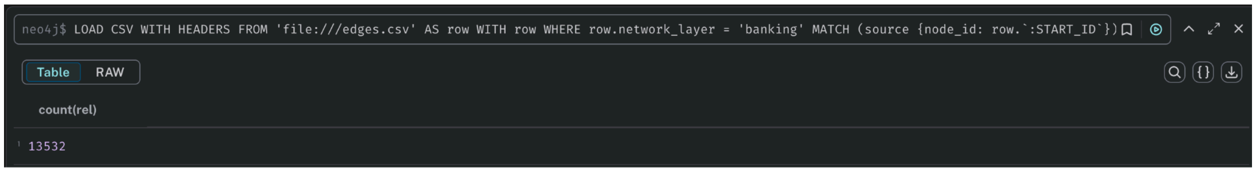
```
// STEP 1: Create constraints for better performance (optional but recommended)
CREATE CONSTRAINT bank_id IF NOT EXISTS FOR (b:bank) REQUIRE b.node_id IS
UNIQUE;
CREATE CONSTRAINT stock_id IF NOT EXISTS FOR (s:stock) REQUIRE s.node_id IS
UNIQUE;
CREATE CONSTRAINT inst_id IF NOT EXISTS FOR (i:institutional_investor) REQUIRE
i.node_id IS UNIQUE;
CREATE CONSTRAINT fdic_id IF NOT EXISTS FOR (f:fdic_bank) REQUIRE f.node_id IS
UNIQUE;

// STEP 2: Load BANK nodes
LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' AS row
WITH row WHERE row.node_type = 'bank'
CREATE (n:bank)
SET n.node_id = row.`node_id:ID`,
    n.name = row.name,
    n.institution_id = row.institution_id,
    n.tier = row.tier,
    n.total_assets = toFloat(row.total_assets),
    n.total_deposits = toFloat(row.total_deposits),
    n.total_loans = toFloat(row.total_loans),
    n.equity = toFloat(row.equity),
    n.num_branches = toInteger(row.num_branches),
    n.num_employees = toInteger(row.num_employees),
    n.network_layer = row.network_layer,
    n.created_at = row.created_at;

// STEP 3: Load STOCK nodes
LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' AS row
WITH row WHERE row.node_type = 'stock'
CREATE (n:stock)
SET n.node_id = row.`node_id:ID`,
    n.name = row.name,
    n.ticker = row.ticker,
    n.sector = row.sector,
    n.industry = row.industry,
    n.market_cap = toFloat(row.market_cap),
    n.trailing_pe = toFloat(row.trailing_pe),
    n.debt_to_equity = toFloat(row.debt_to_equity),
    n.beta = toFloat(row.beta),
    n.network_layer = row.network_layer,
    n.created_at = row.created_at;
```

```
// STEP 4: Load FDIC_BANK nodes
LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' AS row
WITH row WHERE row.node_type = 'fdic_bank'
CREATE (n:fdic_bank)
SET n.node_id = row.`node_id:ID`,
   n.name = row.name,
   n.cert = toInteger(row.cert),
   n.assets = toFloat(row.assets),
   n.city = row.city,
   n.state = row.state,
   n.network_layer = row.network_layer,
   n.created_at = row.created_at;

// STEP 5: Load INSTITUTIONAL_INVESTOR nodes
LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' AS row
WITH row WHERE row.node_type = 'institutional_investor'
CREATE (n:institutional_investor)
SET n.node_id = row.`node_id:ID`,
   n.name = row.name,
   n.cik = row.cik,
   n.network_layer = row.network_layer,
   n.created_at = row.created_at;

// STEP 6: Verify node counts
MATCH (n) RETURN labels(n)[0] AS node_type, count(n) AS count ORDER BY count DESC;

// Expected output:
// fdic_bank: 4,376
// bank: 100
// stock: 136
// institutional_investor: 3
// TOTAL: 4,615


// Check total nodes
MATCH (n) RETURN count(n);
// Should return: 4615

// Check by type
MATCH (n) RETURN labels(n)[0] AS type, count(n) AS count;
```

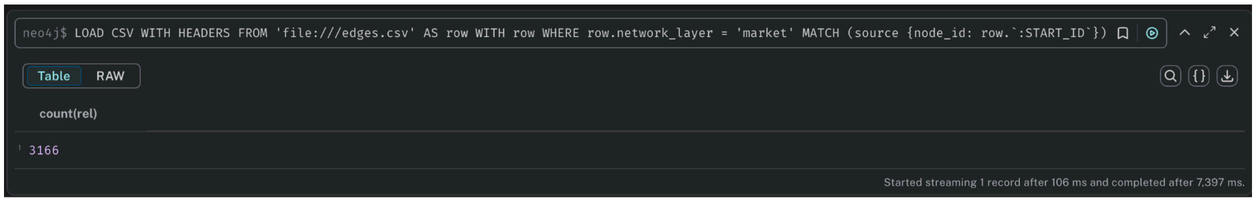
EDGES :

```
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'banking'
MATCH (source {node_id: row.`:START_ID`})
MATCH (target {node_id: row.`:END_ID`})
CALL apoc.create.relationship(source, row.`:TYPE`, {
   network_layer: row.network_layer,
   weight: toFloat(row.weight),
   transaction_date: row.transaction_date,
   maturity_days: toInteger(row.maturity_days),
   interest_rate: toFloat(row.interest_rate),
   currency: row.currency
}, target) YIELD rel
RETURN count(rel);
```



