**Vizualizace rozsáhlých grafových dat na webu**

Aktualni rozsah je 17 1800 znakovych normostran bez obrazku, s temi by to tedy melo vyjit na zhruba 22 normostran, realne v latexu diky mensim strankam to predpokladam bude pres 25 stranek, jiz pocet stran v tomto dokumentu vychazi na 23 – nejsem si ale jist strukturou teto verze

predpokladam vsak, ze tento rozsah je dostacujici

obrazky v tomto dokumentu jsou pouze ilustrativni a budou prekresleny do specificke formy, ktera vyhovuje textu, text take poupravim, aby v nem byly fraze, ktere na obrazky odkazuji

Nejasnosti:

Kolegove resili, ze do 15.1 musi byt hotova i prezentace k obhajobe – musi se tedy nekam odevzdat nebo posilat komisi jiz v tomto terminu nebo je mozne na ni pracovat az do dne obhajoby a prinest ji tam osobne?

Musi tento dokument obsahovat veci jako je abstrakt, podekovani a vlozene zadani (to ani fyzizky momentalne nemam, abych je naskenoval) jak u finalni verze? Zatim prepokladam, ze struktura bude uvodni stranka-obsah-uvod-telo-zdroje

Kdyz mam v textu ukazky kodu, museji byt vlozeny textove, nebo je mozne je vlozit pomoci obrazku? To by mi ulehcilo zvyrazneni syntaxe

Kdyz pouzivam v textu napriklad nazev konkretni tabulky nebo hodnoty, mel bych pouzit uvozovky, kurzivu nebo pripadne zadne zvyrazneni?

V tabulce "users" vidime ….

V tabulce *users* vidime ….

V tabulce users vidime ….

V nejakych castech jsem se inspiroval strukturou nebo prelozil informace z anglicke z wikipedie, muze se vsak uvest do zdroju?

Naopak nekde jsem pouzil primo uryvek z ceskeho zdroje, to predpokladam neni problem pokud uvedu zdroj.

Predem dekuji za zpetnou vazbu

1. Úvod

Grafové databáze v poslední době nabývají na popularitě, hlavně díky schopnosti modelovat objekty z reálného světa v jejich přirozenější formě. Takové příležitosti se naskytují například na sociálních sítích, kde velice dobře reprezentují propojení jednotlivých osob mezi sebou. To vede k efektivnější práci s takovými daty. K tomu poskytují lepší vizualizaci, než jejich negrafové protějšky.

V této práci se budeme zabývat výkonem a vizualizací rozsáhlých grafových dat ve webových prohlížečích. Pro uložení dat budeme využívat formát RDF (Resource Description Framework), kde jsou data uloženy ve formě trojic. Ty mají strukturu subjekt-predikát-objekt a reprezentují propojení dvou uzlů hranou v grafu. Tuto funkcionalitu nám poskytne implementace rdf4j, která je napsána v Javě. Ta poběží na lokálně vytvořeném serveru pomocí aplikace Apache. Pro manipulaci s touto databází využijeme grafový dotazovací jazyk SPARQL, který je de facto standardem v této kategorii. Komunikaci mezi serverem a webovým prohlížečem bude zajišťovat RDF API. To je obsaženo v rdf4j implementaci, vychází z běžně používaného REST API a zajistí nám velice pohodlnou komunikaci mezi serverem a klientem. Samotný klient je poté implementován v jazyce JavaScript s využitím prvku <canvas>. Ten nám umožní dynamické vykreslovaní zpracovaných grafových dat na obrazovku v reálném čase.

digram vyobrazujici strukturu komunikce mezi databázi a webovou strankou

Vizualizace je interaktivní a primárně založena na principu force-directed graph (vykreslení grafu založené na silách) . Databáze bude testována na různých objemech dat, ty se ale budou pohybovat v řádech milionů položek a budou vytvářeny pomoci externího generátoru. Testování bude probíhat v různých prohlížečích s různými databázemi a bude mít za cíl objektivní zhodnocení uživatelský komfortu a výkonu, které jsou aktuální prohlížeče při vizualizaci schopny poskytnout.

2 Grafové databáze

Grafová databáze je databáze, která pro dotazování a ukládání používá grafové struktury s uzly, hranami a dalšími vlastnostmi pro reprezentaci uložených dat. Základ tvoří graf – ten poskytuje vazbu mezi datovými položkami a uzly. Hrany grafu poté reprezentují vztahy mezi nimi. To umožňuje, že data jsou propojena přímo mezi sebou a ve většině případech je možné získat tyto data jednou operací. Propojení je nejdůležitější součást grafové databáze a protože jsou v ní přímo uloženy, jsou dotazy na ně velice efektivní. Grafové databáze navíc umožňují jednoduchou a intuitivní vizualizaci, což je užitečné pro analýzu i silně propojených dat.

