Modelowanie nierelacyjnych baz danych

Bazy grafowe

Bazy grafowe

Model danych

- Graf
  - **Skierowany / nieskierowane**, para
    - węzłów (wierzchołków) reprezentujących obiekty świata rzeczywistego
    - relacji (krawędzi) pomiędzy tymi węzłami
  - Zarówno węzły, jak i relacje można powiązać z dodatkowymi właściwościami

Typy baz danych

- **Bez transakcji** = mała liczba dużych grafów
- **Transakcyjne** = duża liczba małych grafów

Bazy grafowe

Struktura zapytań:

- Utwórz, zaktualizuj lub usuń węzeł/relację w grafie
- **Algorytmy grafowe** (najkrótsze ścieżki, drzewa opinające,…)
- Ogólne **przejścia po grafie**
- Zapytania o **podgrafy** lub zapytania o **supergrafy**
- Zapytania oparte na **podobieństwie** (przybliżone dopasowanie)

Neo4j

Baza NoSQL typu grafowego

- https://neo4j.com/
- Właściwości
  - Open source, duża skalowalność (miliardy węzłów), wysoka dostępność, odporność na błędy, replikacja master-slave, wsparcie dla **transakcji ACID**, posiada możliwość osadzania,

    Ekspresyjny język zapytań grafowych (**Cypher**),

    Moduł dedykowany do przechodzenia po grafie
- Stworzony przez **Neo Technology**
- Zaprogramowany w języku Java
- Systemy operacyjne: wieloplatformowe
- Pierwsze wydanie dostępne w 2007

Model danych

Struktura systemu bazy danych

Instancja→ *skierowany* **graf**

Graf właściwości = skierowany graf z etykietami

- Zbiór wierzchołków (węzłów) i krawędzi (relacji)

Węzeł **grafu:**

- Posiada unikalny (wewnętrzny) **identyfikator**
- Może zostać powiązany z **zestawem etykiet**
  - Umożliwia kategoryzację wierzchołków
- Można go również powiązać ze zbiorem **właściwości**
  - Pozwala to przechowywać dodatkowe dane wraz z węzłami

Model danych

Relacja w grafie

- Posiada unikalny (wewnętrzny) **identyfikator**
- Ma określony **kierunek**
  - Relacje można przeszukiwać tak samo efektywnie, bez względu na kierunek.
  - Kierunek można w ogóle zignorować podczas zapytań
- Zawsze ma węzeł **startowy** i **węzeł końcowy**
  - Może być rekursywna (tj. dozwolone są też pętle)
- Jest powiązana **dokładnie z jednym typem**
- Może być również powiązana z **zestawem właściwości**

Model danych

Właściwości wierzchołków i relacji

- **Para klucz-wartość**
  - Klucz jest ciągiem znaków
  - Wartość jest **dowolną daną** typu prostego, lub **tablicą wartości** typu prostego

Proste typy danych

- boolean – **wartości logiczne:** true, false
- byte, short, int, long – **liczby całkowite** (1B, 2B, 4B, 8B)
- float, double – **liczby zmiennoprzecinkowe** (4B, 8B)
- char – jeden znak kodowany w **Unicode**
- string – sekwencja znaków **Unicode**

Przykładowe dane

Przykładowe dane dotyczące **filmów i aktorów**

```
(m1:MOVIE { id: "niezniszczalni", title: "Niezniszczalni", year: 2012 })
(m2:MOVIE { id: "tombrider", title: "Tomb Raider", year: 2001 })
(m3:MOVIE { id: "niezniszczalni3", title: " Niezniszczalni 3", year: 2014 })
(m4:MOVIE { id: "wyscigsmierci", title: "Wyścig śmierci", year: 2008 })
(m5:MOVIE { id: "panipanismith", title: "Pan i Pani Smith", year: 2005 })

(a1:ACTOR { id: "stallone", name: "Sylwester Stallone", year: 1946 })
(a2:ACTOR { id: "statham", name: "Jason Statham", year: 1967 })
(a3:ACTOR { id: "martinez", name: "Natalie Martinez", year: 1984 })
(a4:ACTOR { id: "jolie", name: "Angelina Jolie", year: 1975 })

(m1)-[c1:PLAY { role: "Barney Ross" }]->(a1)
(m1)-[c2:PLAY { role: "Lee Christmas" }]->(a2)
(m2)-[c3:PLAY { role: "Lara Croft" }]->(a4)
(m3)-[c4:PLAY { role: " Barney Ross" }]->(a1)
(m3)-[c5:PLAY { role: " Lee Christmas" }]->(a2)
(m4)-[c6:PLAY { role: "Jensen Ames", award: "Best Actor Award"}]->(a2)
(m4)-[c7:PLAY { role: "Case "}]->(a3)
```

Interfejs do Neo4j

Architektura bazy danych

- Klient-serwer
- **Wbudowana baza**
  - Bezpośrednio zintegrowana z Twoją aplikacją

Neo4j - podłączenie do języków programowania

- Oficjalne: Java, .NET, JavaScript, Python
- Społeczność: C, C++, PHP, Ruby, Perl, R itp.

