



# Lecture Outline

## Graph databases

- Introduction

## Neo4j

- Data model: **property graphs**
- **Traversal framework**
- **Cypher** query language
  - Read, write, and general clauses

# Neo4j Graph Database



# Sample Data

## Sample graph with **movies** and **actors**

(m1:MOVIE { id: "vratnelahve", title: "Vratné lahve", year: 2006 })

(m2:MOVIE { id: "samotari", title: "Samotáři", year: 2000 })

(m3:MOVIE { id: "medvidek", title: "Medvídek", year: 2007 })

(m4:MOVIE { id: "stesti", title: "Štěstí", year: 2005 })

(a1:ACTOR { id: "trojan", name: "Ivan Trojan", year: 1964 })

(a2:ACTOR { id: "machacek", name: "Jiří Macháček", year: 1966 })

(a3:ACTOR { id: "schneiderova", name: "Jitka Schneiderová", year: 1973 })

(a4:ACTOR { id: "sverak", name: "Zdeněk Svěrák", year: 1936 })

(m1)-[c1:PLAY { role: "Robert Landa" }]->(a2)

(m1)-[c2:PLAY { role: "Josef Tkaloun" }]->(a4)

(m2)-[c3:PLAY { role: "Ondřej" }]->(a1)

(m2)-[c4:PLAY { role: "Jakub" }]->(a2)

(m2)-[c5:PLAY { role: "Hanka" }]->(a3)

(m3)-[c6:PLAY { role: "Ivan" }]->(a1)

(m3)-[c7:PLAY { role: "Jirka", award: "Czech Lion" }]->(a2)

(a1)-[f1:KNOW]->(a2)

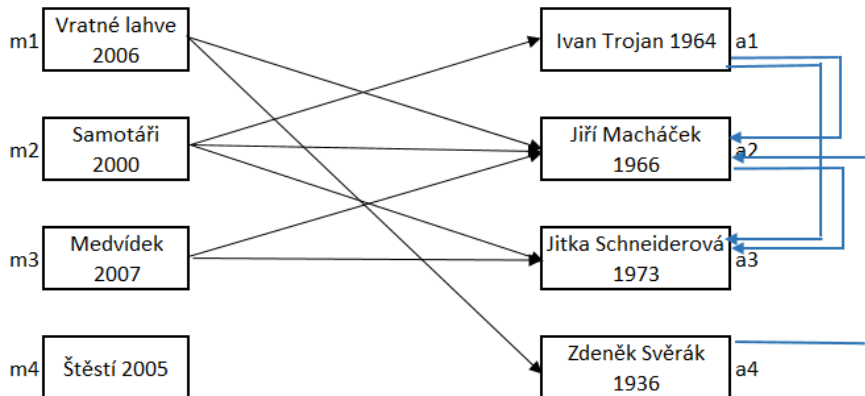
(a1)-[f2:KNOW]->(a3)

(a2)-[f3:KNOW]->(a3)

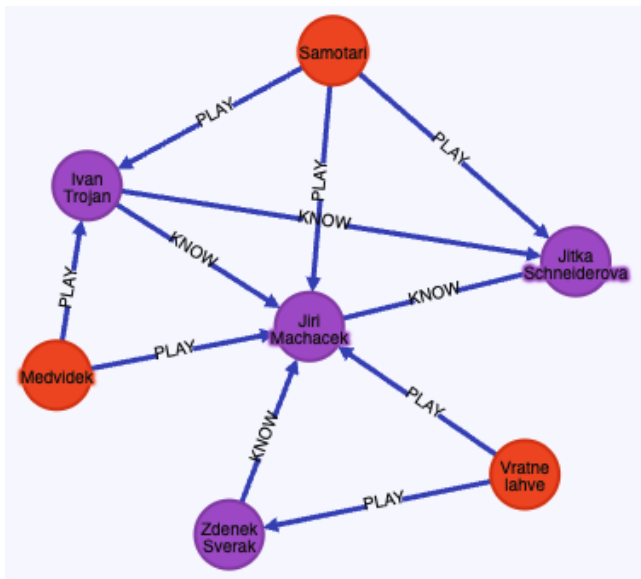
(a4)-[f4:KNOW]->(a2)

# Sample Data

Sample graph with **movies** and **actors**



# Sample Data (<https://console.neo4j.org>)



**Cypher**

# Cypher

## Cypher

- Declarative **graph query language**
  - Allows for expressive and efficient querying and updates
  - Inspired by SQL (query clauses) and SPARQL (pattern matching)
- **OpenCypher**
  - Ongoing project aiming at Cypher standardization
  - <http://www.opencypher.org/>

