Grafové databázy

Zástupca s ktorým budeme pracovať: Neo4j

EURÓPSKA ÚNIA
Európsky sociálny fond
Európsky fond regionálneho rozvoja

OPERAČNÝ PROGRAM
ĽUDSKÉ ZDROJE

MINISTERSTVO
ŠKOLSTVA, VEDY,
VÝSKUMU A ŠPORTU
SLOVENSKEJ REPUBLIKY

UNIVERZITA
MATEJA BELA

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu a Európskeho fondu regionálneho rozvoja v rámci Operačného programu Ľudské zdroje

www.minedu.sk www.employment.gov.sk/sk/esf/ www.itakademia.sk

Grafové DB

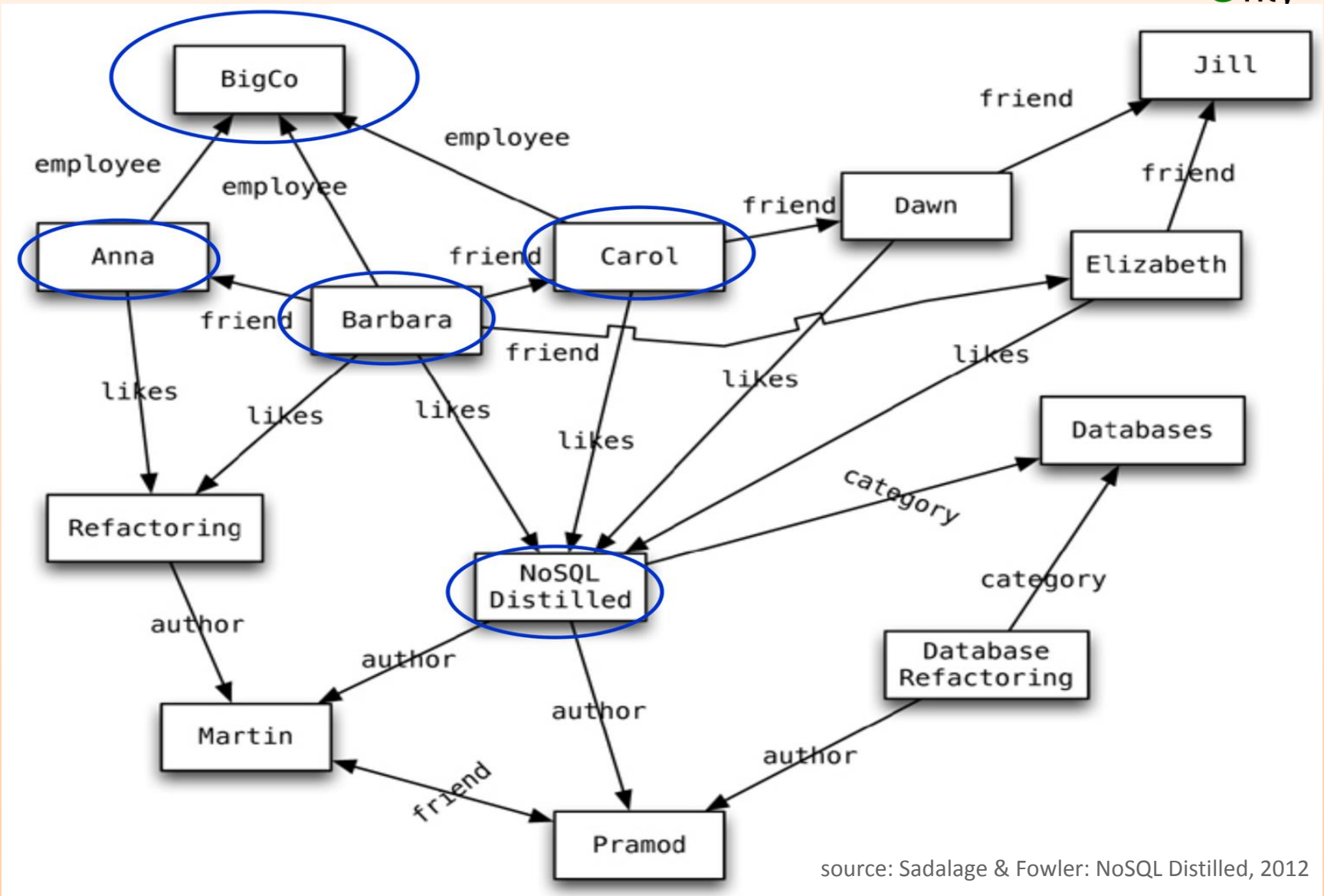
- Grafové typy informácií
  - Vzťahy medzi objektmi sú dôležitejšie ako vlastnosti objektov
- Sociálne siete, prepojenia medzi webovými stránkami, chemické zlúčeniny, podobnosti a väzby medzi objetmi, záujmy používateľov,...
- Keď v štandardnej relačnej databáze si neporadíme bez rekurzie

Porovnanie s dokumentovými DB

- Dokumentové DB sú určené pre dáta, ktoré môžu byť reprezentované ako stromy
- Prepojenia medzi stromami sú možné, ale ich využívanie je drahé
- Grafovým databázam vyhovujú rovnako stromové ako aj všeobecné grafy

Typy grafov

- orientované vs. neorientované
- jednovzťahové vs. viacvzťahové (labeled)
  - či všetky hrany vyjadrujú rovnaký druh vzťahu medzi vrcholmi, alebo hrany majú svoj typ
- atribútové – uzly aj hrany môžu mať ľubovoľné množstvo vlastností (niečo ako hodnoty inštančných premenných objektov)
- multigrafy – medzi dvoma uzlami môže byť viac hrán, obvykle iného typu
- hypergrafy – hrany môžu spájať viac ako dva vrcholy



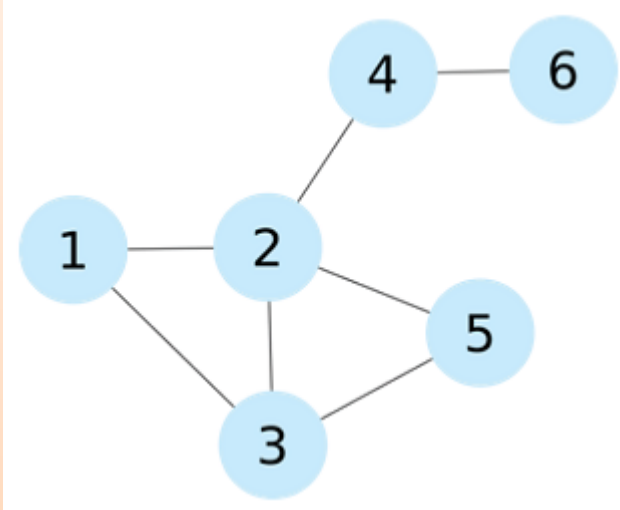
source: Sadalage & Fowler: NoSQL Distilled, 2012

Trochu teórie

- Dáta = množina vrcholov a vzťahov medzi nimi
  - Potrebujeme efektívnu formu reprezentácie grafu
- Základné operácie (potreba efektívnych grafových operácií)
  - Nájdenie susedov vrcholu
  - Zistenie, či dva vrcholy majú spoločnú hranu
  - Zmena štruktúry grafu
- Akú zvoliť dátovú štruktúru?