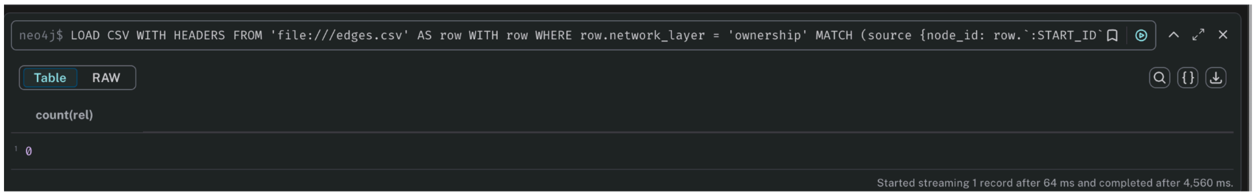
```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row WITH row WHERE row.network_layer = 'banking' MATCH (source {node_id: row.`:START_ID`})

count(rel)
13532
```

```
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'market'
MATCH (source {node_id: row.`:START_ID`})
MATCH (target {node_id: row.`:END_ID`})
CALL apoc.create.relationship(source, row.`:TYPE`, {
   network_layer: row.network_layer,
   weight: toFloat(row.weight),
   correlation: toFloat(row.correlation),
   window_days: toInteger(row.window_days)
}, target) YIELD rel
RETURN count(rel);
```



```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row WITH row WHERE row.network_layer = 'market' MATCH (source {node_id: row.`:START_ID`})

count(rel)
3166

Started streaming 1 record after 106 ms and completed after 7,397 ms.
```

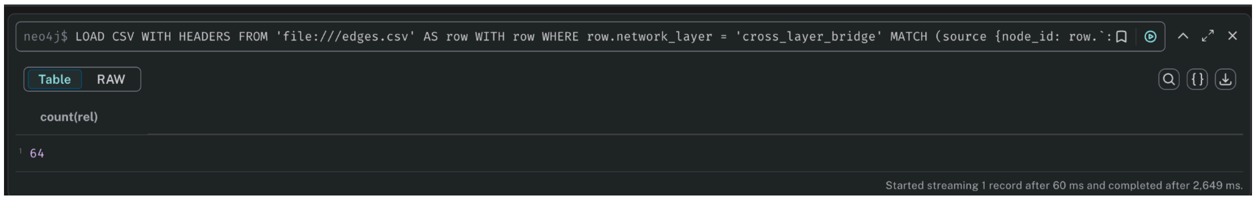
```
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'ownership'
MATCH (source {node_id: row.`:START_ID`})
MATCH (target {node_id: row.`:END_ID`})
CALL apoc.create.relationship(source, row.`:TYPE`, {
    network_layer: row.network_layer,
    weight: toFloat(row.weight),
    shares_held: toInteger(row.shares_held)
}, target) YIELD rel
RETURN count(rel);
```



```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row WITH row WHERE row.network_layer = 'ownership' MATCH (source {node_id: row.`:START_ID`
```

```
count(rel)
0
Started streaming 1 record after 64 ms and completed after 4,560 ms.
```

```
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'cross_layer_bridge'
MATCH (source {node_id: row.`:START_ID`})
MATCH (target {node_id: row.`:END_ID`})
CALL apoc.create.relationship(source, row.`:TYPE`, {
    network_layer: row.network_layer,
    layer_bridge: true,
    bridge_type: row.bridge_type,
    ticker: row.ticker,
    weight: toFloat(row.weight),
    propagation_factor: toFloat(row.propagation_factor)
}, target) YIELD rel
RETURN count(rel);
```

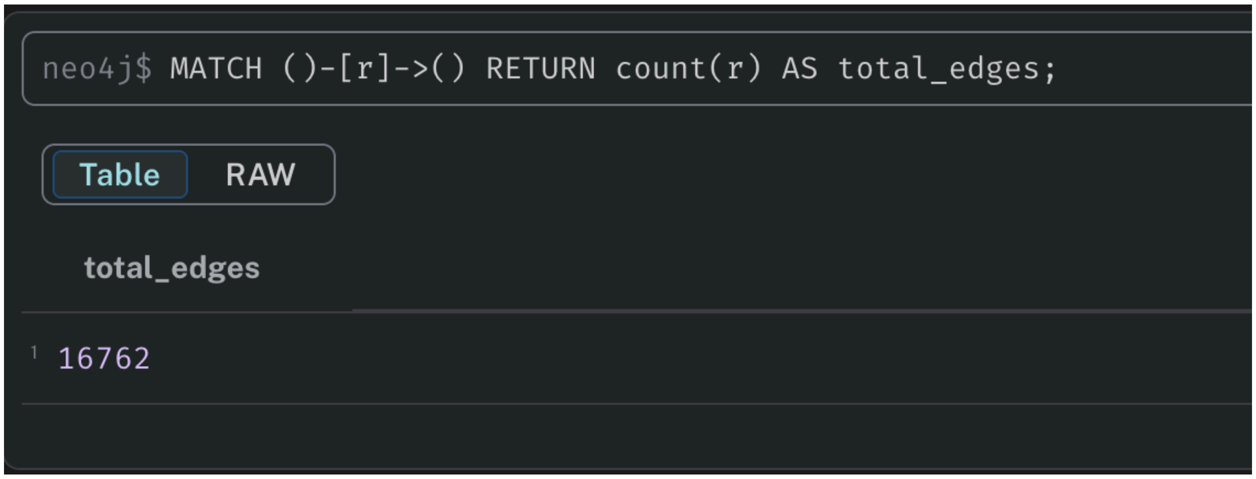


```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row WITH row WHERE row.network_layer = 'cross_layer_bridge' MATCH (source {node_id: row.`:
```

```
count(rel)
64
Started streaming 1 record after 60 ms and completed after 2,649 ms.
```