Grafové databáze, jsou součástí databázové třídy NoSQL, které se snaží vyřešit nebo omezit limitace existujících relačních databází. Vztahy jsou v grafových databázích mezi daty explicitně definovány, na rozdíl od dalších NoSQL návrhů a relačních databází, u kterých jsou vztahy definovány implicitně. To má výhodu především v práci s komplikovanými hierarchickými strukturami, které se dají jen obtížně modelovat v ostatních typech databázových systémů. K tomu se k datům z těchto struktur dá jednoduše a rychle přistoupit. Grafové databáze jsou podobné k síťovým databázím. Podobají se tím, že reprezentují data pomocí obecných grafů, liší se však mírou abstrakce a obtížností průchodu grafem po hranách.

Princip uložení dat v grafových databázích může být realizován několika různými způsoby. Některé závisejí na již existujícím relačním databázovém systému (enginu), ty ukládají grafové data v tabulce (tabulka sama o sobě představuje logický objekt, a proto si tento přístup vynucuje další úroveň abstrakce mezi grafovou databází, systémem řídící databázi a fyzickými zařízeními, na kterých jsou samotné data uloženy).

Jiné používají pro ukládání dat dokumentové databáze nebo databáze typu klíč-hodnota (key-value), z čehož následně vyplývá, že implicitně spadají do modelu NoSQL. Většina takových grafových databází také přidává koncept značek a vlastností, což umožňuje snazší kategorizaci prvků a tedy jejich mnohem rychlejší načítaní ve velkých skupinách.

Načítání dat z grafové databáze potřebuje dotazovací jazyk jiný než SQL, který byl navržen pro manipulaci s daty v relačních systémech a tedy nemůže být ve své standardní formě jednoduše použit pro průchod grafem. Aktuálně zatím neexistuje univerzální standard pro grafové dotazovací jazyky, který by měl stejné postavení, jako má SQL pro relační databáze. Je tedy vytvořeno několik implementací, z nichž většina se vztahuje k jednomu konkrétnímu produktu. To ale neznamená, že snaha o standardizaci neexistuje. Standardizační snahy vedli k vytvoření víceúčelových dotazovacích jazyků, jako je Gremlin, SPARQL a Cypher. Kromě toho, že grafové databáze disponují dotazovacím rozhraním, umožňují některé implementace i přístup přes aplikační rozhraní (API).

Grafové databáze přitáhly velkou pozornost na přelomu tisíciletí díky úspěchu, který s nimi zaznamenali velké počítačové firmy. Tyto implementace ale byly proprietární, což následně vedlo k rozmachu open-source implementací.

2.1 Struktura

Grafové databáze vyobrazují data v koncepční podobě. To je realizováno převedením dat do uzlů a vztahů do hran. Graf v grafové databázi je založen na konceptech grafové teorie. Je to množina objektů - uzlů a hran.

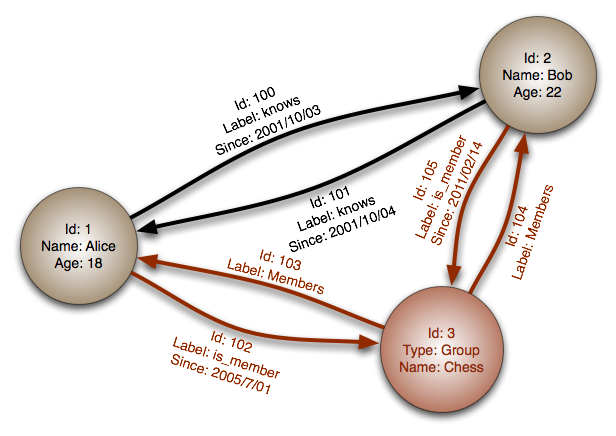
Uzly reprezentují entity nebo instance, jako osoby, firmy, účty nebo jakoukoliv jinou věc, o které se má udržovat záznam. Obecně můžeme říci, že uzel je ekvivalentní záznamu nebo řadě v relační databázi nebo dokumentu v dokumentové databázi.

Hrany naproti tomu reprezentují vztahy mezi nimi. Mohou být orientované nebo neorientované. V neorientovaném grafu jedna hrana reprezentuje jednu vlastnost, naproti tomu v orientovaném grafu mohou hrany s různou orientací reprezentovat naprosto rozdílné vlastnosti. Hrany jsou klíčovým konceptem v grafových databázích , protože do modelu přidávají abstrakci, která v relačních nebo NoSQL databázích neexistuje.

Každý uzel a hrana poté mají svoje vlastnosti, což jsou data, které je popisují. U uzlů to může například být jméno nebo věk, pokud uzel reprezentuje osobu. Vlastnosti hran poté konkrétně popisují vztah, který hrana reprezentuje. Dvě osoby mohou být ve vztahu s názvem "rodinný příslušník" nebo osoby a škola mohou být ve vztahu pojmenovaném "student".

V grafových databázích je propojení entit jasně viditelné a tedy pozorování dat, které jsou takto reprezentovány umožňuje lepší a efektivnější identifikaci struktur, které obsahují.