Neo4j shell

- Interaktywny wiersz poleceń

Zapytania

- **Cypher** – deklaratywny język zapytań do grafów
- **Traversal framework**

Przechodzenie po grafie

Traversal framework

- Pozwala wyrażać i wykonywać zapytania dotyczące przechodzenia przez graf
- Działa wykorzystując wywołania zwrotne (ang. *callbacks*)

Opis przejścia

- **Definiuje reguły i inne własności przechodzenia przez graf**

Traverser

- Inicjuje i **zarządza określonym sposobem przejścia przez graf** ze względu na…
  - Dostarczony opis przejścia oraz
  - Węzła grafu/ zestawu węzłów w którym rozpoczyna się przejście
- Umożliwia **iteracje po pasujących ścieżkach**, jedna po drugiej

Przechodzenie po grafie – przykład

Znajdź aktorów, którzy grali w filmie *Niezniszczalni*

```
TraversalDescription td = db.traversalDescription()
    .breadthFirst()
    .relationships(Types.PLAY, Direction.OUTGOING)
    .evaluator(Evaluators.atDepth(1));

Node s = db.findNode(Label.label("MOVIE"), "id", "niezniszczalni");
Traverser t = td.traverse(s);

for (Path p : t) {
    Node n = p.endNode();
    System.out.println(
        n.getProperty("name")
    );
}
```

Sylwester Stallone
Jason Statham

Przechodzenie po grafie

Elementy składowe **opisu przejścia**

- **Kierunek**
  - Określa niejawnie jaki algorytm przejścia zastosować
- **Ekspander**
  - Określa jakie relacje powinny być brane pod uwagę
- **Unikalność**
  - Czy węzły bądź relacje można odwiedzać wielokrotnie
- **Ewaluacja**
  - Kiedy proces przejścia powinien zostać zakończony
  - Co powinno zostać dołączone do wyniku zapytania

Przechodzenie: kolejność

Kolejność

Jaki algorytm przejścia zastosować?

- Standardowe metody **wgłąb** lub **wszerz** mogą zostać wybrane
- Można również wdrożyć specyficzne zasady uporządkowania gałęzi
- Użycie:
  - td.**breadthFirst**(), td.**depthFirst**()

Przechodzenie: rozszerzenie

Ekspander ścieżek (ang. *Path expanders*)

Będąc w danym węźle…

które relacje należy następnie odwiedzić?

- **Ekspander określa jeden dozwolony…**
  typ relacji oraz **kierunek:**
  - Direction.**INCOMING**
  - Direction.**OUTGOING**
  - Direction.**BOTH**
- Można określić wiele ekspanderów jednocześnie
  - Gdy żaden ekspander nie został jawnie określony, wszystkie relacje stają się dozwolone
- Użycie:
  td.**relationships**(type, direction)

Przechodzenie: unikalność

Unikalność

Czy można ponownie odwiedzić poszczególne węzły/relacje?

- Dostępne są różne **poziomy unikalności:**

  Uniqueness.**NONE**– żaden filtr nie jest nałożony

  Uniqueness.**RELATIONSHIP_PATH**
  Uniqueness.**NODE_PATH**
  - Węzły/relacje w ramach bieżącej ścieżki muszą być unikalne

  Uniqueness.**RELATIONSHIP_GLOBAL** Uniqueness.**NODE_GLOBAL** (**default**)
  - Żaden węzeł/relacja nie może być odwiedzana więcej niż raz

- Użycie: td.**uniqueness**(level)

Przechodzenie: ewaluacja

Ewaluator

Biorąc pod uwagę konkretną ścieżkę:

- *czy ta ścieżka powinna zostać uwzględniona w wyniku?*
- *czy przeprawa powinna być kontynuowana?*

- Dostępne **akcje przy ewaluacji**

  Evaluation.**INCLUDE_AND_CONTINUE**

  Evaluation.**INCLUDE_AND_PRUNE**

  Evaluation.**EXCLUDE_AND_CONTINUE**

  Evaluation.**EXCLUDE_AND_PRUNE**

- Znacznie powyższych akcji:
  - INCLUDE/ EXCLUDE= czy uwzględnić ścieżkę w wyniku
  - CONTINUE/ PRUNE= czy kontynuować przeprawę

Przechodzenie: ewaluacja

Predefiniowane ewaluatory

- Evaluators.**all**()
  - Nigdy nie przycina, uwzględnia wszystko
- Evaluators.**excludeStartPosition**()
  - Nigdy nie przycina, obejmuje wszystko oprócz węzłów początkowych
- Evaluators.**atDepth**(depth)
  Evaluators.**toDepth**(maxDepth)
  Evaluators.**fromDepth**(minDepth)
  Evaluators.**includingDepths**(minDepth, maxDepth)
  - Obejmuje tylko pozycje w określonym przedziale głębokości

Przechodzenie: ewaluacja

Ewaluatorzy

- Użycie:
  td.**evaluator**(evaluator)
- Należy pamiętać, że ewaluatory są stosowane **nawet dla węzłów początkowych!**
- Gdy dostępnych jest **wiele ewaluatorów:**
  - Wszystkie muszą osiągnąć konsensus
- Gdy **żaden ewaluator** nie został podany:
  - Przejście nigdy niczego nie przycina i obejmuje wszystko