## Clauses

- E.g. MATCH, RETURN, CREATE, ...
- Clauses can be (almost arbitrarily) **chained together**
  - Intermediate result of one clause is passed to a subsequent one

# Sample Query

Find names of actors who played in *Medvídek* movie

```
MATCH(m:MOVIE)-[r:PLAY]->(a:ACTOR)
  WHERE m.title = "Medvídek"
RETURN a.name, a.year
ORDER BY a.year
```

a.name	a.year
Ivan Trojan	1964
Jiří Macháček	1966



# Clauses

## Read clauses and their sub-clauses

- MATCH – specifies graph patterns to be searched for
  - WHERE – adds additional filtering constraints
- ...

## Write clauses and their sub-clauses

- CREATE – creates new nodes or relationships
- DELETE – deletes nodes or relationships
- SET – updates labels or properties
- REMOVE – removes labels or properties
- ...

# Clauses

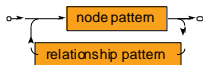
## General clauses and their sub-clauses

- `RETURN` – defines what the query result should contain
  - `ORDER BY` – describes how the query result should be ordered
  - `SKIP` – excludes certain number of solutions from the result
  - `LIMIT` – limits the number of solutions to be included
- `WITH` – allows query parts to be chained together
- ...

# Path Patterns

## Path pattern expression

- **Sequence of interleaved node and relationship patterns**
- Describes a single path (not a general subgraph)



- ASCII-Art inspired syntax
  - Circles ( ) for nodes
  - Arrows <--, --, --> for relationships

```
(m:MOVIE)-[r:PLAY]->(a:ACTOR)
(a:ACTOR {name: "Jiří Macháček"})-[r:PLAY]->(m:MOVIE)
```

# Path Patterns

## Node pattern

- Matches one data node

```
(a:ACTOR)  
(m:MOVIE {title: "Samotáři"})
```

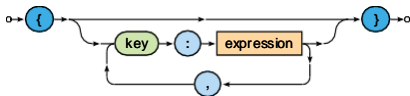


- Variable**
  - Allows us to access a given node later on
- Set of **labels**
  - Data node must have **all the specified labels** to be matched  
MATCH (n:PERSON:ACTOR)
  - Labels are case sensitive
- Property map**
  - Data node must have **all the requested properties** (including their values) to be matched (the order is unimportant)

# Path Patterns

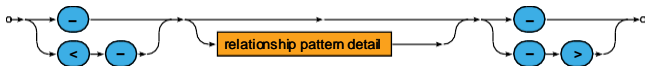
## Property map

```
(m:MOVIE {title: "Vratné lahve"})
```



## Relationship pattern

- Matches one data relationship



```
-[r:PLAY]->
```

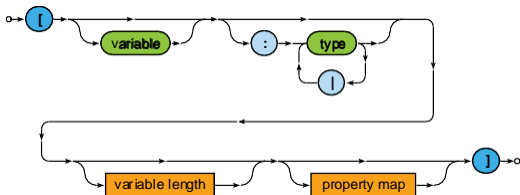
```
-[r:PLAY {award: "Czech Lion"}]->
```

# Path Patterns

-[r:PLAY]->

-[r:PLAY {award: "Czech Lion"}]->

Relationship pattern



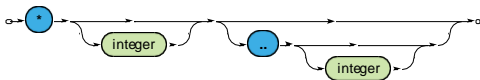
- **Variable**
  - Allows us to access a given node later on
- **Set of types**
  - Data relationship must be of **one of the enumerated types** to be matched

(a)-[r:TYPE1 | TYPE2]->(b)

# Path Patterns

## Relationship pattern (*cont.*)

- **Property map**
  - Data relationship must have **all the requested properties**
- Variable path **length**
  - Allows us to match **paths of arbitrary lengths** (not just exactly one relationship)



- Examples:  $*$ ,  $*_4$ ,  $*_{2..6}$ ,  $*_{..6}$ ,  $*_{2..}$

(a)-[r\*1..3]->(b)

# Path Patterns

## Examples

```
()
```

```
(x)--(y)
```

```
(m:MOVIE)-->(a:ACTOR)
```

```
(:MOVIE)-->(a { name: "Ivan Trojan" })
```

```
()<-[r:PLAY]-()
```

```
(m)-[:PLAY { role: "Ivan" }]->()
```

```
(:ACTOR { name: "Ivan Trojan" })-[:KNOW *2]->(:ACTOR)
```

```
()-[:KNOW *5..]->(f)
```