Total edges :

```
MATCH ()-[r]->()
RETURN count(r) AS total_edges;
// Should return: 19,411
```

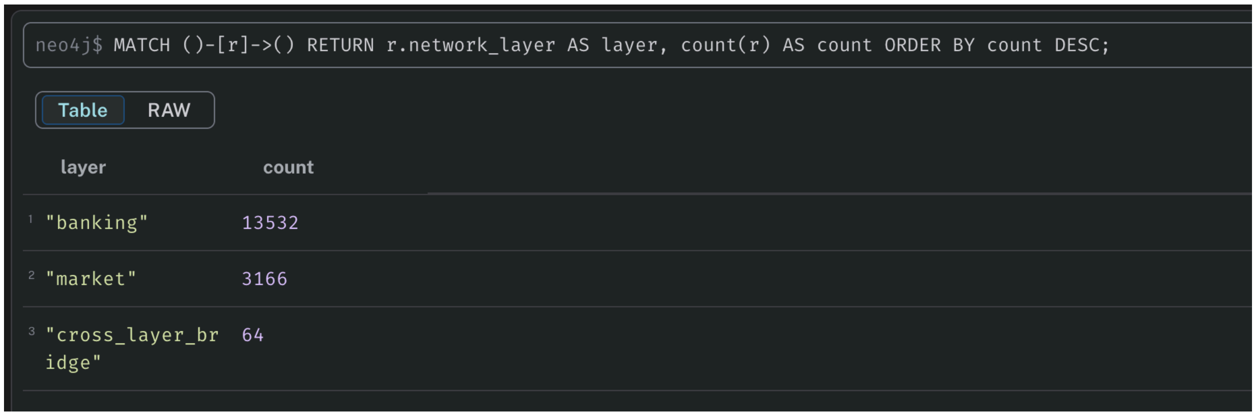


```
neo4j$ MATCH ()-[r]->() RETURN count(r) AS total_edges;
```

```
  total_edges
1 16762
```

Check by layer :
MATCH ()-[r]->()
RETURN r.network_layer AS layer, count(r) AS count
ORDER BY count DESC;

// Expected:
// banking: 13,532
// market: 3,934
// ownership: 1,881
// cross_layer_bridge: 64



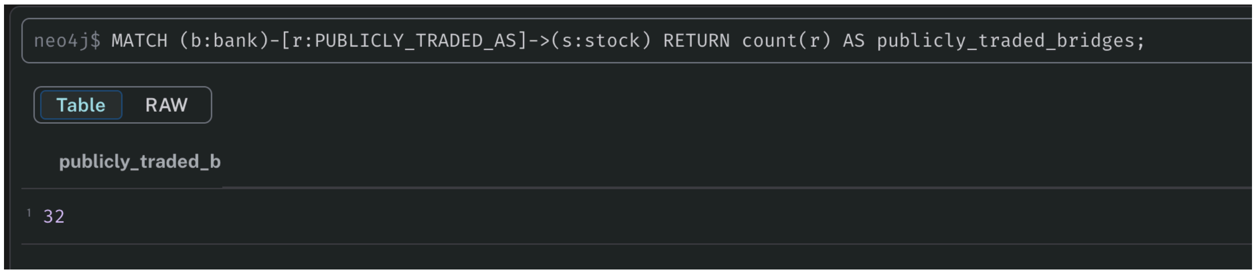
```
neo4j$ MATCH ()-[r]->() RETURN r.network_layer AS layer, count(r) AS count ORDER BY count DESC;
```

```
  layer              count
1 "banking"          13532
2 "market"           3166
3 "cross_layer_br    64
  idge"
```

✨

Verify cross -layers :

