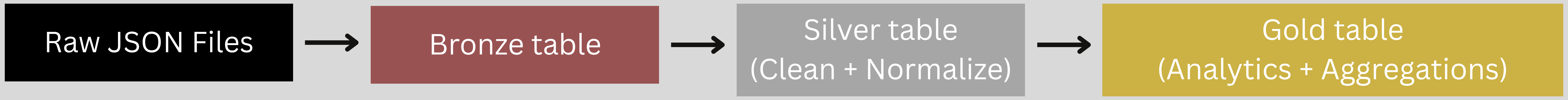
File Edit Selection View Go Run Terminal Help

Intro Imperative SDP Comparison ×

Spark Declarative Pipelines vs. Imperative Jobs

Feature	Imperative	Declarative
Programming style	Procedural	SQL-like, functional
Control	Manual	Automated
State Management	You Manage checkpoints	Framework Handles it
Lineage	None	Auto lineage graph
Optimization	Manual	Automatic
Fault Tolerance	You Implement	Built-in
Data Quality	Manual	Declarative rules

Example Pipeline Architecture


Raw JSON Files
Bronze table
Silver table (Clean + Normalize)
Gold table (Analytics + Aggregations)

Imperative Spark	SDP
You Write 3 jobs	You Declare 3 tables
You Manage Dependencies	Spark figures out dependencies

Tutorial in Databricks

THANK YOU FOR LISTENING