
Loading data
neo4j
v 1.0

# Indexes/Constraints

# Indexes in Neo4j

- Neo4j supports two types of indexes on a node of a specific type:
  - single property
  - composite properties
- Indexes store redundant data that points to nodes with the specific property value or values.
- Unlike SQL, there is no such thing as a primary key in Neo4j. You can have multiple properties on nodes that must be unique.
- Add indexes before you create relationships between nodes.
- Creating an index on a property does not guarantee uniqueness.
  - But uniqueness and node key constraints are indexes that guarantee uniqueness.

# When indexes are used

Single property indexes are used to determine the starting point for graph traversal using:

- Equality checks =
- Range comparisons >,>=,<, <=
- List membership IN
- String comparisons STARTS WITH, ENDS WITH, CONTAINS
- Existence checks exists()
- Spatial distance searches distance()
- Spatial bounding searches point()

**Note**: Composite indexes are only used for equality checks and list membership.

# Creating a single-property index

Create a single-property index on the *released* property of all nodes of type *Movie*:

```
CREATE INDEX FOR (m:Movie) ON (m.released)
```

```
neo4j$ CREATE INDEX FOR (m:Movie) ON (m.released)
```

Table

Added 1 index, completed after 76 ms.

# Creating a composite index - 1

Suppose first that we added the property, *videoFormat* to every *Movie* node and set its value, based upon the released date of the movie as follows:

```
MATCH (m:Movie)
WHERE m.released >= 2000
SET m.videoFormat = 'DVD';
MATCH (m:Movie)
WHERE m.released < 2000
SET m.videoFormat = 'VHS'
```


All *Movie* nodes in the graph now have both a *released* and *videoFormat* property.

# Creating a composite index - 2

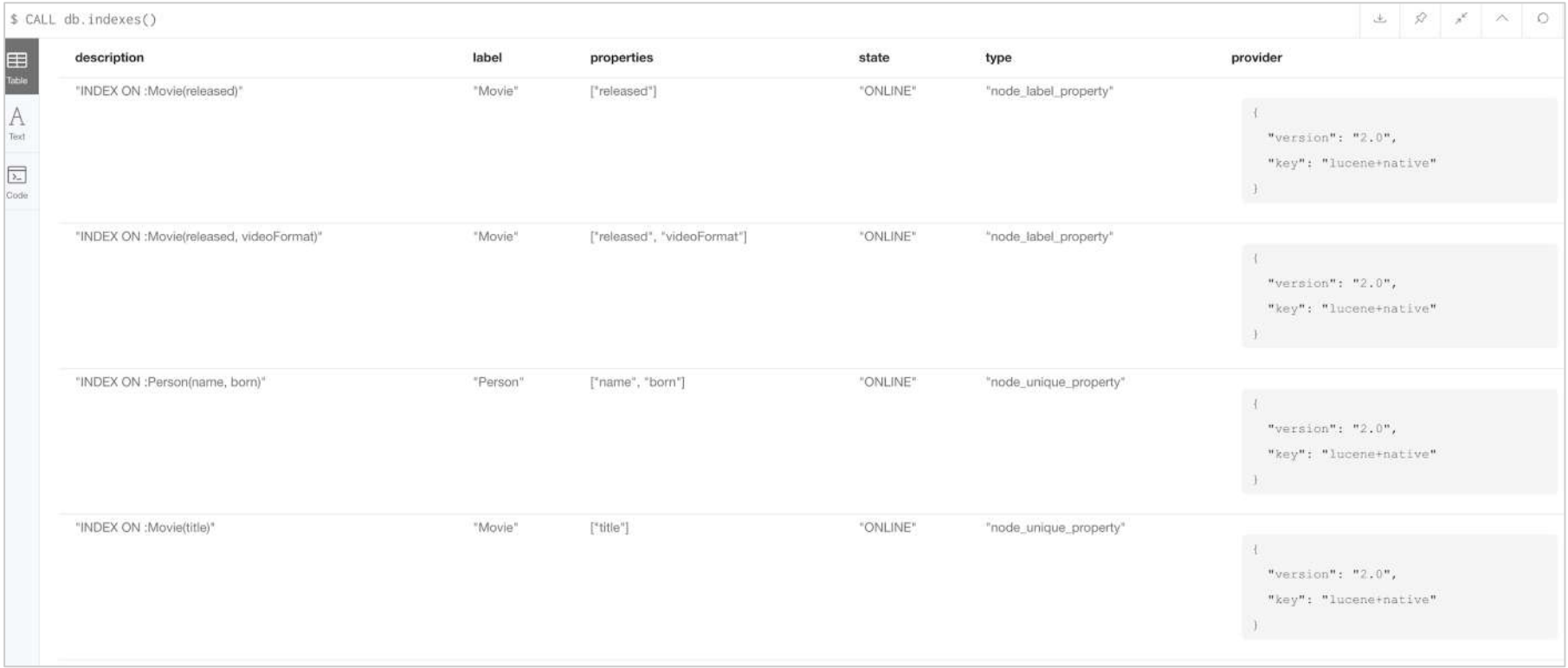
Create a composite index for every *Movie* node that uses the *videoFormat* and *released* properties:

```
CREATE INDEX FOR (m:Movie) ON (m.released, m.videoFormat)
```


