

Paweł Rajba

pawel@cs.uni.wroc.pl

<http://pawel.ii.uni.wroc.pl/>

Neo4j & Cassandra

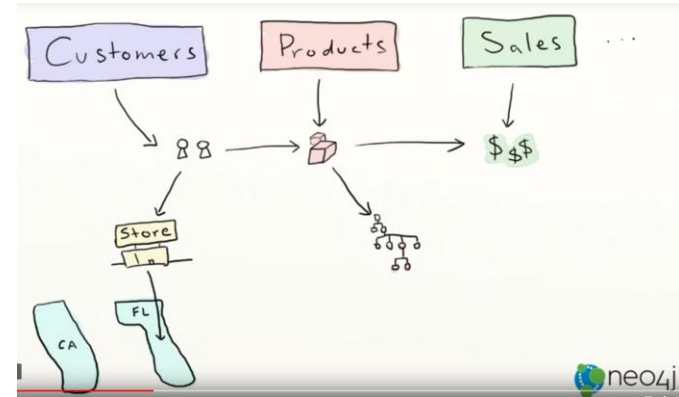
Neo4j

Agenda

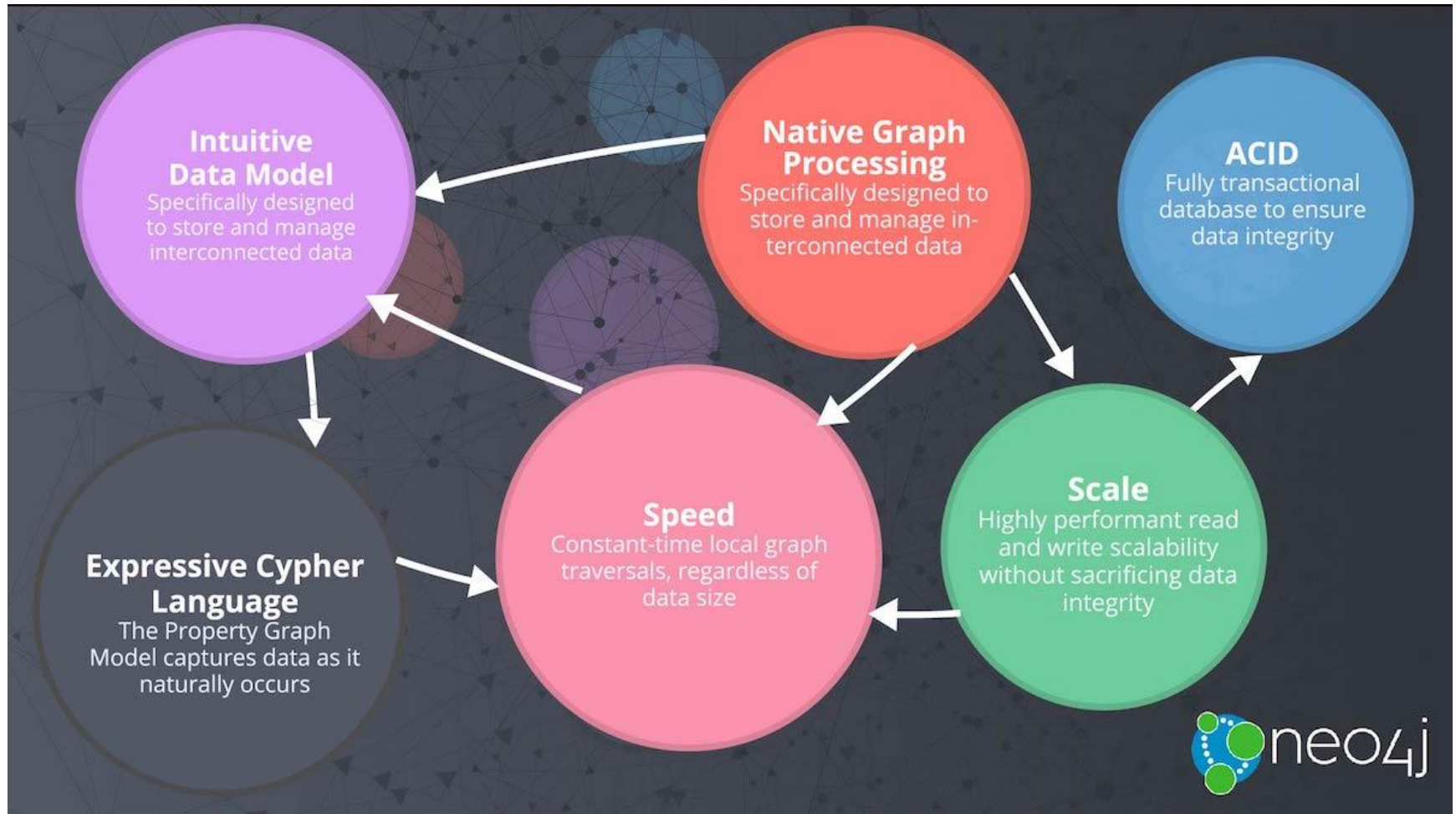
- Introduction
- Graph model
- Properties of graph database
- Cypher and example database