Trawersowanie

Traverser

- Pozwala nam wykonać konkretny traversal grafu
  w odniesieniu do danego opisu przejścia
  zaczynając od danego węzła / węzłów
- Użycie: t = td.**traverse**(node, ...)
  - for(Pathp:t){...}
    - Iteruje po wszystkich ścieżkach
  - for(Noden:t.nodes()){...}
    - Iteruje po wszystkich ścieżkach, zwraca ich węzły końcowe
  - for(Relationshipr:t.relationships()){...}
    - Iteruje po wszystkich ścieżkach, zwraca ich ostatnie relacje

Ścieżka

- Dobrze uformowana **sekwencja** przeplatanych **węzłów** i **relacji**

Przechodzenie – przykład

Znajdź aktorów, którzy grali z *Natalie Martinez*

```
TraversalDescription td = db.traversalDescription()
    .depthFirst()
    .uniqueness(Uniqueness.NODE_GLOBAL)
    .relationships(Types.PLAY)
    .evaluator(Evaluators.atDepth(2))
    .evaluator(Evaluators.excludeStartPosition());

Node s = db.findNode(Label.label("ACTOR"), "id", "martinez");
Traverser t = td.traverse(s);

for (Node n : t.nodes()) {
    System.out.println(
        n.getProperty("name")
    );
}
```

Jason Statham

Cypher

Cypher

- Deklaratywny **język zapytań** dla grafu
  - Umożliwia ekspresyjne oraz wydajne zapytania i aktualizacje
  - Zainspirowany przez SQL (klauzule zapytań) i SPARQL (dopasowywanie wzorców)
- **OpenCypher**
  - Trwający projekt mający na celu standaryzację Cyphera

    http://www.opencypher.org/

Klauzule

- Na przykład - MATCH, RETURN, CREATE,…
- Klauzule mogą być (niemal arbitralnie) **łączone ze sobą**
  - Wynik pośredni jednej klauzuli jest przekazywany do kolejnej

Przykładowe zapytanie

Znajdź nazwiska aktorów i rok ur., którzy grali w filmie Niezniszczalni

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

a.name	a.year
Sylwester Stallone	1946
Jason Statham	1967

Klauzule

Klauzule odczytujące dane:

- MATCH– określa wzorce w grafie, które mają być wyszukiwane
  - WHERE– dodaje dodatkowe ograniczenia filtrowania

Klauzule związane z zapisem danych:

- CREATE– tworzy nowe węzły lub relacje
- DELETE– usuwa węzły lub relacje
- SET– aktualizuje etykiety lub właściwości
- REMOVE– usuwa etykiety lub właściwości

Klauzule

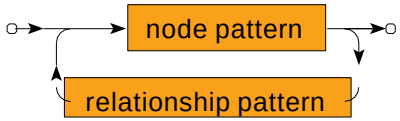
Klauzule ogólne i ich pod-klauzule

- RETURN– definiuje, co powinien zawierać wynik zapytania
  - ORDER BY– opisuje, jak powinien być uporządkowany wynik zapytania
  - SKIP – wyklucza z wyniku określoną liczbę rozwiązań
  - LIMIT– ogranicza liczbę rozwiązań, które mają zostać uwzględnione
- WITH– umożliwia łączenie ze sobą części zapytań

Ścieżki

Wyrażenie **wzorca ścieżki**

- **Sekwencja przeplatanych wzorców węzłów i relacji**
- Opisuje pojedynczą ścieżkę (nie jest to podgraf)

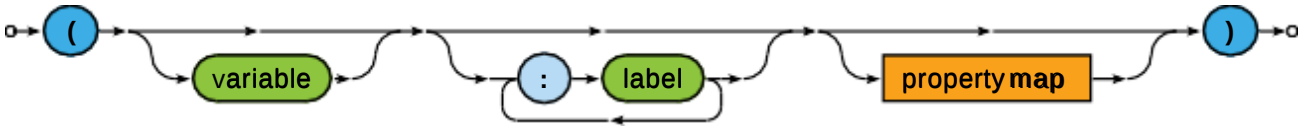


- Składnia inspirowana ASCII-Art
  - Okręgi **()** dla węzłów
  - Strzałki **<--, --, -->** dla relacji

Ścieżki

Wzorzec węzła

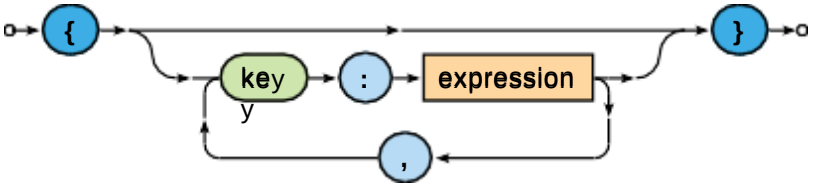
- Dopasowuje jeden węzeł danych



- **Zmienna**
  - Pozwala nam później uzyskać dostęp do danego węzła
- Zestaw **etykiet**
  - Węzeł danych musi mieć **wszystkie określone etykiety**, które mają być dopasowane
- **Właściwości**
  - Węzeł danych musi mieć dopasowane **wszystkie żądane właściwości (w tym ich wartości)** - kolejność jest nieistotna

