Introduction

neo4j

- NoSQL Graph-oriented database
- CSV/JSON documents
- Implemented in Java
- DSL: **Cypher**
- ACID / Causal consistency

- Some readings:
  - Graph databases (O'Reilly ed.)

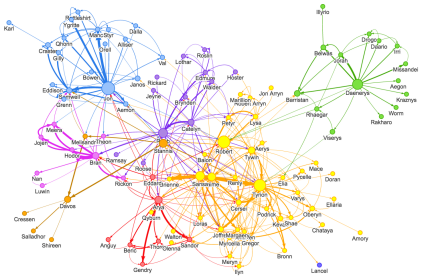
Applications using Neo4j

Adobe · Walmart · ebay · UBS · MONSANTO

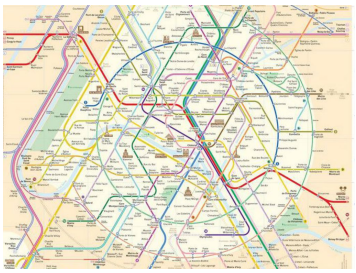
Microsoft · IBM · VOLVO · NASA

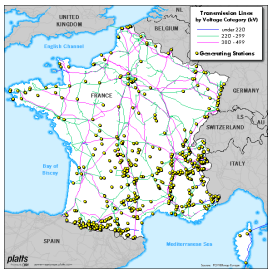
orange · Airbnb · KERBEROS Compliance · Management · Systeme · FT FINANCIAL TIMES

Case Studies

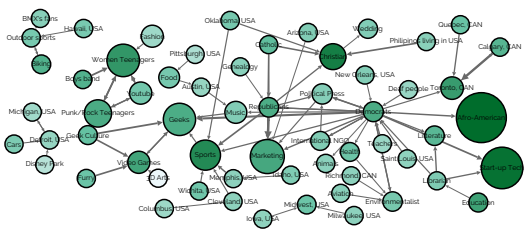


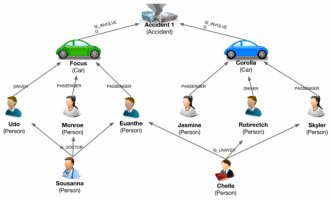
Social networks (analysis/reco)

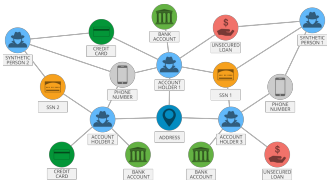




Geo networks (recommandations)







Typed network (Fraud detection)

Neo4j – A graph database

- Storage management
  - Typing
  - Structures
  - Cache Management
- Query on graphs
  - Query language
  - Data manipulation
- Query optimization
  - Indexes
  - Execution plans
- Procedures & Functions

neostore.**nodestore**.db
neostore.**relationshipstore**.db
neostore.**propertystore**.db
neostore.propertystore.index
neostore.propertystore.strings
neostore.propertystore.arrays

9 bytes : *max 2^{35} nodes*
1b Flag
+ 4b for 1st ID rel
+ 4b for 1st ID prop

33 bytes : *max 2^{35} relationship*
1b Flag
+ 8b for ID start & end node
+ 4b for type pointer
+ 16b for relationship chain
+ 4b for 1st ID prop

max 2^{38} properties

```
MATCH (a:Actor) –[:acted_in]-> (m:Movie)
WHERE m.title = "Star Wars
RETURN a.name
```

Schema indexing
Properties indexing (+ composite)
Full-text indexing
Spatial indexing

```
CREATE INDEX ON :Person(name)
```

https://neo4j.com/developer/kb/fulltext-search-in-neo4j/

https://neo4j-contrib.github.io/spatial/0.24-neo4j-3.1/index.html

JAVA programming (extensible)
CALL dbms.procedures()
Libraries : apoc, load, security, db, graph algorithms