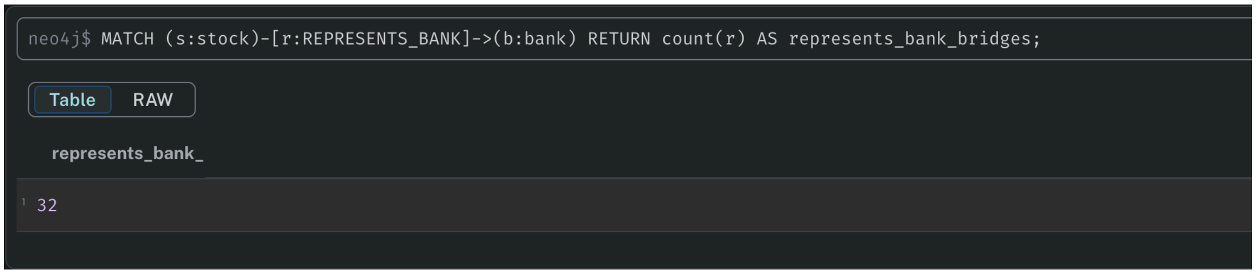
```
// Check PUBLICLY_TRADED_AS edges
MATCH (b:bank)-[r:PUBLICLY_TRADED_AS]->(s:stock)
RETURN count(r) AS publicly_traded_bridges;
// Should return: 32
```



```
neo4j$ MATCH (b:bank)-[r:PUBLICLY_TRADED_AS]->(s:stock) RETURN count(r) AS publicly_traded_bridges;
```

```
publicly_traded_b
1 32
```

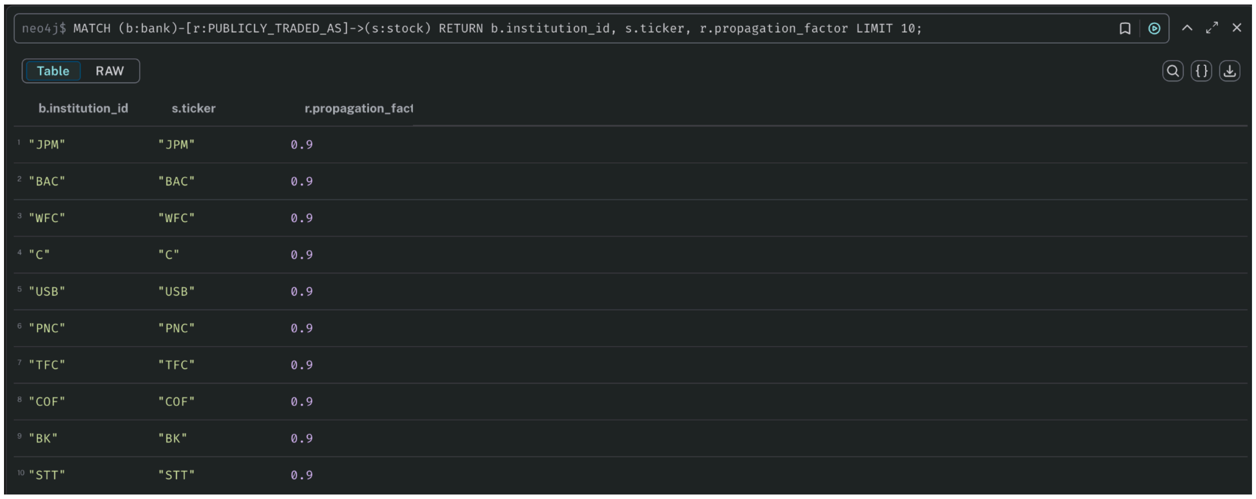
```
// Check REPRESENTS_BANK edges
MATCH (s:stock)-[r:REPRESENTS_BANK]->(b:bank)
RETURN count(r) AS represents_bank_bridges;
// Should return: 32
```



```
neo4j$ MATCH (s:stock)-[r:REPRESENTS_BANK]->(b:bank) RETURN count(r) AS represents_bank_bridges;
```

```
represents_bank_
1 32
```

```
// View some examples
MATCH (b:bank)-[r:PUBLICLY_TRADED_AS]->(s:stock)
RETURN b.institution_id, s.ticker, r.propagation_factor
LIMIT 10;
```