**Note**: You can create a composite index with many properties.

# Retrieving indexes

```
CALL db.indexes()
```



# Dropping indexes

```
DROP INDEX index_name
```

The name of the index can be found using the **SHOW INDEXES command**, given in the output column name

```
neo4j$ DROP INDEX index_5762eea0;
```


Removed 1 index, completed after 5 ms.

# Loading Data

# Importing data

- LOAD CSV - CSV datasets up to 5 million rows
- Neo4j-admin import - CSV files with data sizes in TBs - one time batch build
- Neo4j ETL tool - connection to RDMBS sources
- APOC - many options to sync data in and out of Neo4j e.g. JSON, Elastic Search, MongoDB, HIVE, Spark
- Kettle - an open source ETL tool that has many connectors already pre-built (useful when gathering information from a large number of sources)
- Steaming - e.g. Kafka
- Spark
- Stored Procedures - Java based, can create custom integration to any service
- API/Driver - can create custom integration to any service in preferred language Java/Python/Go etc

# Importing data

CSV import is commonly used to import data into a graph where you can:

- Load data from a URL (http(s) or file).
- Process data as a stream of records.
- Create or update the graph with the data being loaded.
- Use transactions during the load.
- Transform and convert values from the load stream.
- Load up to 10M nodes and relationships.

# Steps for importing data

1. Determine the number of lines that will be loaded.
   - Is the load possible without special processing to handle transactions?
2. Examine the data and see if it may need to be reformatted.
   - Does data need alterations based upon your data requirements?
3. Make sure reformatting you will do is correct.
   - Examine final formatting of data before loading it.
4. Load the data and create nodes in the graph.
5. Load the data and create the relationships in the graph.

# Importing normalized data - 1

Example CSV file, **movies_to_load.csv**:

```
id,title,country,year,summary
1,Wall Street,USA,1987, Every dream has a price.
2,The American President,USA,1995, Why can't the most powerful man in the world have the one thing he wants most?
3,The Shawshank Redemption,USA,1994, Fear can hold you prisoner. Hope can set you free.
```

1. Determine the number of lines that will be loaded:

```
LOAD CSV WITH HEADERS
FROM 'http://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
RETURN count(*)
```