Matica susednosti

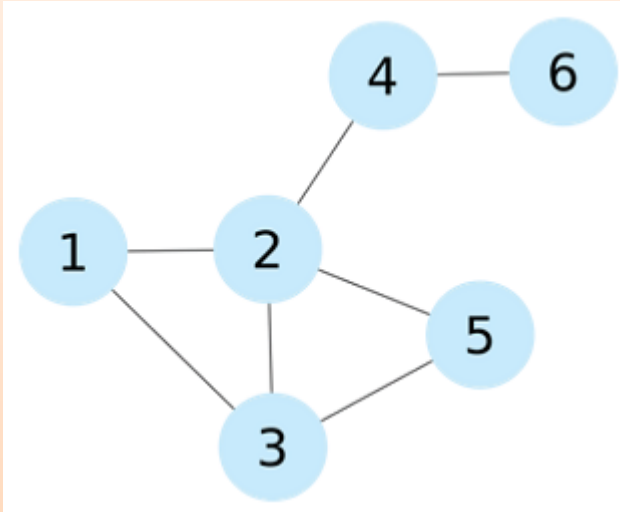
- Dvojrozmerná matica n x n
- i-ty vrchol je v i-tom riadku (výstupná hrana) a i-tom stĺpci (vstupná hrana)
- Ak máme neorientovaný graf, matica je symetrická
- Ak máme vážený/ohodnotený graf – čísla predstavujú váhu/ohodnotenie hrán



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Matica susednosti

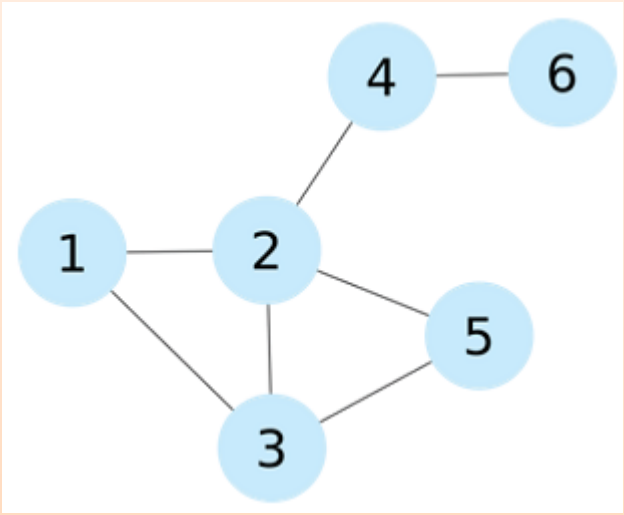
- Výhody:
  - Pridávanie a odoberanie hrán
  - Zisťovanie prepojenia dvoch vrcholov
- Nevýhody:
  - Kvadratický (n^2) priestor
  - V reálnom svete máme riedke grafy: veľa núl
  - Pridávanie a mazanie vrcholov je drahé
  - Získanie všetkých susedov vyžaduje čas O(n)



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Laplaceova matica

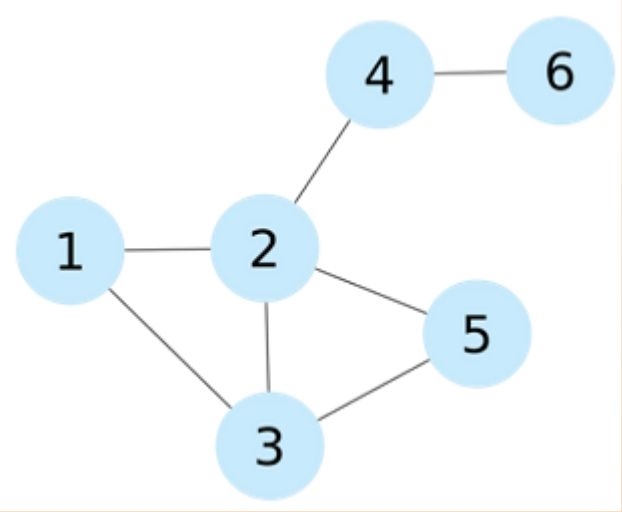
- Podobná matici susednosti (veľkosť je n x n)
- Na diagonále je stupeň vrchola (t.j. počet s ním incidentných hrán)
- Ostatné pozície v matici sú buď -1 (vrcholy sú prepojené hranou), alebo 0 (hrana medzi vrcholmi neexistuje)



$$\begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 & 0 \\ 0 & -1 & 0 & 2 & 0 & -1 \\ 0 & -1 & -1 & 0 & 2 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

Laplaceova matica

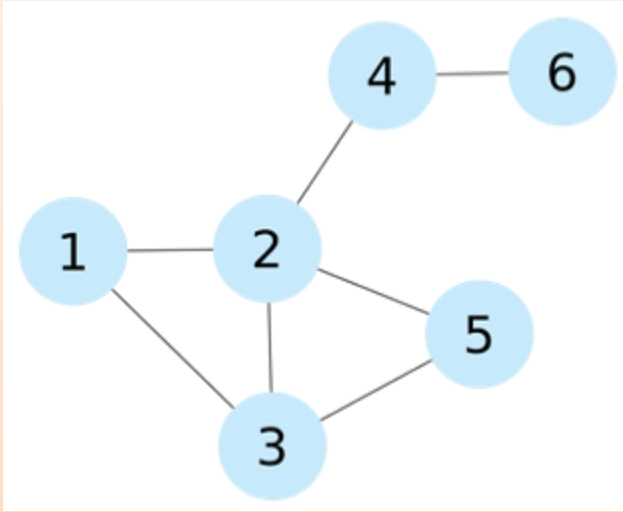
- Rovnaké výhody aj nevýhody ako matica susednosti
- Bonusovou výhodou je analýza štruktúry grafu cez zložitejšie operácie
  - Napr. cez determinant a vlastné čísla matice



$$\begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 & 0 \\ 0 & -1 & 0 & 2 & 0 & -1 \\ 0 & -1 & -1 & 0 & 2 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

Matica incidencie

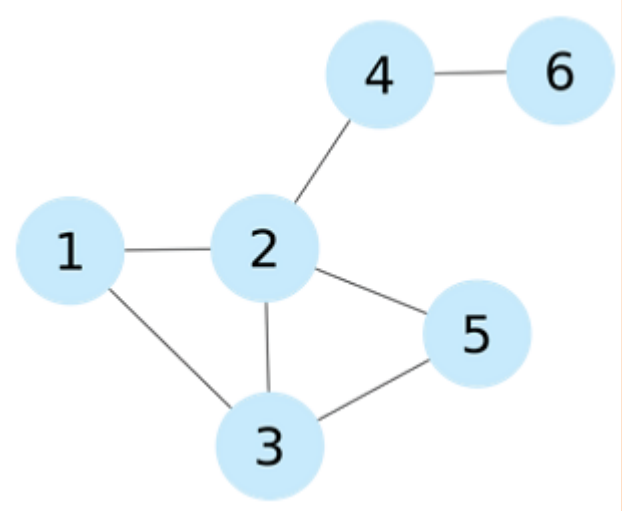
- Riadky reprezentujú uzly
- Stĺpce reprezentuje hrany
  - V každom stĺpci sú práve dve 1
- Výhody:
  - Možnosť reprezentácie hypergrafov (viac ako dve 1 v stĺpci)
- Nevýhody:
  - Obvykle vyžaduje ešte oveľa viac priestoru ako matica susednosti, lebo pre väčšinu grafov je počet hrán oveľa väčší ako počet uzlov



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Zoznam susedov

- Množina zoznamov, kde každý zoznam obsahuje všetkých susedov (smerníkov na nich) uzla
- Ak máme neorientovaný graf a zoznam susedov uzla i obsahuje uzol j, tak aj zoznam susedov uzla j obsahuje uzol i
- Je možné použiť kompresiu



N1 → {N2, N3}

N2 → {N1, N3, N5}

N3 → {N1, N2, N5}

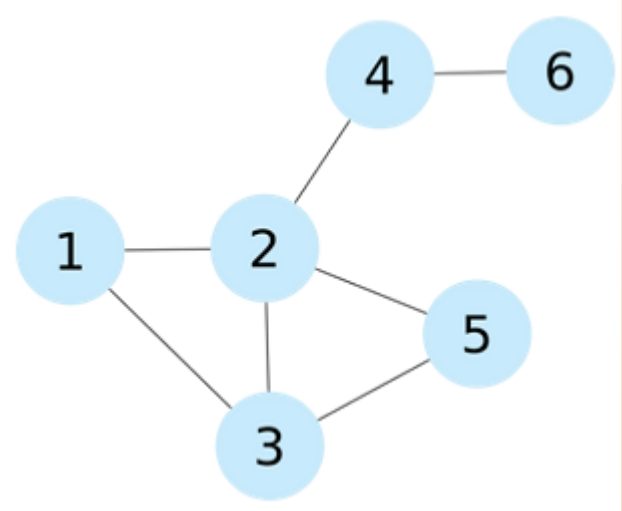
N4 → {N2, N6}

N5 → {N2, N3}

N6 → {N4}

Zoznam susedov

- Výhody:
  - Získanie všetkých susedov uzla
  - Lacné pridávanie uzlov aj hrán
  - Priestorovo úsporná reprezentácia
- Nevýhody:
  - Zisťovanie existencie hrany medzi dvoma uzlami
    - v prípade usporiadaných zoznamov logaritmický čas inak treba iterovať celý zoznam uzla
  - Strácame bonusovú informáciu o hrane (ohodnotenie)