Introduction

- Website: <https://neo4j.com/>
- A graph database
 - Focusing primarily on relations
- Cypher as a query language
 - With roots in SQL
- Drivers for Popular Programming Languages
 - .Net, Java (also Spring), JavaScript, and Python
 - Communication based on binary “Bolt” protocol



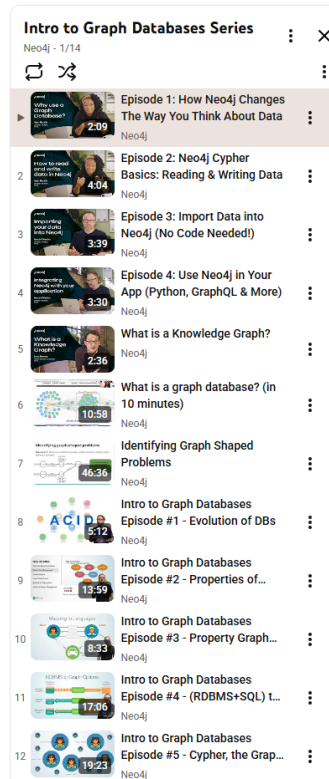
Introduction



Introduction

- A series of introduction videos available

<https://www.youtube.com/playlist?list=PLgHl4pk2FsvWM9GWaguRhICQ-pa-ERd4U>



Introduction

NEO4j USE CASES

Real Time Recommendations

Master Data Management

Fraud Detection

Graph Based Search





















Network & IT-Operations

Identity & Access Management

Filter by Industry: All Industries

Filter by Use Case: All Use Cases

Search by Name: Company name

 Walmart	 Comcast	 Adobe	 eBay
 The ICIJ - Panama Papers	 Digitate	 Lyft	 Lockheed Martin Space
 U.S. Army	 Dun & Bradstreet	 Global 50 Bank	 German Centre for Diabetes Research
 UBS	 Perform Media	 Airbnb	 Microsoft
 MONSANTO	 IBM	 ICIJ	 novo nordisk



Introduction

- Products & Services

<https://neo4j.com/product/auradb/>

<p>AuraDB Free</p> <p>\$0</p> <p>Learn and Explore Graphs</p> <p>Start Learning</p>	<p>AuraDB Professional</p> <p>\$65 /GB/month (minimum 1GB cluster)</p> <p>Build Production-Ready Apps</p> <p>Try for Free</p>	<p>AuraDB Business Critical</p> <p>\$146 /GB/month (minimum 2GB cluster)</p> <p>Scale Apps for Enterprise Use</p> <p>Sign Up</p>
---	---	--