# Path Patterns

Example:

- are these queries equivalent?

```
(a1:ACTOR {name: "Ivan Trojan"})-[:PLAY]->(m:MOVIE)<-[:PLAY]-(a2:ACTOR)
```

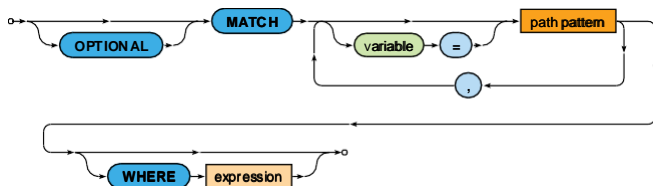
```
(a1:ACTOR {name: "Ivan Trojan"})<-[:PLAY]-(m:MOVIE)-[:PLAY]->(a2:ACTOR)
```

```
(a1:ACTOR {name: "Ivan Trojan"})-[:PLAY]-(m:MOVIE)-[:PLAY]-(a2:ACTOR)
```

# Match Clause

## MATCH clause

- Allows to search for **sub-graphs of the data graph** that match the provided path pattern/patterns (all of them)
  - **Query result** (table) = unordered **set of solutions**
  - One solution (row) = set of **variable bindings**
- Each variable has to be bound



```
MATCH p = (a:ACTOR)-[:PLAY]->(m:MOVIE {title: 'Medvídek'})  
RETURN p;
```

# Match Clause

**WHERE sub-clause** may provide additional constraints

- These constraints are **evaluated directly during the matching phase** (i.e. not after it)
- Typical usage
  - Boolean
  - expressions
  - Comparisons
  - Path patterns – `true` if at least one solution is found
  - ...

# Match Clause: Example

Find names of actors who played with *Ivan Trojan* in any movie

```
MATCH(i:ACTOR)-[:PLAY]-(m:MOVIE)-[:PLAY]->(a:ACTOR)
  WHERE (i.name = "Ivan Trojan")
RETURN a.name
```

```
MATCH(i:ACTOR { name: "Ivan Trojan" })
  <-[:PLAY]-(m:MOVIE)-[:PLAY]->
  (a:ACTOR)
RETURN a.name
```

i	m	a		a.name
(a1)	(m2)	(a2)	⇒	Jiří Macháček
(a1)	(m2)	(a3)		Jitka Schneiderová
(a1)	(m3)	(a2)		Jiří Macháček

The second query might be slightly more efficient because it can use an index on the name property of ACTOR nodes (if such an index exists) from the start of the query execution

# Match Clause

## Uniqueness requirement

- One data node may match several query nodes, but one data relationship may not match several query relationships

```
MATCH (a:ACTOR)-[r:KNOWS]->(b: ACTOR)-[s:KNOWS]->(c: ACTOR)
```

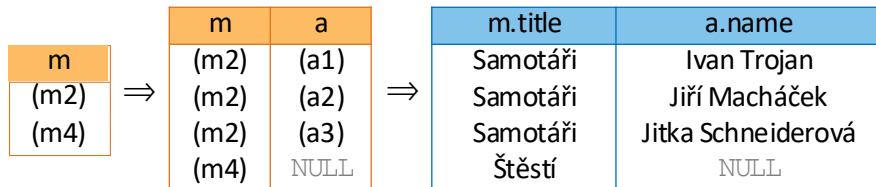
## OPTIONAL MATCH

- Attempts to find matching data sub-graphs as usual...
- but **when no solution is found**, one specific solution with **all the variables bound to NULL** is generated
- Note that either the whole pattern is matched, or nothing is matched

# Match Clause: Example

Find movies filmed in 2005 or earlier and names of their actors (if any)

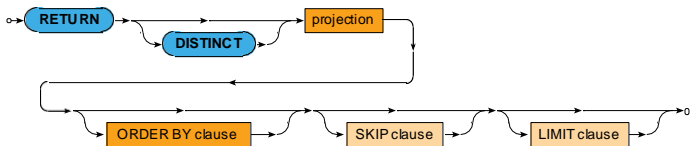
```
MATCH(m:MOVIE)
  WHERE (m.year <= 2005)
OPTIONAL MATCH(m)-[:PLAY]->(a:ACTOR)
RETURN m.title, a.name
```



# Return Clause

## RETURN clause

- Defines what to include in the query result
  - Projection of variables, properties of nodes or relationships (via dot notation), aggregation functions, ...
- Optional ORDER BY, SKIP and LIMIT sub-clauses