N1 → {N2, N3}

N2 → {N1, N3, N5}

N3 → {N1, N2, N5}

N4 → {N2, N6}

N5 → {N2, N3}

N6 → {N4}

Základné grafové algoritmy

- Prechod všetkými uzlami
  - Prechod do hĺbky
  - Prechod do šírky
- Nájdenie najkratšej cesty medzi dvoma uzlami
- Nájdenie najkratších ciest k uzlom
  - Neohodnotené grafy: prechod uzlami
  - Ohodnotené grafy: Dijkstra/Bellman-Ford
- Nájdenie najkratších ciest medzi všetkými dvojicami vrcholov
  - Floyd-Warchall

Not Only SQL

Databázy grafov

- Transakčné databázy – veľké množiny malých grafov
  - Chemické zlúčeniny, lingvistické stromy,
- Netransakčné databázy – málo veľmi veľkých grafov
  - Graf webov, sociálne siete, sémantický web

Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

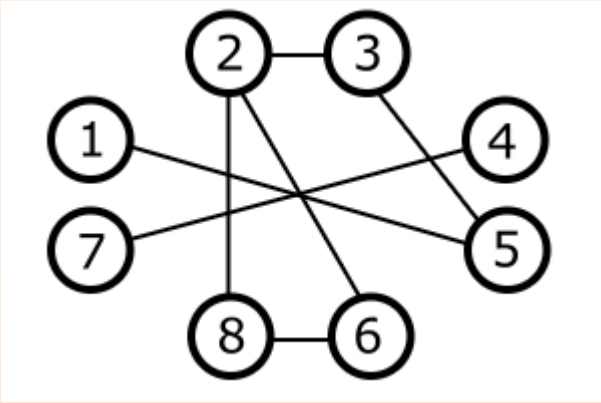
Dopyty typické pre transakčné DB

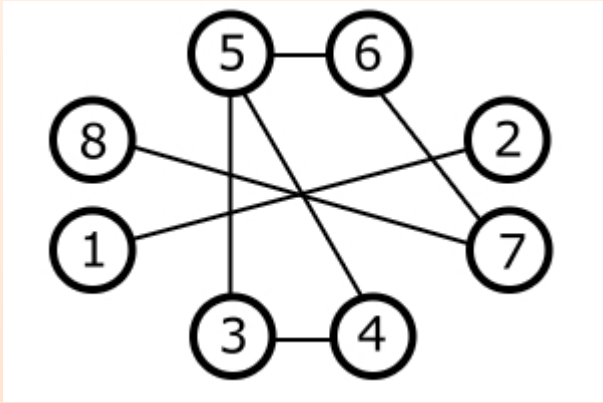
- Hľadanie podgrafu
  - Nájdenie nejakého vzoru vo veľkom grafe (kde všade sa graf izomorfný s dopytovaným vyskytuje)
- Hľadanie nadgrafu
  - Nájdenie všetkých grafov, ktoré sú v dopytovanom grafe obsiahnuté (napr. zisťujeme zloženie zlúčeniny)
- Hľadanie podobného grafu
  - Súčasťou dopytu musí byť aj definícia podobnosti
- Dopytové grafy môžu byť špecifikované rôzne - napr. sa dá definovať, ktoré hrany sú nutné, ktoré iba možné a ktoré zakázané

Umiestnenie vrcholov v štruktúre

- Ak sú dáta na disku, je výhodné, ak s každým vrcholom získam aj jeho okolie (na stránku vojdú 4kB), ktoré s vysokou pravdepodobnosťou budem potrebovať v ďalšom kroku algoritmu
- Problém lokality dát
  - Ukážeme si problém minimalizácie šírky pásma matice
  - Šírka pásma matice = maximálna vzdialenosť medzi nenulovými prvkami riadka matice (susednosti), pričom jeden je nad a druhý pod diagonálou.

Minimalizácia šírky pásma matice





$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

prečíslovanie vrcholov

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Vo všeobecnosti: NP-ťažký problém

Pre väčšie grafy – heuristiky k suboptimálnemu riešeniu

Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

Neo4j

- Uchováva orientované, viacvzťahové, atribútové multigrafy
- Prístup cez:
  - Java API (v Jave je Neo4j aj nakódená)
  - Jazyk Gremlin – „štandard“ pre grafové DB – umožňuje jednoduché definovanie akcií pri prechode ohodnoteným atribútovým grafom
  - **Jazyk Cypher** - deklaratívny jazyk (čo chcem získať a nie to, ako prechádzať grafom)

Not Only SQL

Neo4j: Základné vlastnosti

- Podporuje ACID vlastnosti
  - Všetky WRITE operácie musia byť realizované v transakcii
  - WRITE transakcie sú celé realizované v pamäti až do momentu commitu (veľké zmeny treba kúskovať do viacerých transakcií)
  - READ operácie môžu byť v transakcii (ak nechceme non-repeatable read)
- Dáta ukladá na disk
- Škálovateľná iba cez replikáciu
  - Dáta sa nedelia medzi servery
  - Vždy jeden master a ľubovoľne veľa slave-ov
    - Záležitosť konfiguácie – kód klienta sa nemení (stačí nakonfigurovať driver)
  - Zápis na slave zrealizuje v tej istej transakcii zápis aj na master

Hrany

- Orientované hrany
  - Výstupná na jednom konci a vstupná na druhom
  - Môžu vchádzať do toho istého vrchola z ktorého vychádzajú (rekurzívne hrany)
  - Prechod hranou oboma smermi je rovnako efektívny
- Orientácia hrán môže byť ignorovaná

Neo4j: Dátový model

- Iba jeden priestor pre všetky grafy
  - Ak chceme oddeliť množiny grafov, napr. pre rôzne aplikácie, potrebujeme novú inštaláciu databázy
- Základné prvky: vrcholy a hrany
  - Vrcholy môžu obsahovať označenia (labels)
  - Vrcholy aj hrany môžu obsahovať ľubovoľne veľa vlastností
    - Dvojice kľúč – hodnota
    - Kľúč je String
    - Hodnota môže byť primitívny typ Javy (int, char, double, ...), String, alebo pole primitívnych typov
    - null ako hodnota nie je povolená – stačí vynechať daný kľúč
- Indexy nad vrcholmi/hranami podľa vlastností alebo explicitné

Java API – pre lokálne úložisko

```
GraphDatabaseService db = new GraphDatabaseFactory()
        .newEmbeddedDatabase(new File(LOCAL_STORE_DIR));
Node peto = null;
try ( Transaction tx = db.beginTx() ) {
  peto = db.createNode();
  peto.setProperty("name","Peter");
  peto.setProperty("e-mail","peter.gursky@upjs.sk");
  Node nosql = db.createNode();
  peto.setProperty("web","http://nosql.ics.upjs.sk");
  Relationship petoNosql = peto.createRelationshipTo(nosql,TEACHES);
  petoNosql.setProperty("at", "UPJŠ");
  tx.success();
}
```

Java API – pre lokálne úložisko

```
System.out.println(peto.getProperty("e-mail"));
Relationship r = peto.getSingleRelationship(TEACHES,
                                    Direction.OUTGOING);
System.out.println(r.getProperty("at"));
Node n2 =r.getEndNode();
System.out.println(n2.getProperty("web"));

//zápisy musia byť v transakcii
try ( Transaction tx = db.beginTx() ) {
  r.delete();
}
db.shutdown();
```

Prechádzanie grafom

- Najdôležitejšia úloha grafovej databázy
- Najčastejšie prechádzanie do hĺbky/šírky zo štartovacieho vrchola (vrcholov) sledujúc pravidlá dané v dopyte
  - označenia a vlastnosti vrcholov a hrán
  - hĺbka vnorenia
  - opakovanie navštevovania vrcholov, hrán, alebo ciest
  - akcie pre každý vrchol prechodu
- Výsledkom môžu byť cesty zo štartovacieho vrchola, vrcholy alebo hrany

Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

Prechádzanie grafom – Java API

```
TraversalDescription traversalDescription = db.traversalDescription();
Traverser traverser = traversalDescription.traverse(startNode);
```

- traversalDescription je objekt s pravidlami prechodu grafom
  - Default je prechod celým grafom do hĺbky bez opakovania vrcholov, ignorujúce smer hrán
- Traverser môže vrátiť:
  - .iterator() – cesty od štartovacieho vrchola (vrcholov) ku všetkým vrcholom prechodu
  - .nodes() – vrchioly v poradí prechádzania (konce ciest)
  - .relationships() – posledné hrany ciest od štarovacieho vrchola (vrcholov) ku každému vrcholu prechodu