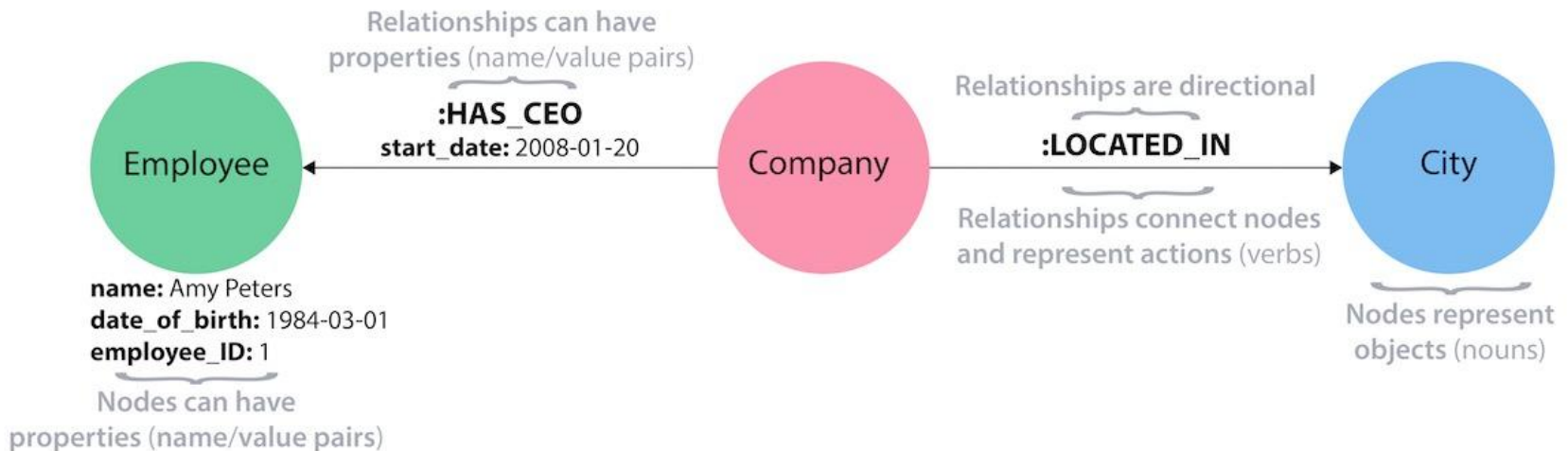
<p> Neo4j Desktop</p> <p>Experience Neo4j on Your Local Desktop. For Free.</p> <p>Download Learn More</p>	<p> Neo4j Browser</p> <p>Explore Your Database and Craft the Perfect Cypher Query.</p> <p>Learn More</p>
---	--