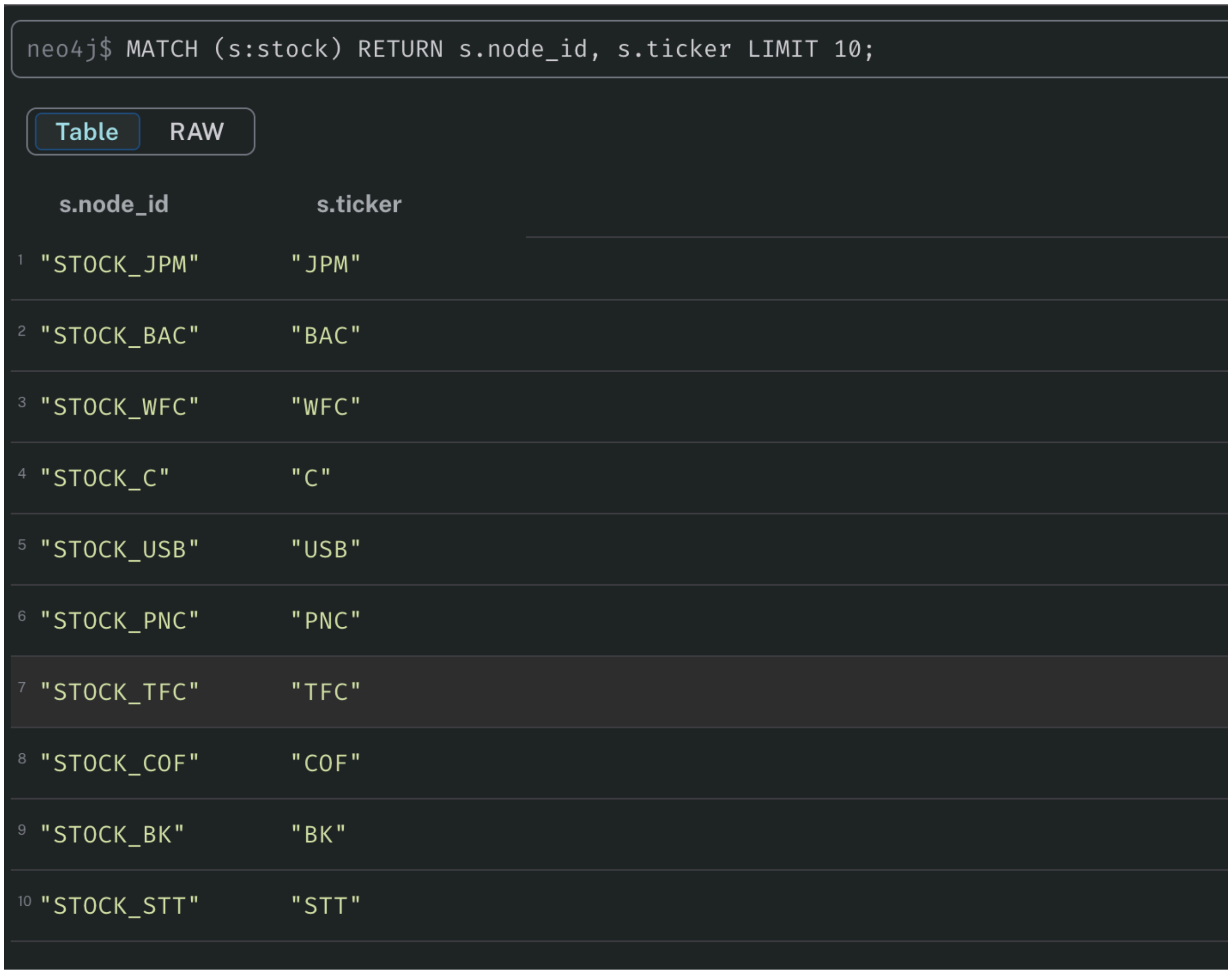
```
neo4j$ MATCH (b:bank)-[r:PUBLICLY_TRADED_AS]->(s:stock) RETURN b.institution_id, s.ticker, r.propagation_factor LIMIT 10;
```

	b.institution_id	s.ticker	r.propagation_fact
1	"JPM"	"JPM"	0.9
2	"BAC"	"BAC"	0.9
3	"WFC"	"WFC"	0.9
4	"C"	"C"	0.9
5	"USB"	"USB"	0.9
6	"PNC"	"PNC"	0.9
7	"TFC"	"TFC"	0.9
8	"COF"	"COF"	0.9
9	"BK"	"BK"	0.9
10	"STT"	"STT"	0.9

Test risk propagation path:

```
// Test if contagion path works ()
MATCH path = (b:bank {institution_id: 'JPM'})
             -[:PUBLICLY_TRADED_AS]->(s:stock)
             <-[:equity_ownership]-(i:institutional_investor)
RETURN path;
// Should return paths if bridges are working!
```

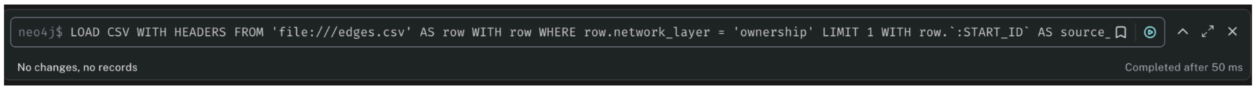
```
// Do we have stock nodes?
MATCH (s:stock)
RETURN s.node_id, s.ticker
LIMIT 10;

LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'ownership'
RETURN row.`:START_ID` AS source,
       row.`:END_ID` AS target,
       row.`:TYPE` AS rel_type
LIMIT 10;
```



```
neo4j$ MATCH (s:stock) RETURN s.node_id, s.ticker LIMIT 10;
```

```
   s.node_id        s.ticker
1  "STOCK_JPM"      "JPM"
2  "STOCK_BAC"      "BAC"
3  "STOCK_WFC"      "WFC"
4  "STOCK_C"        "C"
5  "STOCK_USB"      "USB"
6  "STOCK_PNC"      "PNC"
7  "STOCK_TFC"      "TFC"
8  "STOCK_COF"      "COF"
9  "STOCK_BK"       "BK"
10 "STOCK_STT"      "STT"
```