# Importing normalized data - 2

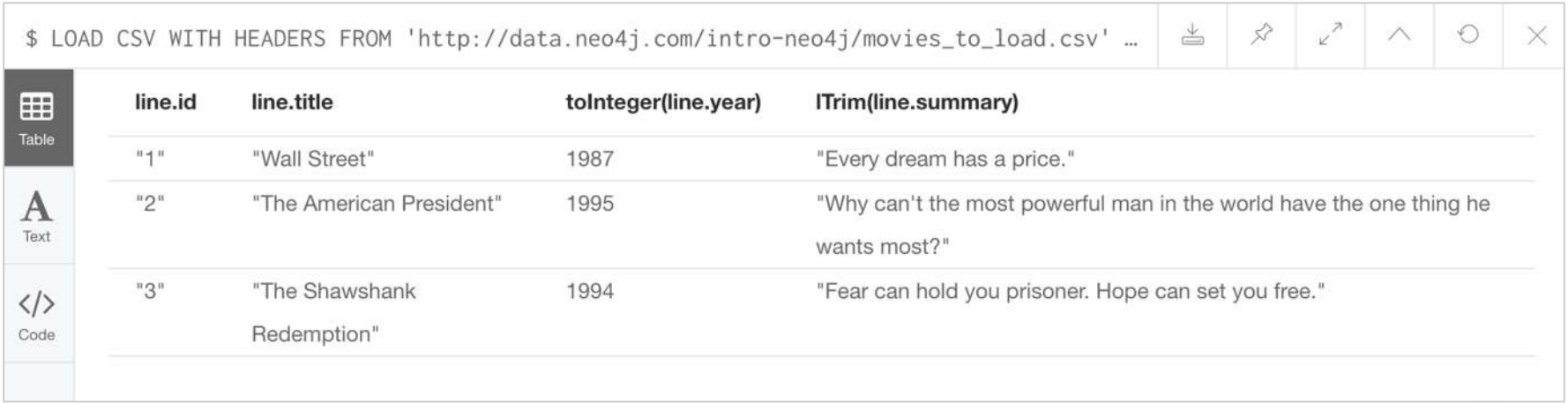
2. Examine the data and see if it may need to be reformatted:

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
RETURN * LIMIT 1
```

# Importing normalized data - 3

3. Format the data prior to loading:

```
LOAD CSV WITH HEADERS
FROM 'http://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
RETURN line.id, line.title, toInteger(line.year), trim(line.summary)
```




| line.id | line.title | toInteger(line.year) | lTrim(line.summary) |
|---|---|---|---|
| "1" | "Wall Street" | 1987 | "Every dream has a price." |
| "2" | "The American President" | 1995 | "Why can't the most powerful man in the world have the one thing he wants most?" |
| "3" | "The Shawshank Redemption" | 1994 | "Fear can hold you prisoner. Hope can set you free." |

# Importing normalized data - 4

4. Load the data and create the nodes in the graph:

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
CREATE (movie:Movie { movieId: line.id,
                      title: line.title,
                      released: toInteger(line.year) ,
                      tagline: trim(line.summary)})
```

```
$ LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/intro-neo4j/movies_to_load.csv" AS line CREATE (movie:Movie {…
```

Table

Added 3 labels, created 3 nodes, set 12 properties, completed after 289 ms.

# Importing the Person data

Example CSV file, **persons_to_load.csv**:

```
Id,name,birthyear
1,Charlie Sheen, 1965
2,Oliver Stone, 1946
3,Michael Douglas, 1944
4,Martin Sheen, 1940
5,Morgan Freeman, 1937
```

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/persons_to_load.csv'
AS line
MERGE (actor:Person { personId: line.Id })
ON CREATE SET actor.name = line.name,
              actor.born = toInteger(trim(line.birthyear))
```

We use MERGE to ensure that we will not create any duplicate nodes

# Creating the relationships

Example CSV file, **roles_to_load.csv**:

```
personId,movieId,role
1,1,Bud Fox
4,1,Carl Fox
3,1,Gordon Gekko
4,2,A.J. MacInerney
3,2,President Andrew Shepherd
5,3,Ellis Boyd 'Red' Redding
```

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/roles_to_load.csv'
AS line
MATCH (movie:Movie { movieId: line.movieId })
MATCH (person:Person { personId: line.personId })
CREATE (person)-[:ACTED_IN { roles: [line.role]}]->(movie)
```

# Importing denormalized data

Example CSV file, **movie_actor_roles_to_load.csv**:

```
title;released;summary;actor;birthyear;characters
Back to the Future;1985;17 year old Marty McFly got home early last night. 30 years early.;Michael J. Fox;1961;Marty McFly
Back to the Future;1985;17 year old Marty McFly got home early last night. 30 years early.;Christopher Lloyd;1938;Dr. Emmet Brown
```