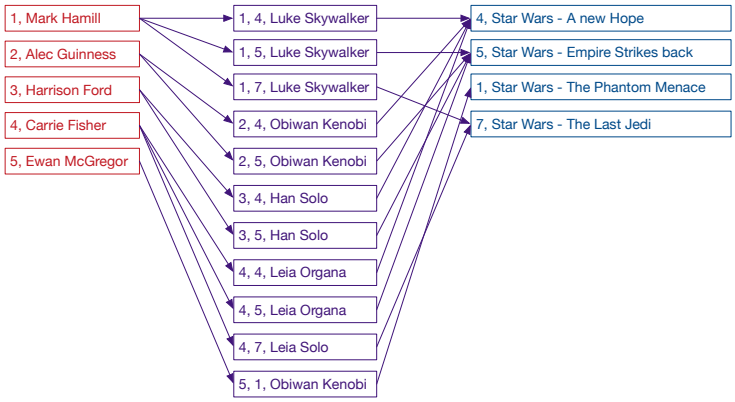
From RDBMS to GDBMS

Person
1, Mark Hamill
2, Alec Guinness
3, Harrison Ford
4, Carrie Fisher
5, Ewan McGregor

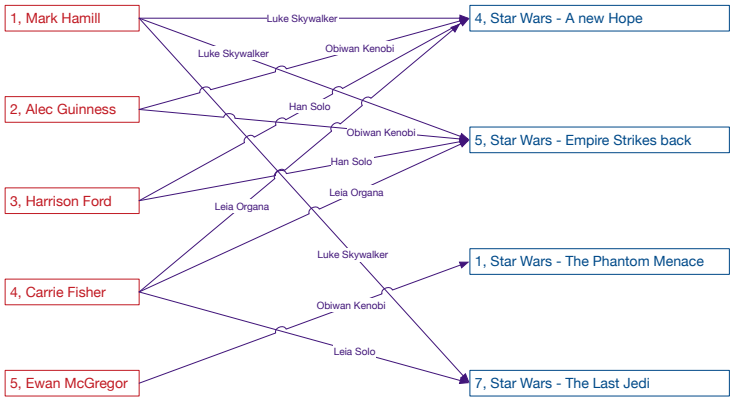
acted_in
1, 4, Luke Skywalker
1, 5, Luke Skywalker
1, 7, Luke Skywalker
2, 4, Obiwan Kenobi
2, 5, Obiwan Kenobi
3, 4, Han Solo
3, 5, Han Solo
4, 4, Leia Organa
4, 5, Leia Organa
4, 7, Leia Solo
5, 1, Obiwan Kenobi

Movie
4, Star Wars - A new Hope
5, Star Wars - Empire Strikes back
1, Star Wars - The Phantom Menace
7, Star Wars - The Last Jedi

From RDBMS to GDBMS



From RDBMS to GDBMS

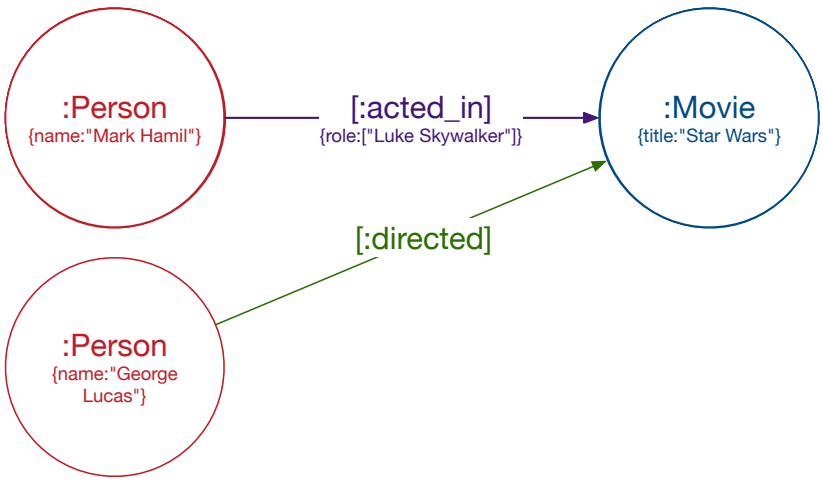


Evolutions

- V1.3 (2011)
  - Store formats, Lucene Index, cache management, transactions (ACID)
- V2.0 (2013)
  - Node labels, Cypher enhancement, Neo4j Browser
- V3.0 (2015)
  - Bolt (network protocol for drivers), Cypher cost planner, spatial functions, variables
- V3.4 (05-2018)
  - Multi-clustering (partitioning graphs with flag/geo)
  - Data-Type for Space and Time
- V3.5 (11-2018)
  - Performances on indexes (nodes&rel, full-text, sorting)
  - Security enhancement