Štelovanie traversalDescription

- .breathFirst()/depthFirst() – do šírky/hĺbky
- .relationships(typ_hrany, smer)
  - viac zavolaní – viac možných typov hrán
  - smer – Direction.BOTH / .INCOMING / .OUTGOING
- .uniqueness(opakovanie)
  - Uniqueness.X_Y
    - X: *RELATIONSHIP / NODE*
    - *Y*:
      - *GLOBAL - každý X sa navštívi maximálne raz*
      - *LEVEL - X na rovnakej vzdialenosti od štartovacieho uzla sú unikátne*
      - *PATH - pre každý X je unikátna cesta zo štartovacieho uzla*
      - *RECENT - unikátnosť rozhodujeme na základe posledných n X, pričom n sa nastaví cez druhý parameter metódy*
    - *NONE – unikátnosť sa nesleduje*

Štelovanie traversalDescription

```
.evaluator(new Evaluator() {
   public Evaluation evaluate(Path path) {
      ...
   }
});
```

- Evaluation
  - .INCLUDE_AND_CONTINUE – pošli cestu do výsledku a pokračuj v prechode grafom
  - .INCLUDE_AND_PRUNE – pošli cestu do výsledku a ukonči prechod grafom
  - .EXCLUDE_AND_CONTINUE – cestu neposielaj do výsledku ale pokračuj ďalej v prechode grafom
  - .EXCLUDE_AND_PRUNE – cestu neposielaj do výsledku a skonči prechod grafom
- Hotové evaluátory ako statické metódy triedy Evaluators
  - **toDepth**(hĺbka), **fromDepth**(hĺbka), **atDepth**(hĺbka), **includingDepths**(minHĺbka, maxHĺbka), **excludeStartPosition**()
  - **includeIfAcceptedByAny**(evaluator1, evaluator2, ...)
  - **includeIfContainsAll** (node1, node2,...)
  - **includeWhereLastRelationshipTypeIs**(type1, type2,...)
  - **pruneWhereEndNodeIs**(node1, node2,...)
  - **pruneWhereLastRelationshipTypeIs**(type1, type2,...)
  - **lastRelationshipTypeIs**(evaluationIfMatch, evaluationIfNoMatch, type1, type2,...)

Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

Cypher

- deklaratívny dopytovací jazyk
  - píšeme, čo chceme dosiahnuť / získať
  - nemusíme definovať spôsob prechádzania grafom
- možnosť pracovať so vzdialenou DB
  - java API sa nedá použiť nad vzdialenou DB

Klauzule jazyka Cypher

- **Zápis do DB**
  - CREATE – vytvára nové vrcholy a hrany s úvodnými vlastnosťami a úvodnými označeniami vrcholov (labels)
  - SET – nastavuje označenia vrcholov a vlastnosti vrcholov a hrán
  - FOREACH – používa sa na hromadné zmeny dát
  - MERGE – ak sa vzor v grafe nájde, tak sa vráti a ak sa nenájde, tak sa vytvorí a vráti
- **Mazanie z DB**
  - DELETE – maže vrcholy, hrany alebo cesty, pričom každý mazaný vrchol musí mať pred tým vymazané všetky s ním incidentné hrany
  - DETACH DELETE – maže vrcholy, hrany s nimi incidentné sa mažú automaticky
  - REMOVE – maže označenia vrcholov a vlastnosti vrcholov a hrán

Klauzule jazyka Cypher

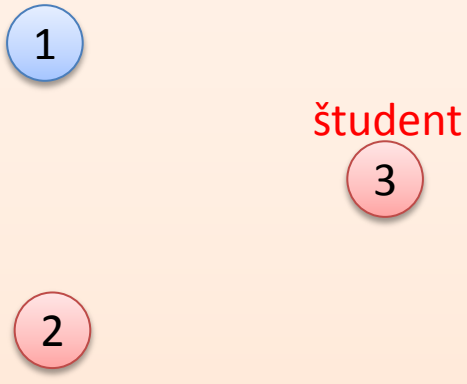
- **Čítanie z DB** – niečo ako klauzula FROM z SQL
  - MATCH
  - OPTIONAL MATCH – ak sa nájde iba časť zvyšok sa doplní s null
- **Čo chceme vrátiť**
  - RETURN – definuje stĺpec výsledku vyskladaný z premenných z MATCH
  - WITH – umožňuje výsledok jedného dopytu použiť v ďalšom
  - UNWIND – rozloží list objektov do riadkov
- **Subklauzule čítania**
  - WHERE – definuje obmedzenia pre to, čo nájde MATCH
  - ORDER BY
  - SKIP – od ktorého riadku výsledku začať posielať
  - LIMIT – koľko riadkov výsledku sa má poslať
  - UNION, UNION ALL – spojenie viacerých výsledkov

- CREATE (n)

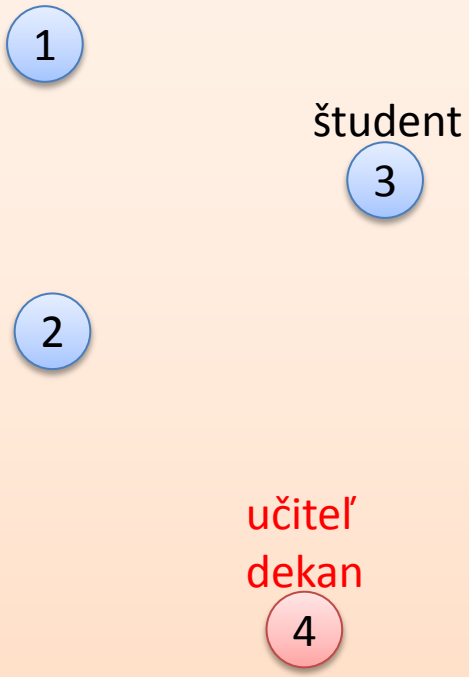
1

Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

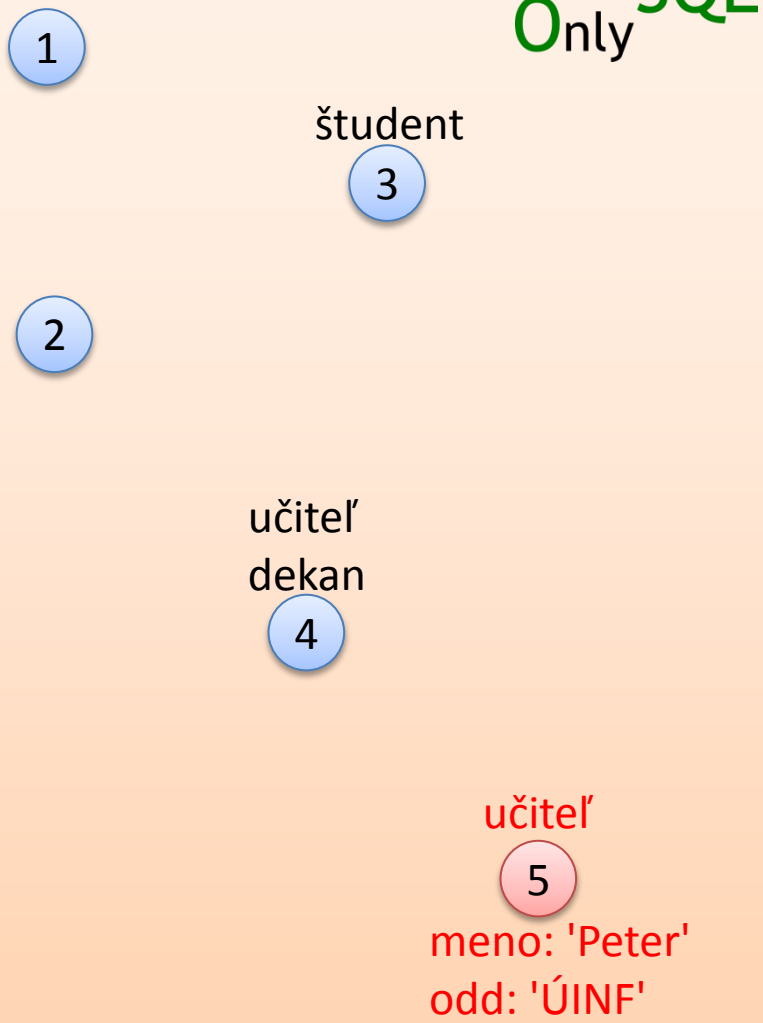
- CREATE (n)
- CREATE (n),(m:študent)