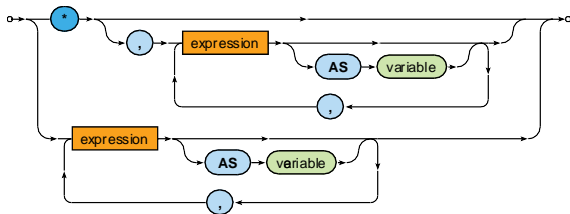
## RETURN DISTINCT

- Duplicate solutions (rows) are removed

# Return Clause

## Projection

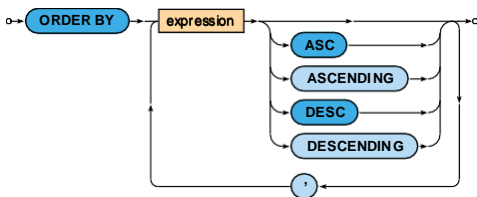
- \* = **all the variables**
  - Can only be specified as the very first item
- AS allows to **explicitly (re)name** output records



# Return Clause

## ORDER BY sub-clause

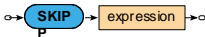
- Defines the **order of solutions** within the query result
  - Multiple criteria can be specified
  - Default direction is ASC
- The order is undefined unless explicitly defined
- Nodes and relationships as such cannot be used as criteria



# Return Clause

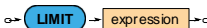
## SKIP sub-clause

- Determines the **number of solutions to be skipped** in the query result



## LIMIT sub-clause

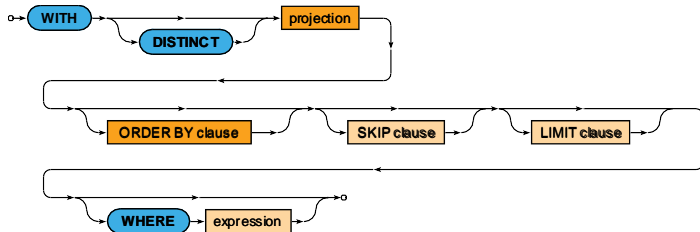
- Determines the **number of solutions to be included** in the query result



# With Clause

## WITH clause

- **Constructs intermediate result**
  - Analogous behavior to the `RETURN` clause
  - Does not output anything to the user, just **forwards the current result to the subsequent clause**
- Optional `WHERE` sub-clause can also be provided



# With Clause: Example

Numbers of movies in which actors born in 1965 or later played

```
MATCH(a:ACTOR)
  WHERE (a.year >= 1965)
WITH a, [(a)-[:PLAY]-(movie:MOVIE) | movie] AS moviesList
RETURN a.name,
       SIZE(moviesList) AS movies
ORDER BY movies ASC;
```

The created list  
can contain 0

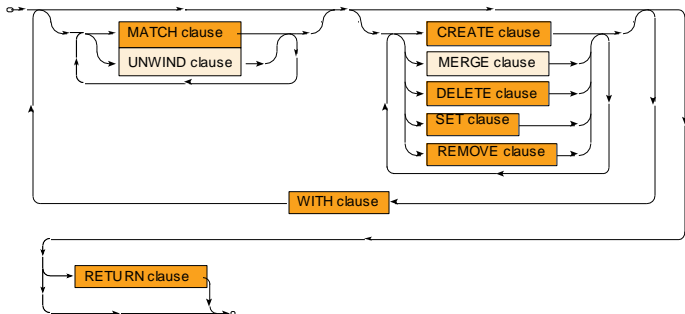
```
MATCH(a:ACTOR)
  WHERE (a.year >= 1965)
MATCH (a)-[:PLAY]-(m:MOVIE)
WITH a, COUNT(m) AS movies
RETURN a.name, movies
ORDER BY movies ASC;
```

COUNT omits actors  
without movies

<table><thead><tr><th>a</th></tr></thead><tbody><tr><td>(a2)</td></tr><tr><td>(a3)</td></tr></tbody></table>	a	(a2)	(a3)	⇒	<table><thead><tr><th>a</th><th>movies</th></tr></thead><tbody><tr><td>(a2)</td><td>3</td></tr><tr><td>(a3)</td><td>1</td></tr></tbody></table>	a	movies	(a2)	3	(a3)	1	⇒	<table><thead><tr><th>a.name</th><th>movies</th></tr></thead><tbody><tr><td>Jitka Schneiderová</td><td>1</td></tr><tr><td>Jiří Macháček</td><td>3</td></tr></tbody></table>	a.name	movies	Jitka Schneiderová	1	Jiří Macháček	3
a																			
(a2)																			
(a3)																			
a	movies																		
(a2)	3																		
(a3)	1																		
a.name	movies																		
Jitka Schneiderová	1																		
Jiří Macháček	3																		

# Query Structure

## Chaining of Cypher clauses (*simplified*)



- **Read** clauses: MATCH, ...
- **Write** clauses: CREATE, DELETE, SET, REMOVE, ...