Graph data model

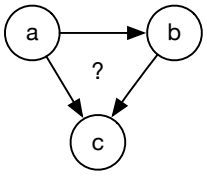
- **Nodes**
  - Type
  - Labels[1]
- **Relations**
  - Between 2 nodes
  - Oriented
  - Type
  - Labels

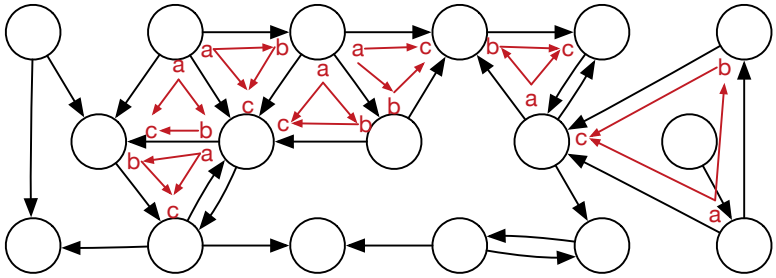


1 – case sensitive

Cypher – "SQL for graphs"

- Motivation
  - Pattern query



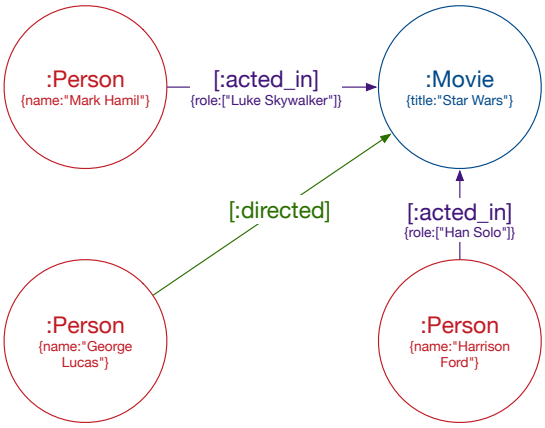


Cypher – Pattern Matching

- **MATCH**
  - () --> () Filtering pattern
  - **()** node
  - (**:Person**) typed node
  - (**a**:Person) node reference "*a*" in the query (variable)
  - (:Person**{name:"Mark Hamill"}**) Direct filter an node properties
  - **-->** relationship
  - -[**:acted_in**]-> typed relationship
  - -[**r**:acted_in]-> relationship reference "*r*" in the query (variable)
- **WHERE**
  - **a.name =** "Mark Hamill" Filter on properties
- **RETURN**
  - Projection of values
  - a.name properties
  - a nodes, relationships
- Other clauses : OPTIONAL MATCH, LIMIT, ORDER BY, MERGE, CREATE, ON CREATE, ON MATCH, SET, DELETE

Cypher – Simple queries

1. List of Persons linked to something
   - MATCH (p:Person) --> () RETURN p
2. List of Persons with their relationship to a Movie
   - MATCH (p:Person) -[r]-> (:Movie) RETURN p, r
3. List of Persons who acted in something
   - MATCH (p:Person) -[:acted_in]-> () RETURN p
4. List of Persons who acted in something which was directed by something
   - MATCH (p:Person) -[:acted_in]-> (m) <-[:directed]- (d)
     RETURN d.name, p.name, m.title
5. Titles of things linked to a Person who's name is Mark Hamill
   - MATCH (mh:Person{name:"Mark Hamill"}) --> (m) RETURN m.title
6. Titles of things in which has acted in a Person who's role was Luke Skywalker
   - MATCH (:Person) -[r:acted_in]-> (m)
     WHERE "Luke Skywalker" IN r.role RETURN m.title
7. Titles and name of things who acted in a thing with a Person who's name is Mark Hamill (without himself)
   - MATCH (mh) -[:acted_in]-> (m) <-[:acted_in]- (p)
     WHERE mh.name = "Mark Hamill" and mh != p
     RETURN m.title AS title, p.name AS name