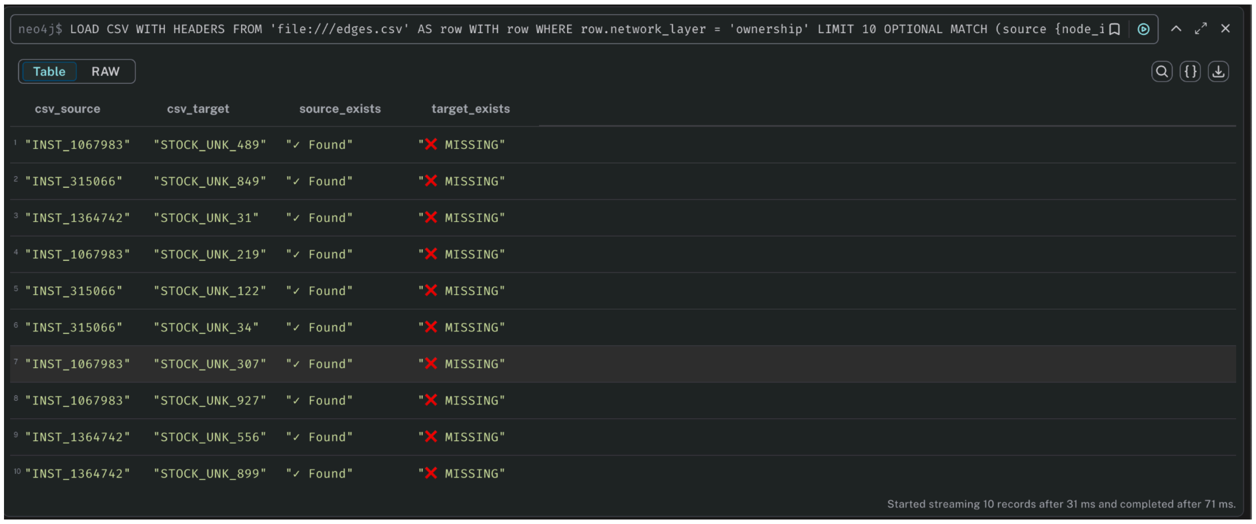
```
// Check a specific ownership edge from CSV
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'ownership' LIMIT 1
WITH row.`:START_ID` AS source_id, row.`:END_ID` AS target_id
MATCH (source {node_id: source_id})
MATCH (target {node_id: target_id})
RETURN source, target;
```



```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row WITH row WHERE row.network_layer = 'ownership' LIMIT 1 WITH row.`:START_ID` AS source_
No changes, no records                                                    Completed after 50 ms
```

```
// Find ownership edges where nodes don't exist
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
WITH row WHERE row.network_layer = 'ownership' LIMIT 10
OPTIONAL MATCH (source {node_id: row.`:START_ID`})
OPTIONAL MATCH (target {node_id: row.`:END_ID`})
```

```
RETURN
    row.`:START_ID` AS csv_source,
    row.`:END_ID` AS csv_target,
    CASE WHEN source IS NULL THEN '❌ MISSING' ELSE '✓ Found' END AS source_exists,
    CASE WHEN target IS NULL THEN '❌ MISSING' ELSE '✓ Found' END AS target_exists;
```



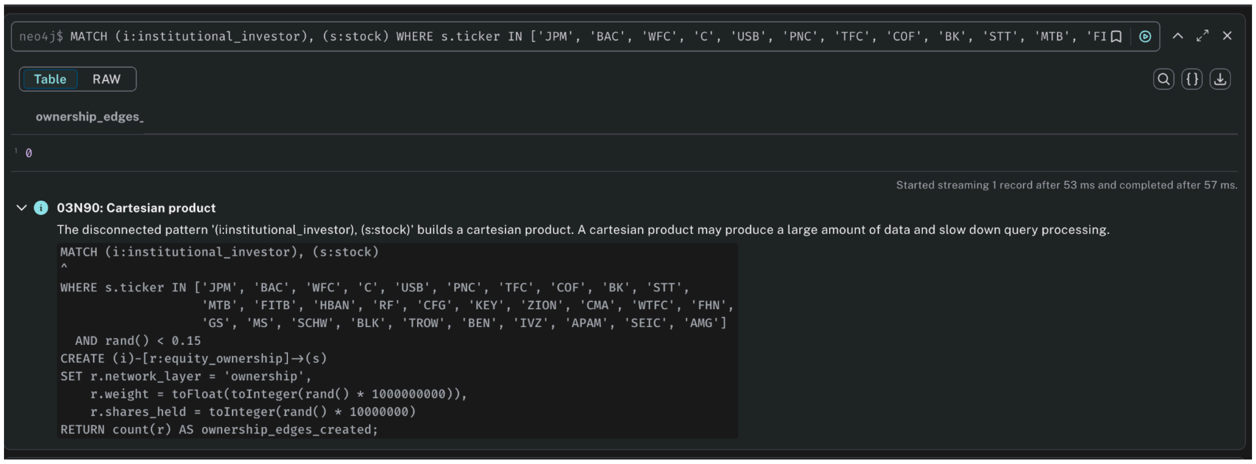
```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row WITH row WHERE row.network_layer = 'ownership' LIMIT 10 OPTIONAL MATCH (source {node_i
```