- CREATE (n)
- CREATE (n),(m:študent)
- CREATE (n:učiteľ:dekan)

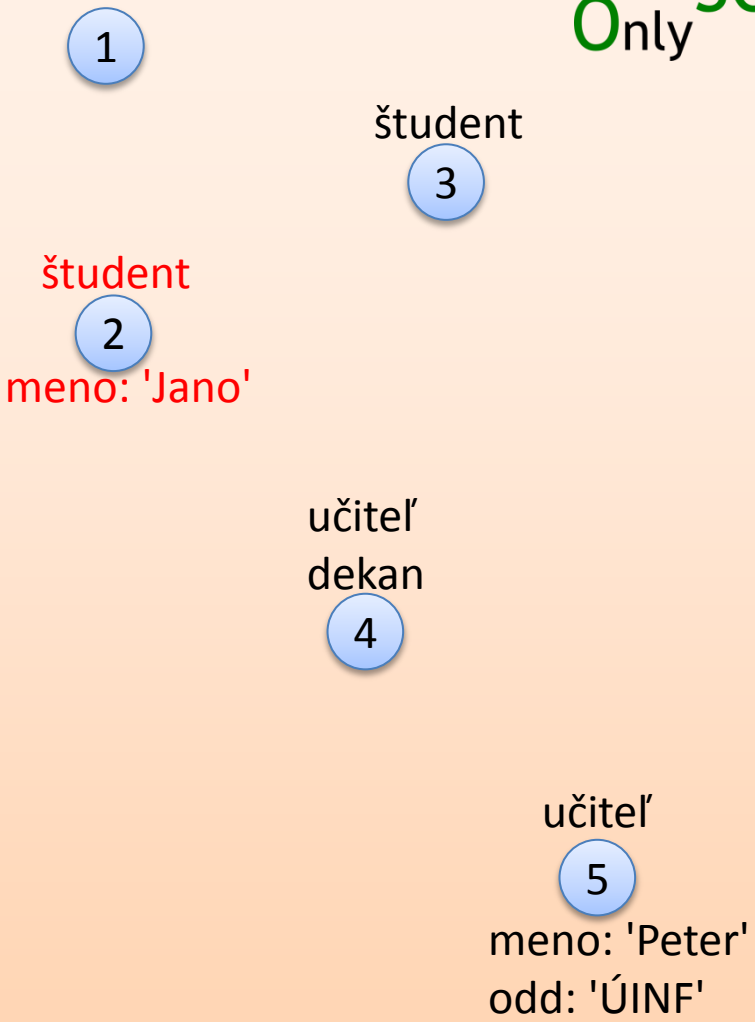


- CREATE (n)
- CREATE (n),(m:študent)
- CREATE (n:učiteľ:dekan)
- CREATE (n:učiteľ {meno: 'Peter', odd: 'ÚINF'})



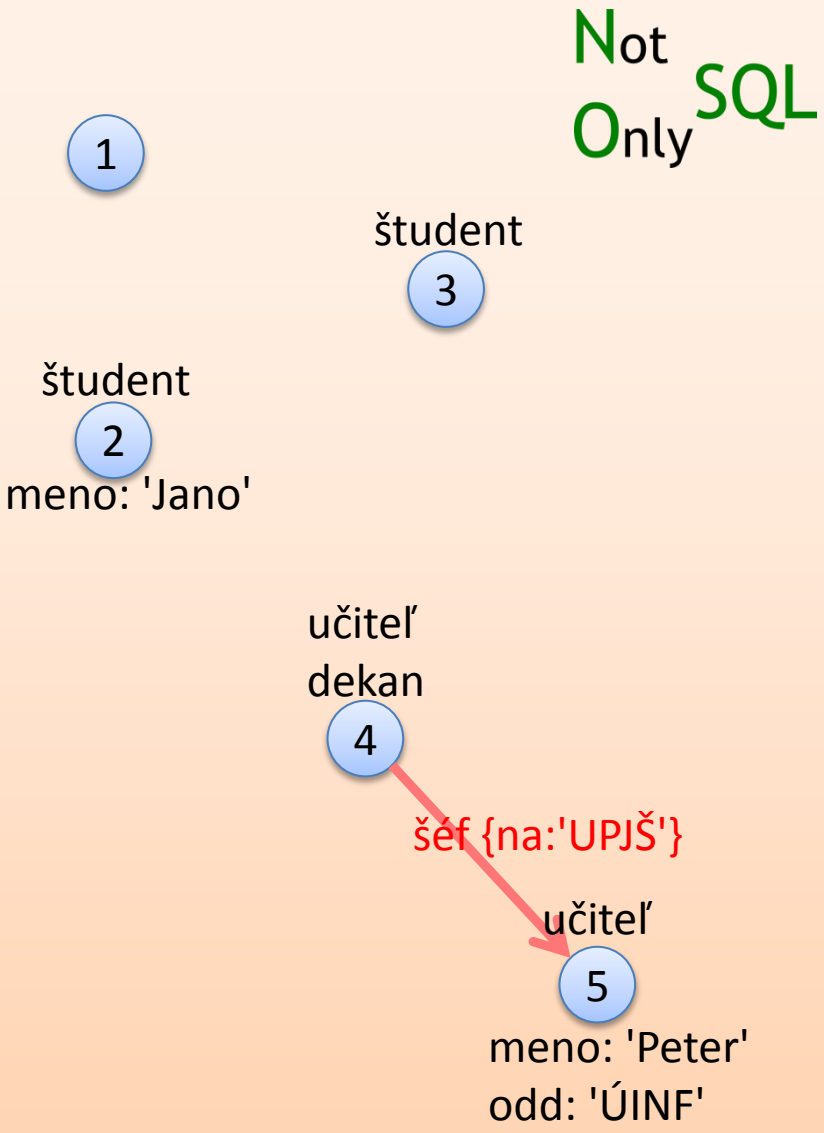
- CREATE (n)
- CREATE (n),(m:študent)
- CREATE (n:učiteľ:dekan)
- CREATE (n:učiteľ {meno: 'Peter', odd: 'ÚINF'})
- MATCH (n)
  WHERE ID(n) = 2
  SET n.meno = 'Jano', n:študent

Not Only SQL



- CREATE (n)
- CREATE (n),(m:študent)
- CREATE (n:učiteľ:dekan)
- CREATE (n:učiteľ {meno: 'Peter', odd: 'ÚINF'})
- MATCH (n)
  WHERE ID(n) = 2
  SET n.meno = 'Jano', n:študent
- MATCH (a:dekan), (b:učiteľ)
  WHERE b.meno= 'Peter'
  CREATE (a)-[r:šéf {na:'UPJŠ'}]->(b)
  RETURN type(r), r.na
  - vráti tabuľku:

type(r)	r.Na
"šéf"	"UPJŠ"