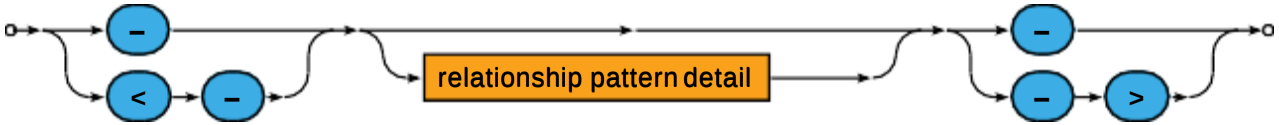
Ścieżki

Dopasowanie **właściwości**



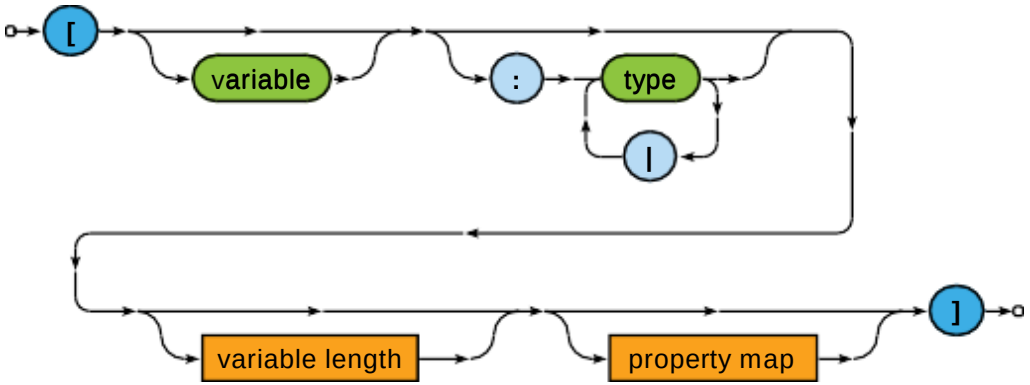
Wzorzec **relacji**

- Dopasowuje jedną relację danych



Ścieżki

Wzorzec **relacji**

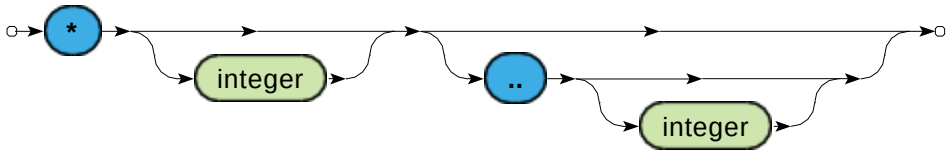


- **Zmienna**
  - Pozwala nam później uzyskać dostęp do danego węzła
- Zestaw **typów**
  - Relacja danych musi należeć do jednego z typów wyliczeniowych (by możliwe było dopasowanie)

Ścieżki

Wzorzec **relacji**

- **Właściwości**
  - Relacja danych musi mieć wszystkie żądane właściwości
- Zmienna długość ścieżki
  - Pozwala nam dopasować ścieżki o dowolnej długości (nie tylko dokładnie jeden związek)



  - Przykłady: *, *4, *2..6, *..6, *2..

Ścieżki

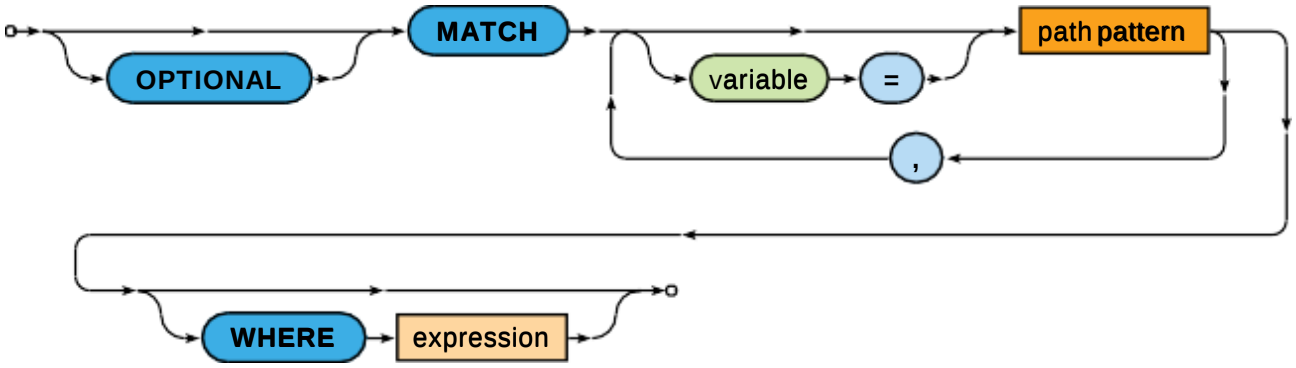
Przykłady:

()
(x)--(y)
(m:MOVIE)-->(a:ACTOR)
(:MOVIE)-->(a { name: "Sylwester Stallone" })
()<-[r:PLAY]-()
(m)-[:PLAY { role: "Lee Christmas" }]->()
(:ACTOR { name: "Jason Statham" })-[:KNOW *2]->(:ACTOR)
()-[:KNOW *5..]->(f)

Klauzula Match

Klauzula MATCH

- Umożliwia wyszukiwanie **podgrafów** w danych, które pasują do podanego wzorca/wzorców ścieżek (wszystkich)
  - **Wynik zapytania** (tabela) = nieuporządkowany zestaw rozwiązań
  - Jedno rozwiązanie (wiersz)= zestaw dopasowanych zmiennych
- Każda zmienna musi być ograniczona



Klauzula Match

Klauzula podrzędna **WHERE** może zawierać dodatkowe **ograniczenia**

- Ograniczenia te są **ewaluowane bezpośrednio na etapie dopasowywania** (tzn. nie po nim)
- Typowe zastosowanie
  - Wyrażenia logiczne
  - Porównania
  - Wzorce ścieżek– prawdziwe, jeśli zostanie znalezione co najmniej jedno rozwiązanie

Klauzula Match – przykład

Znajdź nazwiska aktorów, którzy grali z Jasonem Stathamem w dowolnym filmie

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

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

i	m	a
(a2)	(m1)	(a1)
(a2)	(m3)	(a1)
(a2)	(m4)	(a3)

⇒

a.name
Sylwester Stallone
Sylwester Stallone
Natalie Martinez

Klauzula Match

Wymóg unikalności

- Jeden węzeł danych może pasować do kilku węzłów zapytania, ale jedna relacja danych może nie pasować do kilku relacji zapytań

Opcjonalne dopasowanie

- Próbuje znaleźć dopasowany podgraf w standardowy sposób...
- ale gdy **nie zostanie znalezione rozwiązanie**, jedno specyficzne rozwiązanie ze wszystkimi zmiennymi powiązanymi z wartością NULL jest generowane
- Należy pamiętać, że albo cały wzorzec jest dopasowany, albo nic nie jest dopasowane

Klauzula Match – przykład

Znajdowanie filmów nakręconych w 2008 roku lub wcześniej oraz nazwisk grających w nich aktorów (jeśli istnieją)

```
MATCH (m:MOVIE)
    WHERE (m.year <= 2008)
OPTIONAL MATCH (m)-[:PLAY]->(a:ACTOR)
RETURN m.title, a.name
```

m
(m2)
(m4)
(m5)

⇒

m	a
(m2)	(a2)
(m4) (m4)	(a2) (a3)
(m5)	NULL

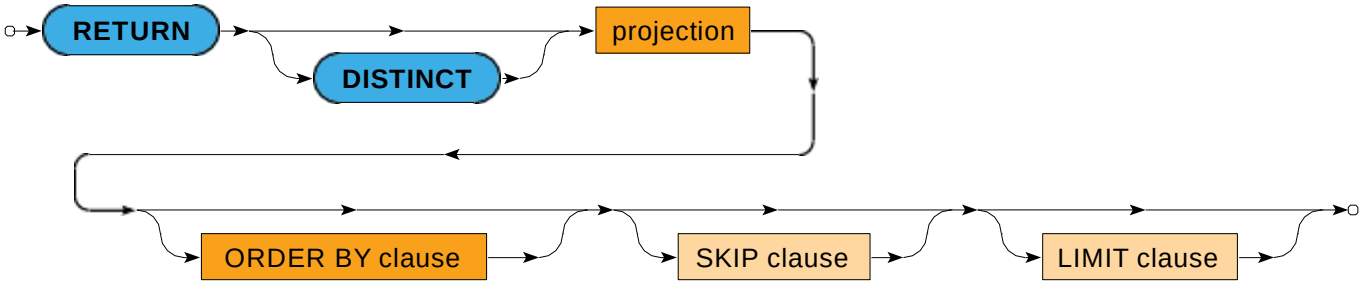
⇒

m.title	a.name
Tomb Raider	Angelina Jolie
Wyścig śmierci Wyścig śmierci	Jason Statham Natalie Martinez
Pan i Pani Smith	NULL

Klauzula Return

Klauzula RETURN

- Definiuje, co ma zostać uwzględnione w wyniku zapytania
  - Projekcja zmiennych, właściwości węzłów lub relacji (za pomocą notacji kropkowej), funkcje agregacji.
- Opcjonalne podklauzule ORDER BY, SKIP i LIMIT