	csv_source	csv_target	source_exists	target_exists
1	"INST_1067983"	"STOCK_UNK_489"	"✓ Found"	"❌ MISSING"
2	"INST_315066"	"STOCK_UNK_849"	"✓ Found"	"❌ MISSING"
3	"INST_1364742"	"STOCK_UNK_31"	"✓ Found"	"❌ MISSING"
4	"INST_1067983"	"STOCK_UNK_219"	"✓ Found"	"❌ MISSING"
5	"INST_315066"	"STOCK_UNK_122"	"✓ Found"	"❌ MISSING"
6	"INST_315066"	"STOCK_UNK_34"	"✓ Found"	"❌ MISSING"
7	"INST_1067983"	"STOCK_UNK_307"	"✓ Found"	"❌ MISSING"
8	"INST_1067983"	"STOCK_UNK_927"	"✓ Found"	"❌ MISSING"
9	"INST_1364742"	"STOCK_UNK_556"	"✓ Found"	"❌ MISSING"
10	"INST_1364742"	"STOCK_UNK_899"	"✓ Found"	"❌ MISSING"

Started streaming 10 records after 31 ms and completed after 71 ms.

FIX :

// Run this in Neo4j Browser RIGHT NOW



```
neo4j$ MATCH (i:institutional_investor), (s:stock) WHERE s.ticker IN ['JPM', 'BAC', 'WFC', 'C', 'USB', 'PNC', 'TFC', 'COF', 'BK', 'STT', 'MTB', 'FI
```

	ownership_edges_
1	0

Started streaming 1 record after 53 ms and completed after 57 ms.

03N90: Cartesian product

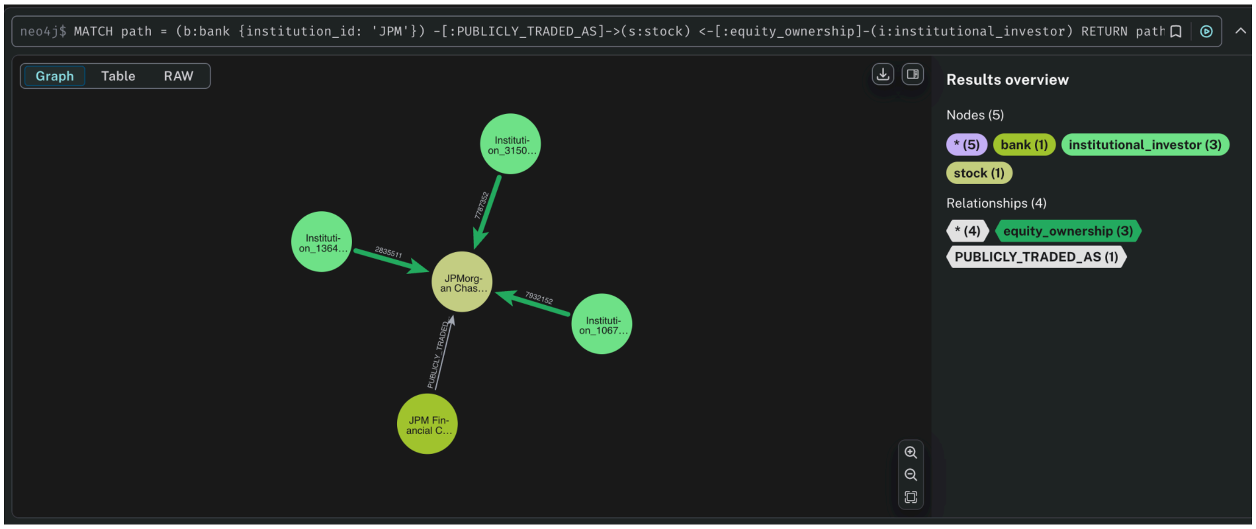
The disconnected pattern '(i:institutional_investor), (s:stock)' builds a cartesian product. A cartesian product may produce a large amount of data and slow down query processing.

```
MATCH (i:institutional_investor), (s:stock)
^
WHERE s.ticker IN ['JPM', 'BAC', 'WFC', 'C', 'USB', 'PNC', 'TFC', 'COF', 'BK', 'STT',
                   'MTB', 'FITB', 'HBAN', 'RF', 'CFG', 'KEY', 'ZION', 'CMA', 'WTFC', 'FHN',
                   'GS', 'MS', 'SCHW', 'BLK', 'TROW', 'BEN', 'IVZ', 'APAM', 'SEIC', 'AMG']
  AND rand() < 0.15
CREATE (i)-[r:equity_ownership]→(s)
SET r.network_layer = 'ownership',
    r.weight = toFloat(toInteger(rand() * 1000000000)),
    r.shares_held = toInteger(rand() * 10000000)
RETURN count(r) AS ownership_edges_created;
```