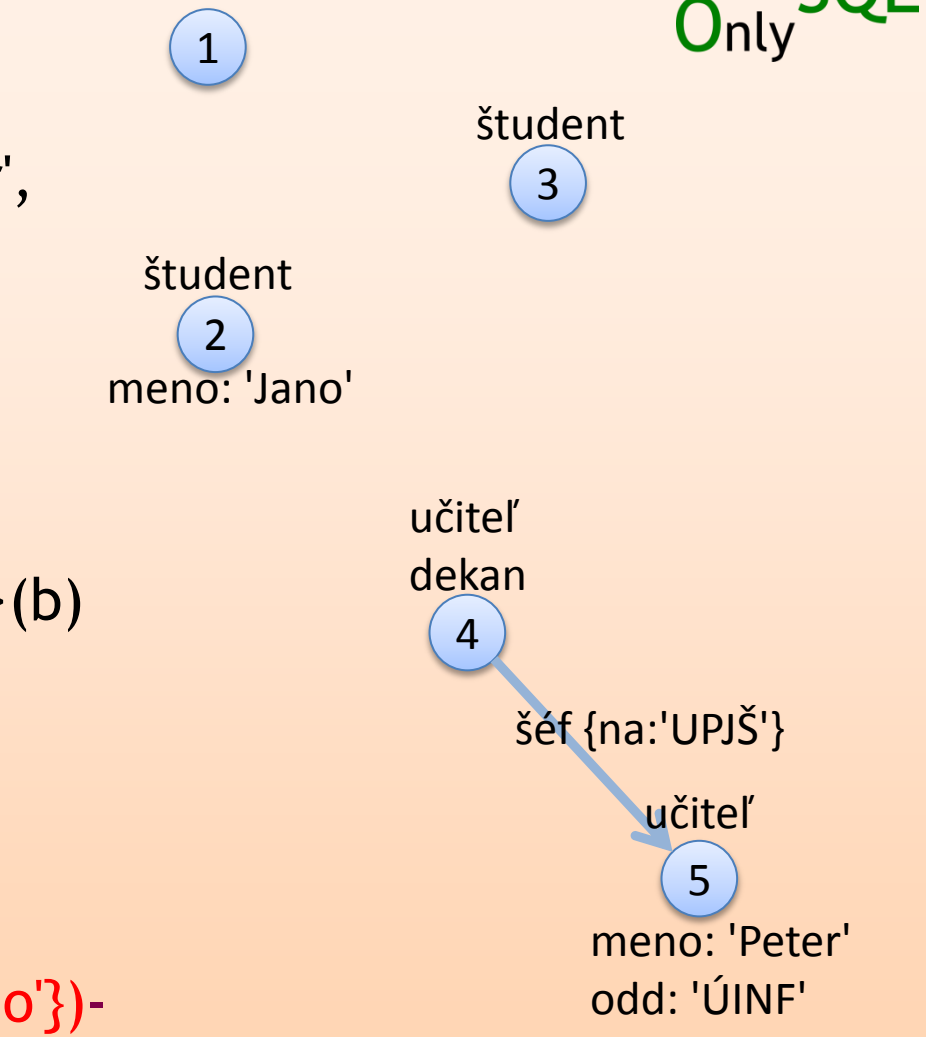


Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

- CREATE (n)
- CREATE (n),(m:študent)
- CREATE (n:učiteľ:dekan)
- CREATE (n:učiteľ {meno: 'Peter', odd: 'ÚINF'})
- MATCH (n)
  WHERE ID(n) = 2
  SET n.meno = 'Jano', n:študent
- MATCH (a:dekan), (b:učiteľ)
  WHERE b.meno= 'Peter'
  CREATE (a)-[r:šéf {na:'UPJŠ'}]->(b)
  RETURN type(r), r.na
  - vráti tabuľku:

type(r)	r.Na
"šéf"	"UPJŠ"

- CREATE (n:študent {meno: 'Jano'})-[:kamoši]->(m:učiteľ:dekan)

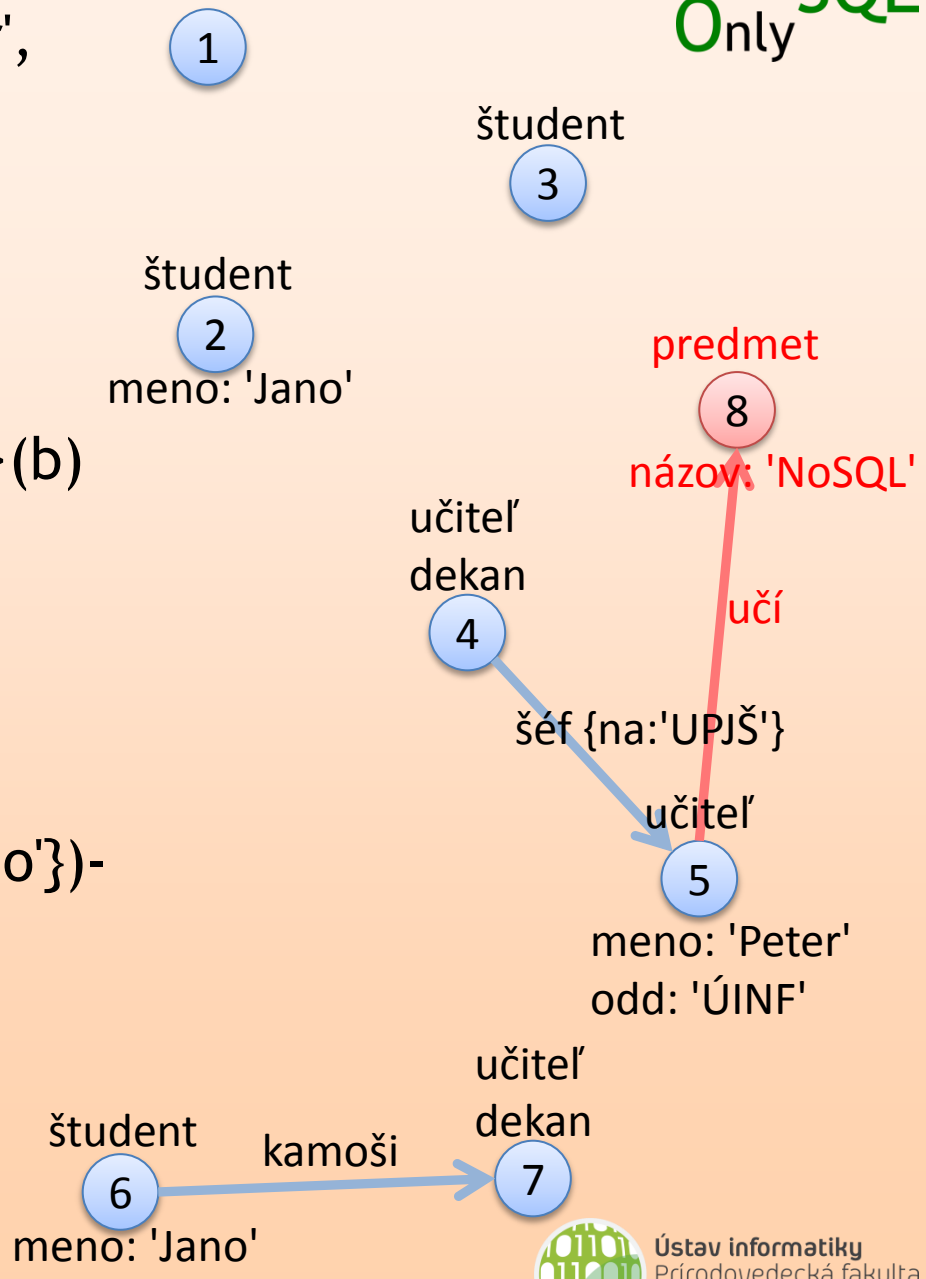


Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

- CREATE (n:učiteľ:dekan)
- CREATE (n:učiteľ {meno: 'Peter', odd: 'ÚINF'})
- MATCH (n)
  WHERE ID(n) = 2
  SET n.meno = 'Jano', n:študent
- MATCH (a:dekan), (b:učiteľ)
  WHERE b.meno= 'Peter'
  CREATE (a)-[r:šéf {na:'UPJŠ'}]->(b)
  RETURN type(r), r.na
  - vráti tabuľku:

type(r)	r.Na
"šéf"	"UPJŠ"

