

# **Lab 4: Export Neo4j to Lakehouse**

**Bidirectional Data Flow**

## Concepts Introduced

**Delta Lake** - Open-source storage layer that brings ACID transactions to data lakes.

### Key Concepts:

- **Delta Tables** - Versioned, transactional tables stored as Parquet
- **Time Travel** - Query previous versions of your data
- **ACID Transactions** - Reliable writes, no partial failures
- **Unity Catalog Tables** - Governed Delta tables with access control

### Why Export Graph → Tables?

- Enable SQL-based tools (BI, Genie) to query graph data
- Leverage Databricks compute for large-scale analytics
- Create governed, shareable datasets from graph insights

## Basic Node Read

```
df = spark.read \
    .format("org.neo4j.spark.DataSource") \
    .option("url", "neo4j://localhost:7687") \
    .option("authentication.type", "basic") \
    .option("authentication.basic.username", "neo4j") \
    .option("authentication.basic.password", "password") \
    .option("labels", "Customer") \
    .load()

df.show()
```

The `labels` option specifies which nodes to read.

## What Gets Returned

The resulting DataFrame contains:

Column	Description
<id>	Neo4j internal node ID
<labels>	Array of all labels on the node
name	Node property
email	Node property
created_at	Node property

Angle brackets indicate metadata columns added by the Neo4j Spark Connector.

## Filter Pushdown

Spark filters become Cypher WHERE clauses:

```
df = spark.read \
    .option("labels", "Customer") \
    .load() \
    .filter("age > 30") \
    .filter("country = 'USA'")
```

Generated Cypher:

```
MATCH (n:Customer)
WHERE n.age > 30 AND n.country = 'USA'
RETURN ...
```

Filtering happens in Neo4j—less data transferred over the network.

## Column Pruning

Selecting columns limits what Neo4j returns:

```
df = spark.read \
    .option("labels", "Customer") \
    .load() \
    .select("name", "email")
```

Generated Cypher:

```
RETURN n.name, n.email
-- NOT: RETURN n (all properties)
```

**Benefit:** Reduces network bandwidth and memory.

## Aggregation Pushdown

Some aggregations execute in Neo4j:

```
df = spark.read \
    .option("labels", "Customer") \
    .load() \
    .groupBy("country") \
    .count()
```

**Supported:** COUNT, SUM, MIN, MAX, AVG

**Limitation:** Complex aggregations fall back to Spark.

## Basic Relationship Read

```
df = spark.read \
    .format("org.neo4j.spark.DataSource") \
    .option("url", "neo4j://localhost:7687") \
    .option("authentication.type", "basic") \
    .option("authentication.basic.username", "neo4j") \
    .option("authentication.basic.password", "password") \
    .option("relationship", "TRANSACTED_WITH") \
    .option("relationship.source.labels", "Customer") \
    .option("relationship.target.labels", "Merchant") \
    .load()
```

### Key options:

- **relationship** — The relationship type to read
- **relationship.source.labels** — Label(s) of the source node
- **relationship.target.labels** — Label(s) of the target node

## Relationship Direction

The Neo4j Spark Connector reads relationships in their **stored direction**:

```
(Customer)-[:TRANSACTED_WITH]->(Merchant)
```

**Both endpoints are always returned.**

Direction matters for:

- Understanding data flow
- Write operations (covered later)

You cannot reverse direction during read.

## Filtering Relationships

Filters work on relationship properties:

```
df = spark.read \
    .option("relationship", "TRANSACTIONED_WITH") \
    .option("relationship.source.labels", "Customer") \
    .option("relationship.target.labels", "Merchant") \
    .load() \
    .filter("rel.amount > 1000") \
    .filter("rel.date > '2024-01-01'")
```

Filter pushdown applies to relationship properties too.

# What We're Creating

14 Delta Tables in Unity Catalog:

Type	Tables	Purpose
Node Tables	customer, bank, account, company, stock, position, transaction	Entity data
Relationship Tables	has_account, at_bank, of_company, performs, benefits_to, has_position, of_security	Connection data

```
# Read from Neo4j
df = spark.read.format("org.neo4j.spark.DataSource")
    .option("labels", "Customer").load()

# Write to Delta
df.write.format("delta").saveAsTable("catalog.schema.customer")
```

Next: Create AI agents that query these tables