Test the fix :

```
// Test risk propagation
MATCH path = (b:bank {institution_id: 'JPM'})
```

```
            -[:PUBLICLY_TRADED_AS]->(s:stock)
            <-[:equity_ownership]-(i:institutional_investor)
RETURN path;




// Check how many institutional investors
MATCH (i:institutional_investor) RETURN count(i);
// Should return: 3

// Check how many financial stocks match
MATCH (s:stock)
WHERE s.ticker IN ['JPM', 'BAC', 'WFC', 'C', 'USB', 'PNC', 'TFC', 'COF', 'BK', 'STT',
                  'MTB', 'FITB', 'HBAN', 'RF', 'CFG', 'KEY', 'ZION', 'CMA', 'WTFC', 'FHN',
                  'GS', 'MS', 'SCHW', 'BLK', 'TROW', 'BEN', 'IVZ', 'APAM', 'SEIC', 'AMG']
RETURN count(s);
// Should return: ~30


// Each of the 3 institutional investors holds all 30 financial stocks
// Creates exactly 90 edges (3 × 30)
MATCH (i:institutional_investor)
MATCH (s:stock)
WHERE s.ticker IN ['JPM', 'BAC', 'WFC', 'C', 'USB', 'PNC', 'TFC', 'COF', 'BK', 'STT',
                  'MTB', 'FITB', 'HBAN', 'RF', 'CFG', 'KEY', 'ZION', 'CMA', 'WTFC', 'FHN',
                  'GS', 'MS', 'SCHW', 'BLK', 'TROW', 'BEN', 'IVZ', 'APAM', 'SEIC', 'AMG']
CREATE (i)-[r:equity_ownership]->(s)
SET r.network_layer = 'ownership',
   r.weight = toFloat(toInteger(rand() * 1000000000)),
   r.shares_held = toInteger(rand() * 10000000),
   r.relationship_type = 'equity_ownership'
RETURN count(r) AS edges_created;




// Test the complete risk propagation chain
MATCH path = (b:bank {institution_id: 'JPM'})
            -[:PUBLICLY_TRADED_AS]->(s:stock)
            <-[:equity_ownership]-(i:institutional_investor)
RETURN path;
```



```
neo4j$ MATCH path = (b:bank {institution_id: 'JPM'}) -[:PUBLICLY_TRADED_AS]->(s:stock) <-[:equity_ownership]-(i:institutional_investor) RETURN path
```

Results overview

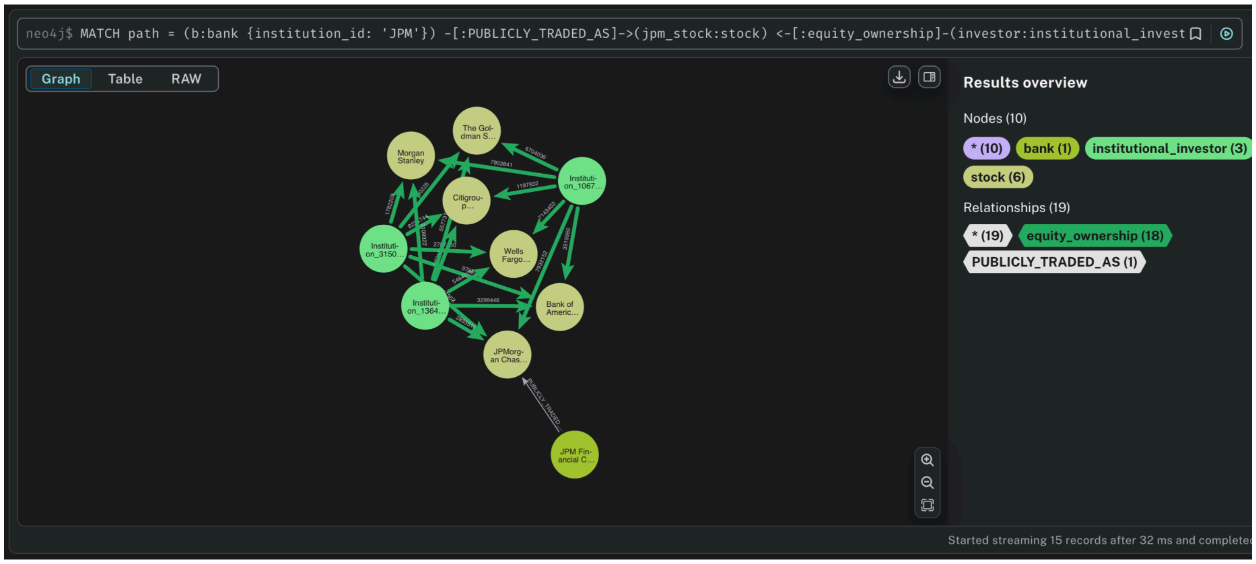
Nodes (5)

* (5) bank (1) institutional_investor (3) stock (1)

Relationships (4)

* (4) equity_ownership (3) PUBLICLY_TRADED_AS (1)

// Show the full contagion cascade
MATCH path = (b:bank {institution_id: 'JPM'})
-[:PUBLICLY_TRADED_AS]->(jpm_stock:stock)
<-[:equity_ownership]-(investor:institutional_investor)
-[:equity_ownership]->(other_stock:stock)
WHERE other_stock.ticker IN ['BAC', 'GS', 'MS', 'C', 'WFC']
RETURN path
LIMIT 20;



```
neo4j$ MATCH path = (b:bank {institution_id: 'JPM'}) -[:PUBLICLY_TRADED_AS]->(jpm_stock:stock) <-[:equity_ownership]-(investor:institutional_invest
```

Results overview

Nodes (10)

* (10) bank (1) institutional_investor (3) stock (6)

Relationships (19)

* (19) equity_ownership (18) PUBLICLY_TRADED_AS (1)

Started streaming 15 records after 32 ms and completed