Cypher – Pattern queries equivalence

- MATCH (p:Person) -[:acted_in]-> (m) <-[:directed]- (d)
  RETURN d.name, p.name, m.title

⇔

- MATCH (p:Person) -[:acted_in]-> (**m**), (**m**) <-[:directed]- (d)
  RETURN d.name, p.name, m.title

⇔

- MATCH (p:Person) -[:acted_in]-> (**m**), (d) **-**[:directed]**->** (**m**)
  RETURN d.name, p.name, m.title

Cypher – Complex queries (1/2)

- **Group by** with aggregate functions
  - MATCH (a) –[:acted_in]-> (m) <-[:directed]- (d)
    RETURN a.name, d.name, COUNT(*) as NB

a.name	d.name	NB
Mark Hamill	George Lucas	1
Harrison Ford	George Lucas	2
Harrison Ford	Steven Spielberg	4

  - MATCH (a) –[:acted_in]-> (m) <-[:directed]- (d)
    RETURN a.name, d.name,
    COLLECT(m.title) as movies

a.name	d.name	movies
Mark Hamill	George Lucas	SW1
Harrison Ford	George Lucas	American Graffiti, SW1
Harrison Ford	Steven Spielberg	Indiana Jones 1, IJ2, IJ3, IJ4

Cypher – Complex queries (2/2)

- Complex path chaining
  - Actors who played with Harrison Ford and also directed a movie

```
MATCH (hf:Person) -[:acted_in]-> (m:Movie), (ad) -[:acted_in]-> (m)
WHERE hf.name="Harrison Ford" AND (ad) -[:directed]-> ()
RETURN DISTINCT ad.name
```

  - Actors who played with Harrison Ford but not when Mark Hamill played

```
MATCH (hf:Person) -[:acted_in]-> (m:Movie), (ad) -[:acted_in]-> (m), (mh:Person)
WHERE hf.name="Harrison Ford" AND mh.name="Mark Hamill"
        AND (ad) -[:directed]-> ()
        AND NOT (mh) -[:acted_in]-> (m)
RETURN DISTINCT ad.name
```

Cypher – Miscellaneous

- type(r) – give the node/relationship type
- a.name CONTAINS 'Ford'
  - STARTS WITH / ENDS WITH
  - ~regex
- OPTIONAL MATCH (a) --> (r)
  - The relationship is not mandatory
- RETURN DISTINCT a.name
  - Distinct values
- ORDER BY a.name

RefCard : https://neo4j.com/docs/cypher-refcard/current/

Cypher – Some training

- Produce the Cypher queries for those sentences
  - Person who has directed the movie Star Wars
  - Title and name of persons who acted in the movies he directed
  - Actors name who played with Tom Hanks and older than him
  - Movies' title where Tom Hanks and Kevin Bacon played together
  - Idem with a common director for two movies
  - Top 5 couples of actors who played together

Build your own graph – Nodes

- **CREATE** (p:Person{ID:1})
  - Create a new node
- **MERGE** (p:Person{ID:1})
  - Create a new node if it does not exists
  - Or get the existing node
- **MATCH** (p:Person{ID:1}) **SET** p.name="Mark Hamill"
  - Add/Modify a property
- **MERGE** (p:Person{ID:1})
  ON CREATE **SET** p.updates = 1 //when created
  ON MATCH **SET** p.updates = p.updates + 1 //when matched

Build your own graph – Relationships

- **MATCH** (dr:Person{name:"Daisy Ridley"}), (sw9:Movie{title:"Star Wars IX"})
  MERGE (dr) **-[:acted_in {roles:["Rey"]}]->** (sw9)
- **MATCH** (dr:Person{name:"Daisy Ridley"})
  DELETE dr
  - Applied only if « dr » is not connected to any node
- **MATCH** (dr:Person{name:"Daisy Ridley"})
  OPTIONAL MATCH (dr) -[r]-> ()
  DELETE dr, r
- **MATCH** (a1:Person) --> (m:Movie) <-- (a2:Person)
  MERGE (a1) **-[:knows]-** (a2) //undirected relationship