<https://neo4j.com/product/developer-tools/>

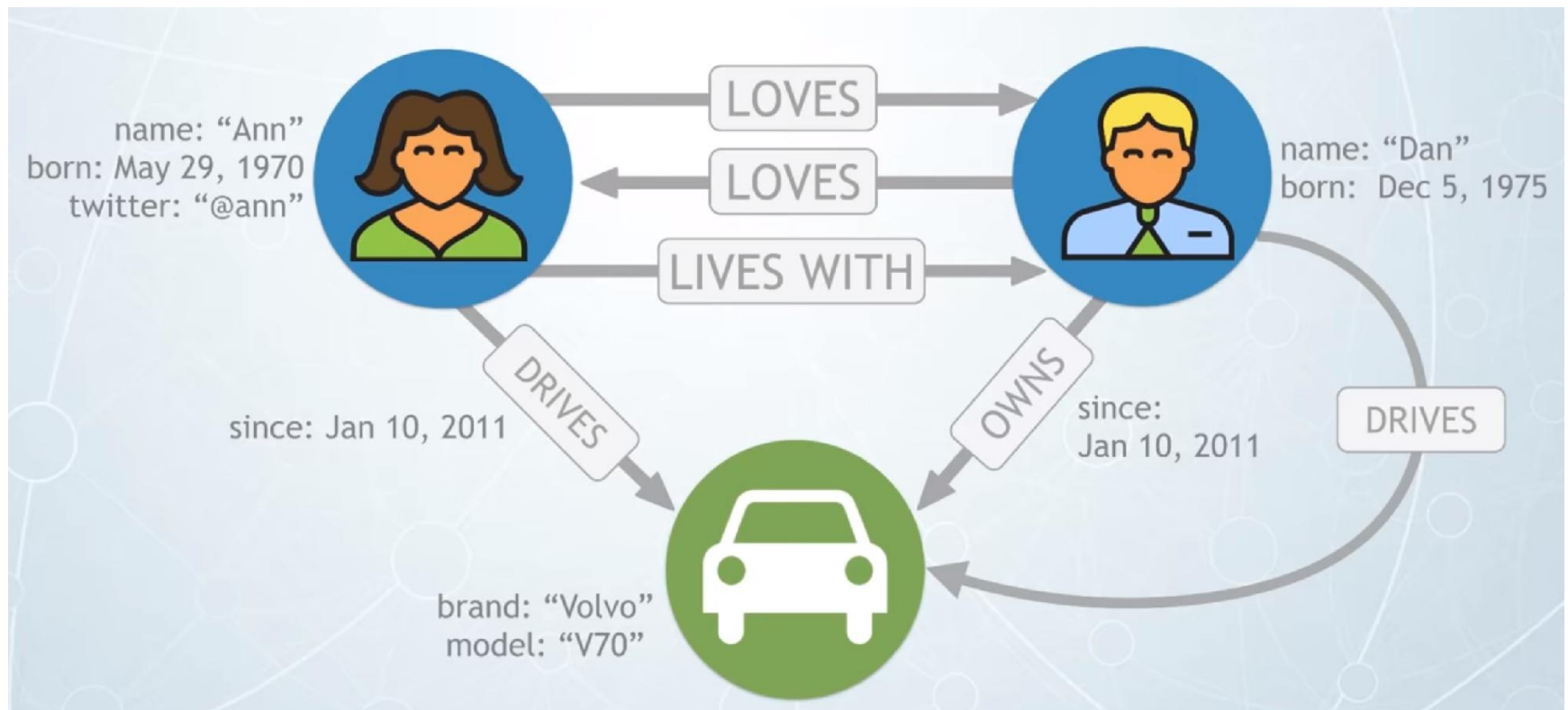
Graph model

- Treat the relationships between data as equally important to the data itself
- Rel. databases compute relationships at query time through JOIN operations
 - A graph database stores connections alongside the data in the model
- Key constituents of the graph model
 - Nodes – entities in the graph
 - Relationships – directed, named connections between two nodes (e.g. Employee *WORKS_FOR* Company)
 - Even if directed, they can be navigated in both directions
 - Properties – key/value pairs that can be attached both to nodes and relationships
 - Labels on nodes – representing types of nodes

Graph model



Graph model



Properties of graph databases

- Criterias to evaluate
 - Intuitiveness
 - Easier to take a journey from requirements and whiteboard to the actual data model
 - graph is natural way of expressing thoughts and though less translations are needed
 - Speed
 - Simpler model, so much quicker from the idea to the deployment
 - Ebay: Neo4j 1000x faster than MySQL based solution with 10-100 less code
 - Agility
 - No schemas, a naturally adaptive model (if sth needed, just add it)
 - Cypher language as a more concise way to express queries
 - What enables quicker understanding and easier way to change

Properties of graph databases

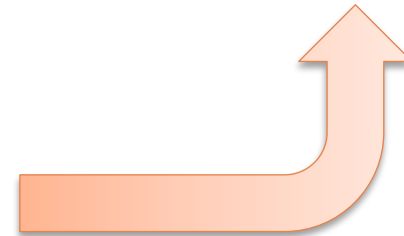
■ Graph & Cypher power*

Typical Complex SQL Join

```
[SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.pid AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT manager.pid AS directReportees, count(manager.directly_manages) AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(L2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT reportee.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT L2Reportees.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT L2Reportees.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
)
) AS T
GROUP BY directReportees)
```