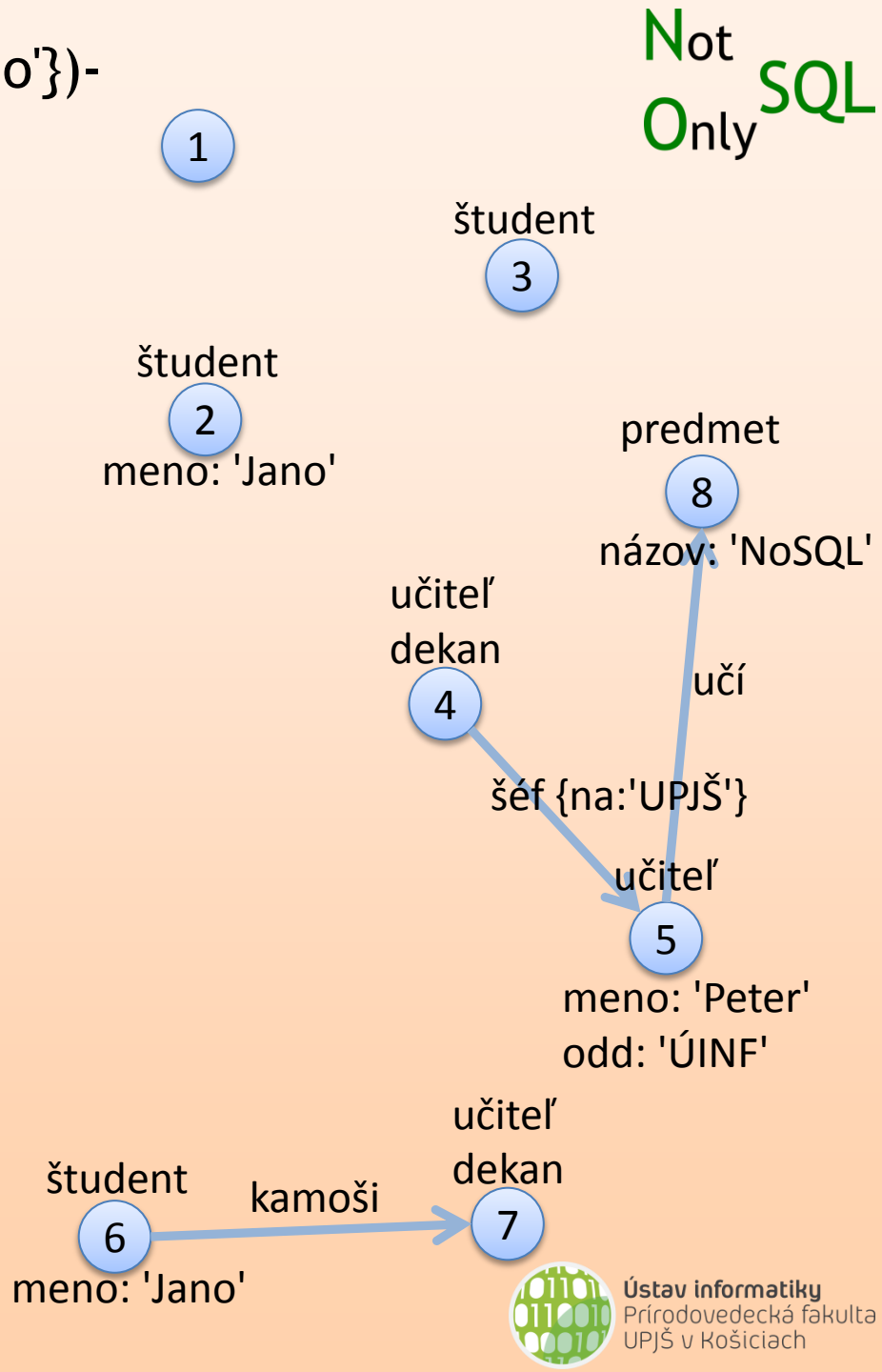
- CREATE (n:študent {meno: 'Jano'})-[:kamoši]->(m:učiteľ:dekan)
- MATCH (n {meno:'Peter'})
  MERGE (n)-[:učí]-> (x:predmet {názov:'NoSQL'})



Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

- CREATE (n:študent {meno: 'Jano'})-[:kamoši]->(m:učiteľ:dekan)
- MATCH (n {meno:'Peter'}) MERGE (n)-[:učí]-> (x:predmet {názov:'NoSQL'})
- MATCH (n:dekan), (m {meno: 'Jano'}) RETURN ID(n), ID(m);
  - vráti tabuľku:

ID(n)	ID(m)
4	2
7	2
4	6
7	6

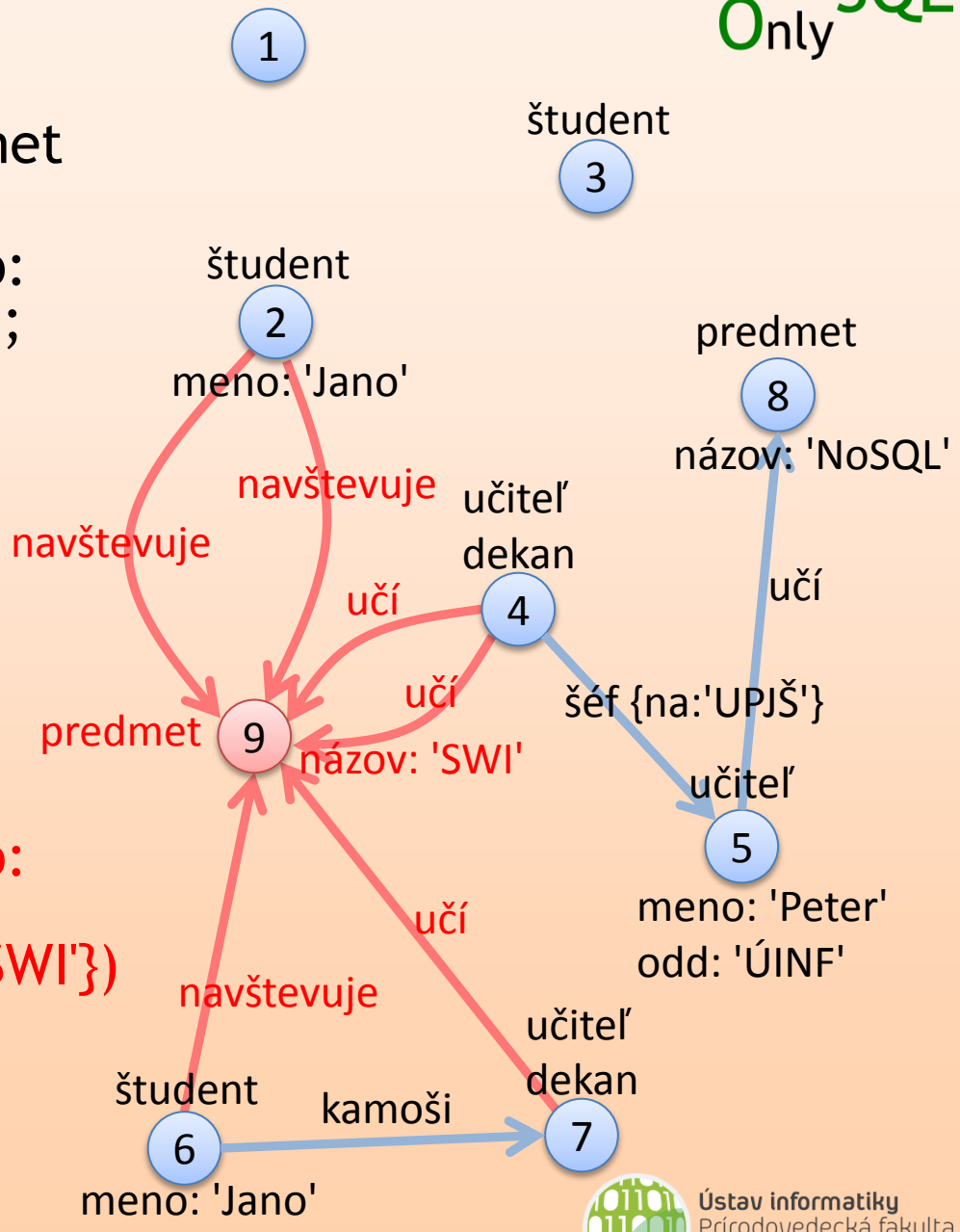


Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

- CREATE (n:študent {meno: 'Jano'})-[:kamoši]->(m:učiteľ:dekan)
- MATCH (n {meno:'Peter'}) MERGE (n)-[:učí]-> (x:predmet {názov:'NoSQL'})
- MATCH (n:dekan), (m {meno: 'Jano'}) RETURN ID(n), ID(m);
  - vráti tabuľku:

ID(n)	ID(m)
4	2
7	2
4	6
7	6

- MATCH (n:dekan), (m {meno: 'Jano'}) MERGE (x:predmet {názov:'SWI'}) MERGE (n)-[:učí]->(x)<-[:navštevuje]-(m)