Cypher – Hard queries (1/2)

- **Unlimited path** length (possible connections)

```
MATCH (mh:Person{name:"Mark Hamill"}) -[:knows*]- (p:Person)
WHERE NOT (mh) -[:knows]- (p)
RETURN DISTINCT p.name
```

- **Path length** from 2 to 3

```
MATCH (mh:Person{name:"Mark Hamill"}) -[:knows*2..3]- (p:Person)
WHERE NOT (mh) -[:knows]- (p)
RETURN DISTINCT p.name
```

- **Shortest path** between two nodes

```
MATCH p=shortestpath( (mh:Person) -[:knows*]-> (em:Person) )
WHERE mh.name = "Mark Hamill » AND em.name = "Ewan McGregor"
RETURN length( rels(p) )
```


Cypher – Hard queries (2/2)

- Graph manipulations:
  - Graph algorithms: centralities (PageRank, betweenness), communities (Louvain, Infomap), pathes (shortestpath, SpanningTree)
  - APOC: XML, JSON, JDBC
  - ETL: Data modeling
  - Neosemantics: XML/RDF

- External libraries import
  - Modify config file: $Neo4j_folder/conf/neo4j.conf
  - Add the extension name: dbms.unmanaged_extension_classes=XXX.extension=/XXX
  - Call the lib: CALL XXX.*function_name*.(*params*)

Cypher – Import datasets

```
LOAD CSV WITH HEADERS FROM "file:/actors.csv" as l
MERGE (a:Person{id:toInteger(l.IDA)})
MERGE (m:Movie{id:toInteger(l.IDM)})
SET a.name=l.name, m.title=l.title
MERGE (a) -[r:acted_in]-> (m)
ON CREATE SET r.roles = [l.role]
ON MATCH SET r.roles = r.roles + l.role
```

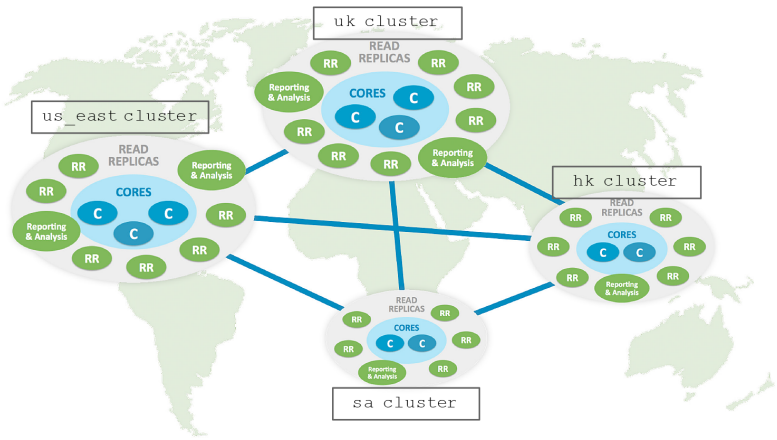
IDA	name	IDM	title	role
1	Mark Hamill	1	SW1	Luke
1	Mark Hamill	2	SW2	Luke
2	Carrie Fisher	1	SW1	Leia
3	Harrison Ford	1	SW1	Han

File « actors.csv » must be placed in $NEO4J_FOLDER/**import**

Create index before import!
CREATE INDEX ON :Person(id)
CREATE INDEX ON :Movie(id)

Sharding: "Distribution" & Replication

- **Causal Clustering**
  - Causal cluster Architecture
    - Core servers
      - Causal Consistency
    - Replicates
- **Multi-clustering**
  - Multitenant database
    - Multiple dbs
    - Shared discovery service
  - Consensus consistency
    - Locally: causal clustering



https://neo4j.com/docs/operations-manual/current/clustering-advanced/multi-clustering/introduction/