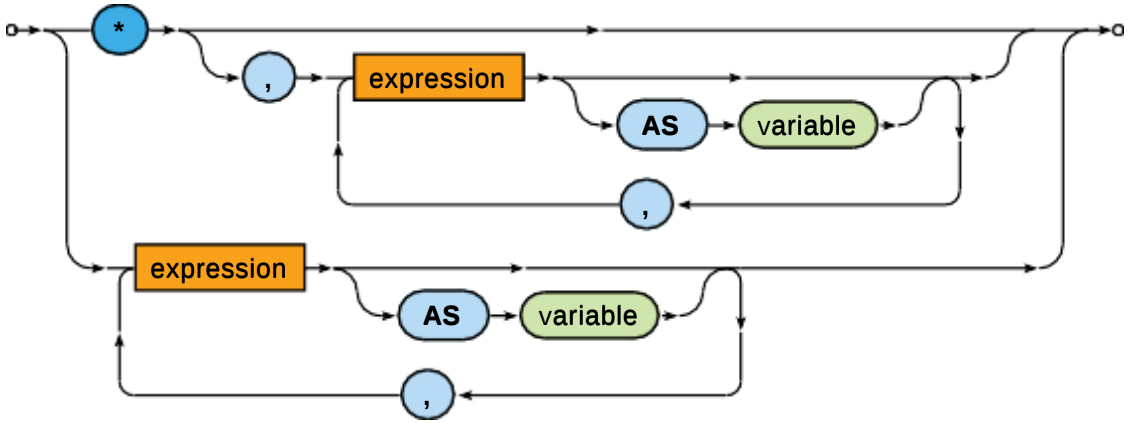
RETURN DISTINCT

- Zduplikowane rozwiązania (wiersze) są usuwane

Klauzula Return

Projekcja

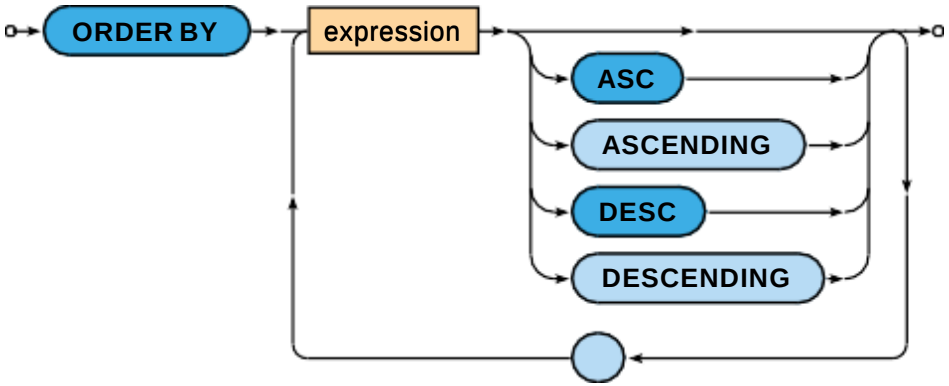
- *= **wszystkie zmienne**
  - Może być określony tylko jako pierwszy element
- AS pozwala na **jawną zmianę** nazw



Klauzula Return

Podklauzula ORDER BY

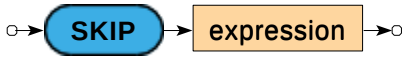
- Definiuje **kolejność** rozwiązań w wyniku zapytania
  - Można określić wiele kryteriów
  - Domyślny kierunek to ASC
- Kolejność jest niezdefiniowana, chyba że zostanie jawnie zdefiniowana
- Węzły i relacje jako takie nie mogą być używane jako kryteria



Klauzula Return

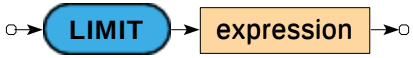
Podklazula SKIP

- Określa liczbę rozwiązań, które **mają zostać pominięte** w wyniku zapytania



Podklauzula LIMIT

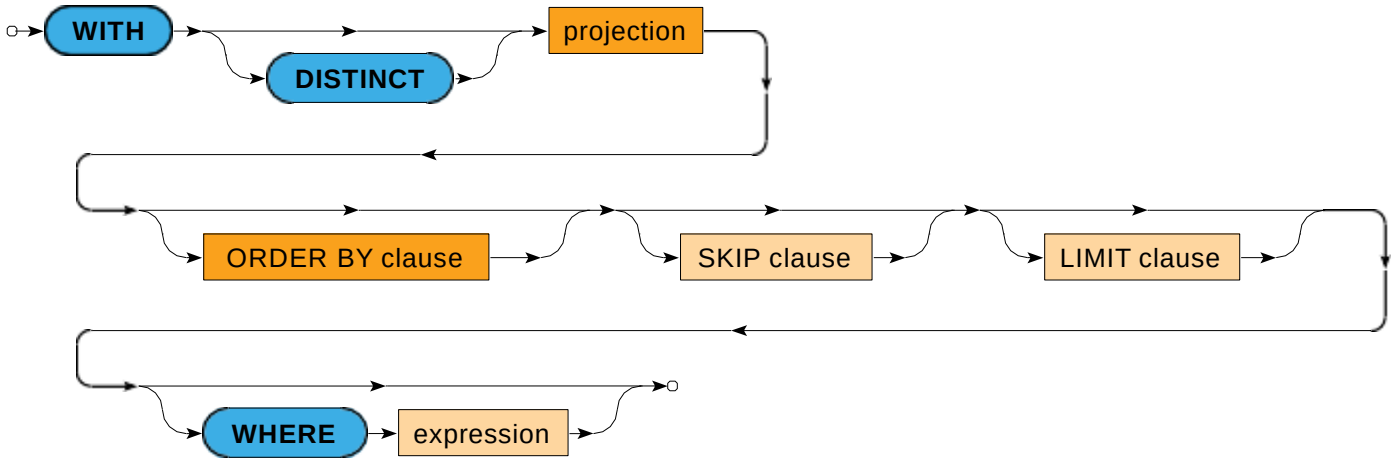
- Określa liczbę rozwiązań, które **mają zostać uwzględnione** w wyniku zapytania