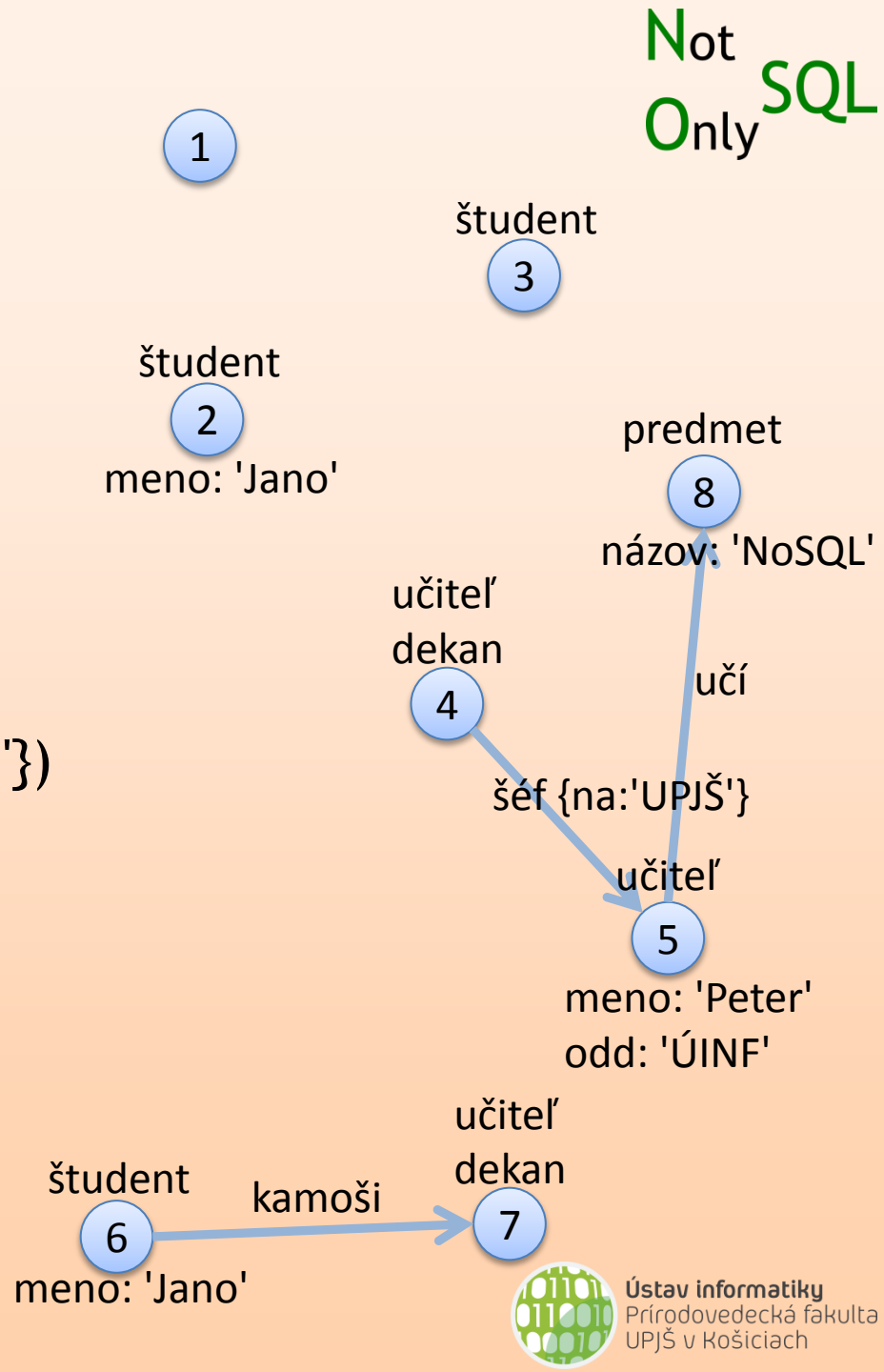


Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

- MATCH (n:dekan), (m {meno: 'Jano'}) RETURN ID(n), ID(m);
  - vráti tabuľku:

ID(n)	ID(m)
4	2
7	2
4	6
7	6

- MATCH (n:dekan), (m {meno: 'Jano'})
  MERGE (x:predmet {názov:'SWI'})
  MERGE (n)-[:učí]->(x)<-[:navštevuje]-(m)
- MATCH (n {názov:'SWI'})
  DETACH DELETE n

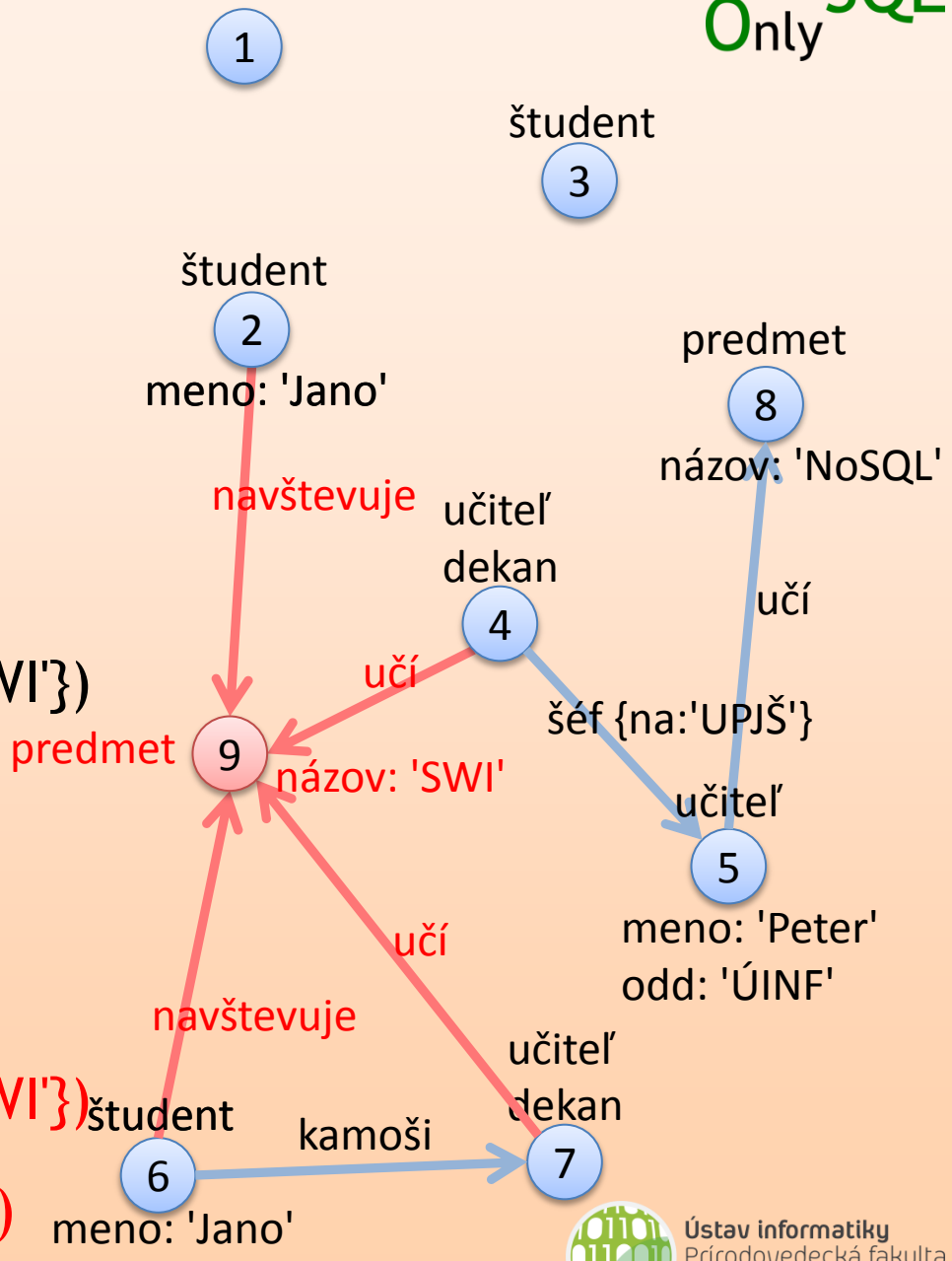


Ústav informatiky
Prírodovedecká fakulta
UPJŠ v Košiciach

- MATCH (n:dekan), (m {meno: 'Jano'}) RETURN ID(n), ID(m);
  - vráti tabuľku:

ID(n)	ID(m)
4	2
7	2
4	6
7	6

- MATCH (n:dekan), (m {meno: 'Jano'})
  MERGE (x:predmet {názov:'SWI'})
  MERGE (n)-[:učí]->(x)<-[:navštevuje]-(m)
- MATCH (n {názov:'SWI'})
  DETACH DELETE n
- MATCH (n:dekan), (m {meno: 'Jano'})
  MERGE (x:predmet {názov:'SWI'})
  MERGE (n)-[:učí]->(x)
  MERGE (x)<-[:navštevuje]-(m)



Užitočné linky

- Java API:
  - https://neo4j.com/docs/java-reference/current/tutorials-java-embedded/
- Jazyk Cypher:
  - https://neo4j.com/docs/developer-manual/current/cypher/
- OGM – objektovo grafové mapovanie
  - https://docs.spring.io/spring-data/data-neo4j/docs/current/reference/html/
  - https://neo4j.com/developer/spring-data-neo4j/

Otázky?