The Same Query using Cypher

```
MATCH (boss)-[:MANAGES*0..3]->(sub),
(sub)-[:MANAGES*1..3]->(report)
WHERE boss.name = "John Doe"
RETURN sub.name AS Subordinate,
count(report) AS Total
```



*Assuming the SQL query is written in the optimal way what probably is not the case

Example database

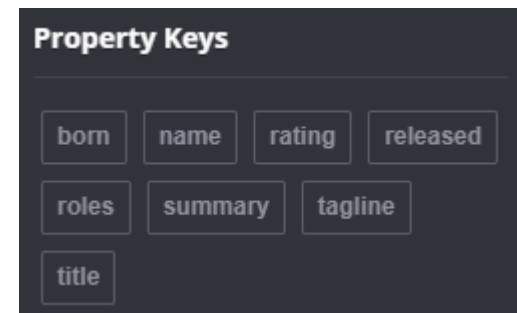
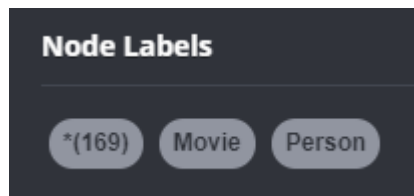
- Movies database

- <https://neo4j.com/docs/getting-started/appendix/example-data/>

- Domain model

```
(:Person {name})-[:ACTED_IN {roles}]->(:Movie {title,released})
```

- What's inside?



- More examples

- <https://github.com/neo4j-examples>

Cypher

- Open language
 - <http://www.opencypher.org/>
- Useful command
 - :help CREATE, :help MATCH, :help STH
- Based on ASCII Art and patterns
- Basic pattern
 - ()-[[:RELATIONSHIP]]->()
 - Node – REL → Node
- Query
 - MATCH pattern
 - WHERE conditions
 - RETURN result

Cypher

- AsciiArt for nodes
 - Nodes
 - `()` or `(p)`
 - Labels, tags
 - `(p:Person:Mammal)`
 - Properties
 - `(p:Person { name: 'John'})`
- AsciiArt for relationship
 - Relationship
 - `--> -[a:ACTED_IN]->`
 - Direction
 - `(p1) -[:ACTED_IN]-> (p2)` `(p1) <-[:ACTED_IN]- (p2)`
 - Properties
 - `(p1) -[a:ACTED_IN { type: 'series' }]-> (p2)`
- Aliases
 - `p1, p2, a`

Cypher

■ Actors

- Marek Kondrat (MK)
 - Ur. 18.10.1950
- Piotr Fronczewski (PF)
 - Ur. 8.6.1946
- Krzysztof Kowalewski (KK)
 - Ur. 20.03.1937
- Janusz Gajos (JG)
 - Ur. 23.09.1939
- Zbigniew Zapasiewicz (ZZ)
 - Ur. 13.09.1934, Zm. 14.07.2009

■ Movies

- Psy (Psy)
- C.K. Dezerterzy (CKD)
- Dzień świra (DS)
- Miś (Miś)
- Akademia Pana Kleksa (APK)

■ Acted in

- Psy
 - MK (Olo)
 - ZZ (Wencel)
 - JG (Siwy)
- CKD
 - MK (Kania)
 - KK (boss)
 - ZZ (Wagner)
- DS
 - MK (Adaś Miauczyński)
 - PF (doctor)
- Miś
 - KK (Jan Hochwander)
- APK
 - PF (Ambroży Kleks)