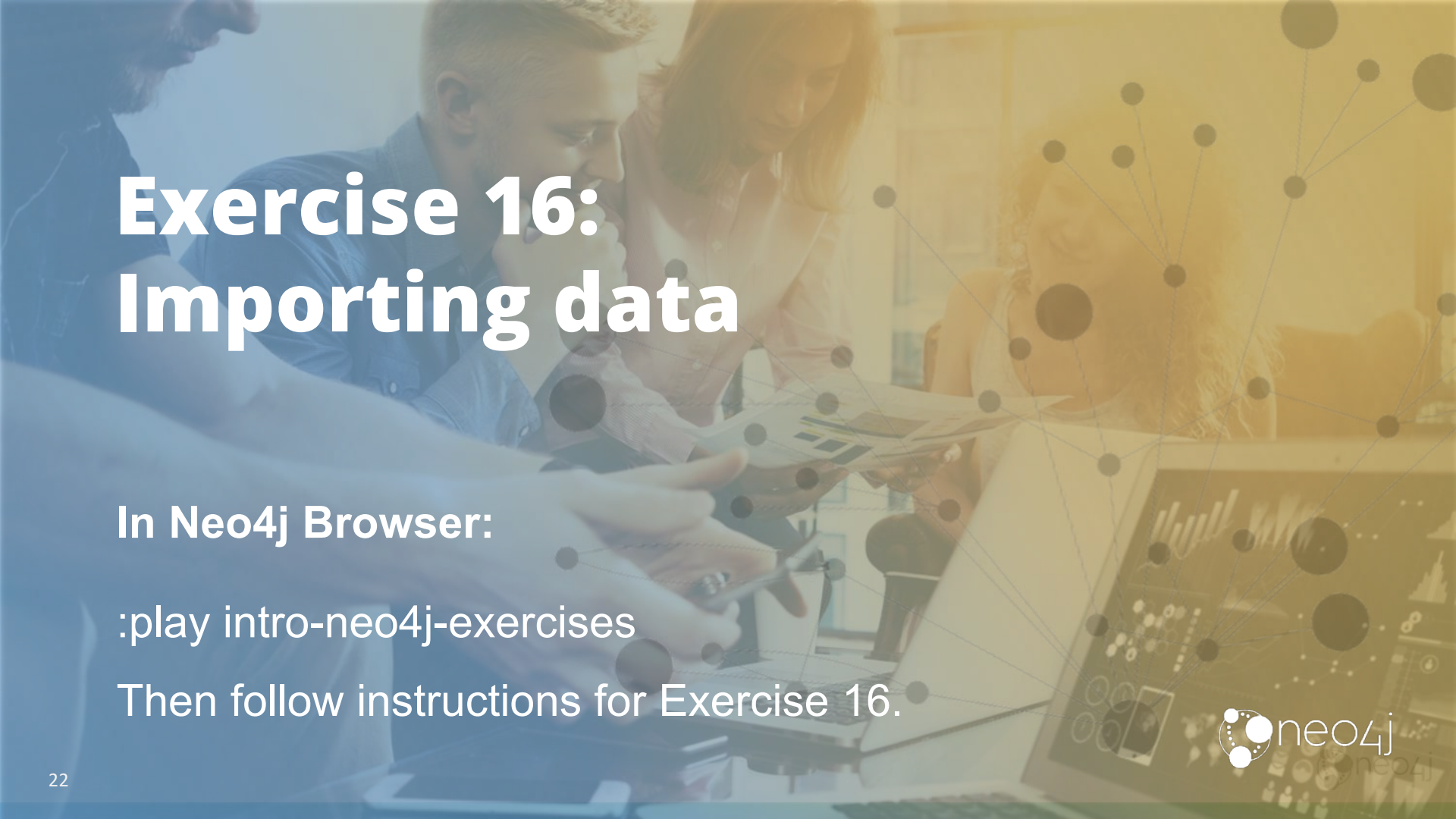
```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/movie_actor_roles_to_load.csv'
AS line FIELDTERMINATOR ';'
MERGE (movie:Movie { title: line.title })
ON CREATE SET movie.released = toInteger(line.released),
              movie.tagline = line.summary
MERGE (actor:Person { name: line.actor })
ON CREATE SET actor.born = toInteger(line.birthyear)
MERGE (actor)-[r:ACTED_IN]->(movie)
ON CREATE SET r.roles = split(line.characters,',')
```

# Importing a large dataset

< very large dataset that is greater than 10K rows>


```
:auto USING PERIODIC COMMIT
LOAD CSV . . .
```

**Benefit**: The graph engine will automatically commit data to avoid memory issues.



# Exercise 16: Importing data

**In Neo4j Browser:**

:play intro-neo4j-exercises

Then follow instructions for Exercise 16.