Klauzula With

Klauzula WITH

- **Konstruuje wynik pośredni**
  - Zachowanie analogiczne do klauzuli RETURN
  - Nie wyświetla niczego użytkownikowi, po prostu przekazuje bieżący wynik do kolejnej klauzuli
- Można również podać opcjonalny warunek WHERE



Klauzula With – przykład

Liczba filmów, w których zagrali aktorzy urodzeni w 1975 roku lub później

```
MATCH (a:ACTOR)
    WHERE (a.year >= 1975)
WITH a, SIZE( (a)<-[:PLAY]-(m:MOVIE) ) AS movies
RETURN a.name, movies
    ORDER BY movies ASC
```

a
(a2)
(a3)

⇒

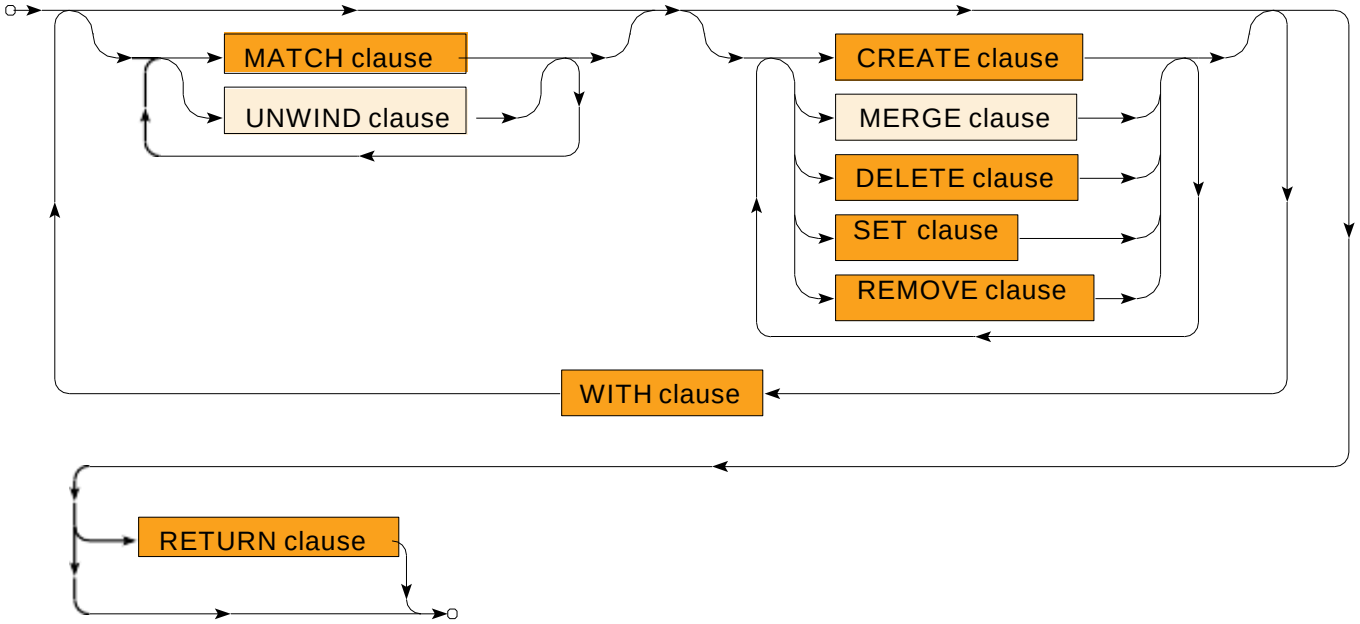
a	movies
(a2)	1
(a3)	1

⇒

a.name	movies
Natalie Martinez	1
Angelina Jolie	1

Struktura zapytań

Łączenie w łańcuch klauzul **Cypher** (uproszczone)



- **Selekcja**: MATCH
- **Modyfikacja**: CREATE, DELETE, SET, REMOVE

Struktura zapytań

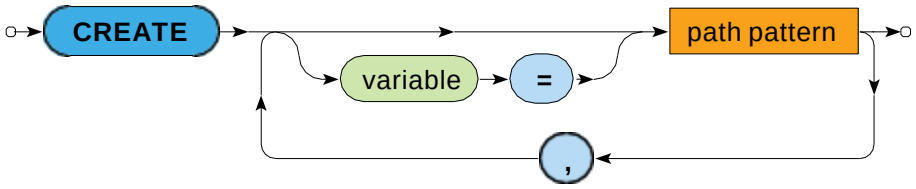
Struktura zapytań:

- **Klauzule WITH dzielą całe zapytanie na części**
- Obowiązują pewne ograniczenia:
  - **Klauzule związane z odczytem** (jeśli występują) **musi poprzedzać te związane z modyfikacja** w każdej części zapytania
  - **Ostatnia część zapytania musi być zakończona klauzulą RETURN**
    - Chyba, że ta część zawiera co najmniej jedną klauzulę write
    - Oznacza to, że zapytania tylko do odczytu muszą zwracać dane

Klauzule związane z zapisem

Klauzula CREATE

- **Wstawia nowe węzły lub relacje** do grafu danych



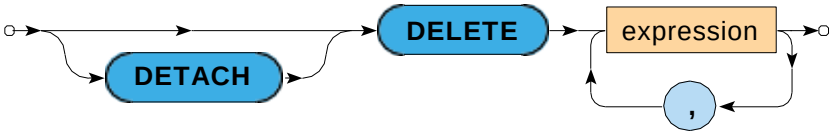
Przykład

```
MATCH (m:MOVIE { id: "panipanismith"})
CREATE
    (a:ACTOR { id: "pitt", name: "Brad Pitt", year: 1963}),
    (m)-[:PLAY]->(a)
```

Klauzule związane z zapisem

Klauzula DELETE

- **Usuwa węzły, relacje lub ścieżki** z grafu danych
- Relacje muszą być zawsze usuwane przed węzłami, z którymi są skojarzone
  - Chyba, że podano modyfikator DETACH



Przykład

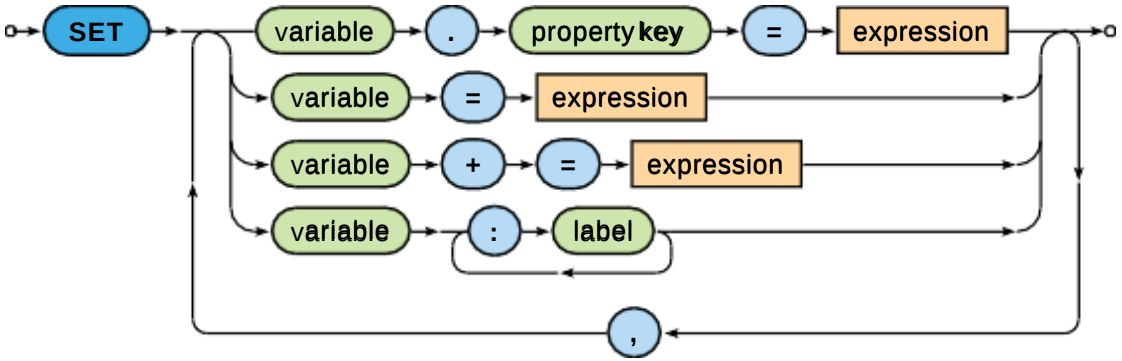
```
MATCH (:MOVIE { id: "panipanismith"})-[r:PLAY]->(a:ACTOR)  DELETE r
```

Klauzule związane z zapisem

Klauzula SET

- Pozwala na…
  - **Ustawianie wartości określonej właściwości**
    - lub usunąć właściwość, gdy przypisana jest wartość NULL
  - **Zastąpienie właściwości** (wszystkich) nowymi
  - **Dodawanie nowych właściwości** do istniejących
  - **Dodawanie etykiet** do węzłów

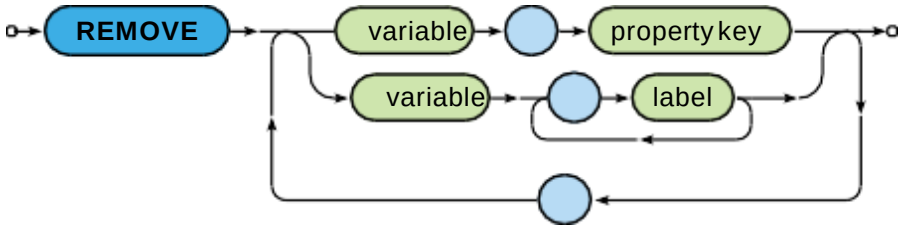
  Nie można używać SET do ustawiania typów relacji



Klauzule związane z zapisem

Klauzula REMOVE

- Pozwala na…
  - **Usuwanie** określonej właściwości, usuwanie etykiet z węzłów
  - **Nie można jej użyć do usuwania typów relacji**



Wyrażenia

Literały

- Liczby całkowite: dziesiętny, ósemkowy, szesnastkowy
- Liczby zmiennoprzecinkowe
- Ciągi znaków
  - Ujęte w cudzysłów lub apostrofy
  - Standardowe sekwencje ucieczki
- Wartości logiczne: true, false
- Wartość NULL (nie może być przechowywana w grafie)

Inne wyrażenia

- Kolekcje, zmienne, metody dostępu do właściwości, wywołania funkcji, wzorce ścieżek, wyrażenia logiczne, wyrażenia arytmetyczne, porównania, wyrażenia regularne, predykaty, itd..

Pytania? Problemy?

Dziękuję za uwagę!