Cypher

- **CRUD Examples**

```
CREATE (:Person { name: "Marek Kondrat" }) -[:ACTED_IN]-> (:Movie { name: "Psy" })
CREATE (:Person { name: "Janusz Gajos", born: "23.09.1939" })
MATCH (:Movie { name: "Psy" })<-[:ACTED_IN]-(:Person { name: "Marek Kondrat"}) SET a.character="Olo"
RETURN a
```

```
MATCH (p:Person),(m:Movie)
WHERE p.name = 'Janusz Gajos' AND m.name = 'Psy'
CREATE (p)-[:ACTED_IN { character: "Siwy" }]->(m)
RETURN type(r), r.character
```

```
MATCH (p:Person {name: "Janusz Gajos"}), (m:Movie {title: "Psy"})
MERGE (p)-[:ACTED_IN { character: "Siwy" }]->(m)
RETURN p.name, type(r), m.title
```

```
MATCH (:Movie { name: "Psy" })<-[:ACTED_IN]-(:Person) RETURN p
MATCH (m:Movie)<-[:ACTED_IN]-(:Person) WHERE m.name="Psy" RETURN *
```

```
MATCH (n:Person { name: 'UNKNOWN' })
DELETE n
```

```
MATCH (n)
DETACH DELETE n
```

Więcej: <https://neo4j.com/docs/cypher-manual/current/clauses/>

Cyber

- Aura console
 - <https://console-preview.neo4j.io/>
- Further learning
 - <https://graphacademy.neo4j.com/>
 - <https://neo4j.com/docs/cypher-manual/current/introduction/>

Others

- Potentially useful
 - <https://neo4j.com/docs/operations-manual/current/configuration/password-and-user-recovery/>
- Additional resources
 - https://medium.com/@arminpezo_62757/meet-neo4j-step-by-step-guide-to-graph-database-3c182dd7cc42
 - <https://github.com/neo4j-examples/movies-dotnetcore-bolt>
 - https://medium.com/@arminpezo_62757/integration-of-net-application-with-neo4j-database-cffa1528929c

Apache Cassandra

Introduction

- Apache Cassandra is an open-source, NoSQL, column-oriented database
- Focus on scalability, availability, fault-tolerance, replication support
 - Less focus on data integrity, no JOINS
- Initially created at Facebook for inbox search
 - Open source at 2008, started in Apache Incubator in 2009 and top level project at Apache from 2010
- CQL as query language (similar to SQL)

Key Concepts

- Keyspace
 - Defines how a dataset is replicated, per datacenter.
 - Replication is the number of copies saved per cluster.
 - Keyspaces contain tables
- Table
 - Tables are partitioned based on the columns provided in the partition key
 - Cassandra tables can flexibly add new columns to tables with zero downtime
- Partition
 - Defines the mandatory part of the primary key for all rows
 - Identify the node in a cluster where the row is stored
 - All performant queries supply the partition key in the query
- Rows & Columns

Getting started

- Starting the server
 - `docker pull cassandra:latest`
 - `docker network create cassandra`
 - `docker run --rm -d --name cassandra --hostname cassandra --network cassandra cassandra`

Getting started

- Pushing sample data and launching client
 - `docker run --rm --network cassandra -v "$(pwd)/data.cql:/scripts/data.cql" -e CQLSH_HOST=cassandra -e CQLSH_PORT=9042 -e CQLVERSION="3.4.7" nuvo/docker-cqlsh`
 - Remark: in case of connectivity problem, execute as a prerequisite the next command
 - `docker run --rm -it --network cassandra nuvo/docker-cqlsh cqlsh cassandra 9042 --cqlversion="3.4.7"`

Getting started

- First queries:
 - `SELECT * FROM store.shopping_cart;`
 - `INSERT INTO store.shopping_cart (userid, item_count) VALUES ('4567', 20);`

The Cassandra Query Language (CQL)

- DDL
 - CREATE KEYSPACE
 - USE (change keyspace)
 - ALTER KEYSPACE
 - DROP KEYSPACE
 - CREATE TABLE
 - ALTER TABLE
 - DROP TABLE
 - TRUNCATE TABLE

The Cassandra Query Language (CQL)

- DML
 - SELECT
 - INSERT
 - UPDATE
 - DELETE
 - BATCH