# Query Structure

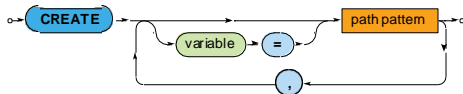
## Query parts

- **WITH clauses split the whole query into query parts**
- Certain restrictions apply...
  - **Read clauses (if any) must precede write clauses (if any) in every query part**
  - **The last query part must be terminated by a RETURN clause**
    - Unless this part contains at least one write clause
  - – I.e. **read-only queries must return data**
- ...

# Write Clauses

## CREATE clause

- Inserts new nodes or relationships into the data graph



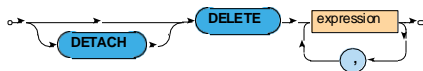
## Example

```
MATCH(m:MOVIE { id: "stesti"})
CREATE
  (a:ACTOR { id: "vilhelmova", name: "Tatiana Vilhelmová", year: 1978}),
  (m)-[:PLAY]->(a)
```

# Write Clauses

## DELETE clause

- **Removes nodes, relationships or paths** from the data graph
- Relationships must always be removed before the nodes they are associated with
  - Unless the `DETACH` modifier is specified



## Example

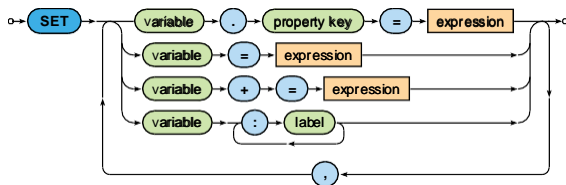
```
MATCH(:MOVIE { id: "stesti"})-[r:PLAY]->(a:ACTOR)
DELETE r
```

# Write Clauses

```
MATCH (a:ACTOR { name: "Ivan Trojan" })  
SET a += { year: 1964 }  
RETURN a
```

## SET clause

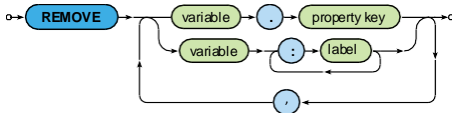
- Allows to...
  - **set a value of a particular property**
    - or remove a property when `NULL` is assigned
  - **replace properties** (all of them) with new ones
  - **add new properties** to the existing ones
  - **add labels** to nodes
- Cannot be used to set relationship types



# Write Clauses

## REMOVE clause

- Allows to...
  - **remove a particular property**
  - **remove labels** from nodes
- Cannot be used to remove relationship types



```
MATCH (a:ACTOR { name: "Ivan Trojan" })  
REMOVE a.year  
RETURN a
```

# Expressions

## Literal expressions

- Integers: decimal, octal, hexadecimal
- Floating-point numbers
- Strings
  - Enclosed in double or single quotes
  - Standard escape sequences
- Boolean values: `true`, `false`
- `NULL` value (cannot be stored in data graphs)

## Other expressions

- Collections, variables, property accessors, function calls, path patterns, boolean expressions, arithmetic expressions, comparisons, regular expressions, predicates, ...

# The shortest path

Find the shortest path between Ivan Trojan and Jitka Schneiderova

```
MATCH p = shortestPath((a1:ACTOR {name: "Ivan Trojan"})  
                        -[*]-(a2:ACTOR {name: "Jitka Schneiderova"}))  
RETURN p;
```

```
MATCH p = shortestPath((a1:ACTOR {name: "Ivan Trojan"})  
                        -[r*]-(a2:ACTOR {name: "Jitka Schneiderova"}))  
RETURN [node in nodes(p) | node.name] as names,  
       [rel in relationships(p) | type(rel)] as relations;
```

names: ["Ivan Trojan", "Jitka Schneiderova"] relations: ["KNOW"]



# Lecture Conclusion

**Neo4j** = graph database

- **Property graphs**
- **Traversal framework**
  - Path expanders, uniqueness, evaluators, traverser

**Cypher** = graph query language

- Read (sub-)clauses: MATCH, WHERE, ...
- Write (sub-)clauses: CREATE, DELETE, SET, REMOVE, ...
- General (sub-)clauses: RETURN, WITH, ORDER BY, LIMIT, ...