Jednoduchy diagram obecne grafove databaze jako napr



2.2 Vlastnosti

Grafové databáze jsou výkonný nástroj pro dotazy, které implicitně vychází z datové struktury. Pokud chceme například zjistit nejkratší vzdálenost mezi dvěma uzly v grafu, vypočítat jeho průměr nebo provést analýzu provázanosti několika uzlů, můžeme tak učinit pomocí několika jednoduchých a přirozených dotazů, které přímo těží z grafové struktury.

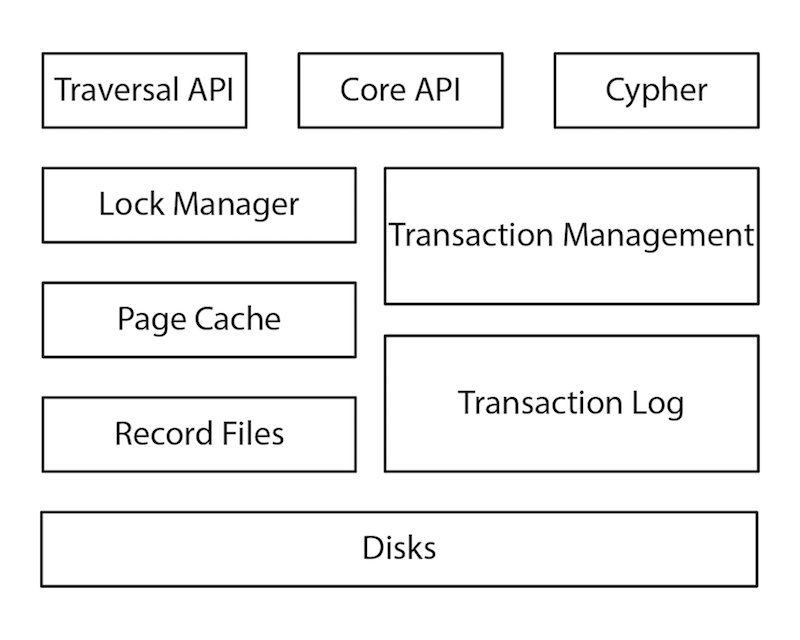
Grafy jsou k tomu vysoce flexibilní, což umožňuje uživateli vkládat nová data do existujícího grafu bez ztráty jakékoliv již existující aplikační funkcionality. Tato vlastnost se také pozitivně odráží v obtížnosti návrhu databázových systémů, protože není nutné znát přesnou strukturu předem. V relačních databázích je nutné přesně navrhnout počet tabulek a jejich obsah a vztahy, podle čeho se také bude silně odvíjet implementace dalších částí, které budou tuto databázi využívat.

Naproti tomu v grafových databázích stačí navrhnout pouze hlavní části a další specifické vlastnosti mohou být přidány bez jakékoliv změny původní struktury.

2.2.1 Uložení

Jak již bylo řečeno v úvodní kapitole, ukládací mechanizmy se různí. Mohou být buď založeny na již existujících implementacích, nebo vystavěny od nuly s maximální podporou grafové architektury (dále nativní grafové databáze) a tou se budeme zabývat.

Nativní grafové databáze jsou navrženy tak, aby dosáhly co nejvyššího výkonu při obecném průchodu grafem. Pro ukázku se podívejme na strukturu zástupce této kategorie: Neo4j .

Preloyit obrazek vychazjici z tohoto

Grafová data jsou uložena v v logických částech, z nichž každá reprezentuje specifickou část grafu – jako uzly, hrany a vlastnosti. Toto rozdělení je jeden z hlavních principů, který urychluje průchod. Každý uzel obsahuje pouze ukazatele seznamy vlastností a vztahů, což ho dělá velice robustním. Grafový systém dále obsahuje dedikované API pro průchod grafem a cache pro rychlé znovupoužití již načtených dat. Zbylé části jsou podobné s relačním systémem.

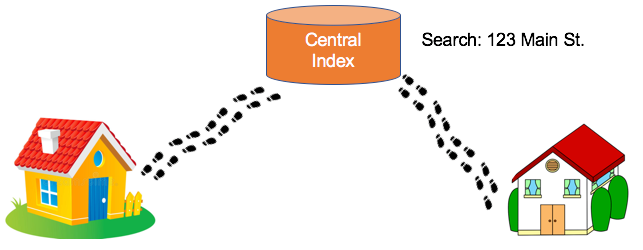
### 2.2.2 Index-free adjacency

Index-free adjacency je princip, který zajišťuje to, že všechny uzly přímo odkazují na všechny svoje sousedy. Jedná se o jeden z pilířů grafových databází a předpokládá se, že všechny nativní implementace jej splňují. Pro srovnání v relačních databázích se jakékoliv indexy na databázové prvky získávají od nějakého mezisystému – prostředníka.

Tento princip zapříčiňuje, že čas potřebný pro zpracování dotazu není proporcionální k objemu dat, ale pouze k té části grafu, která byla prohledána. To zbraňuje zhroucení rozsáhlých grafových databází. Je to také důvod, proč jsou grafové databáze tak dobré pro procházení grafů, což je dále umocněno snahou tento princip implementovat fyzicky, tedy tak, že indexy v databázi budou zastoupeny fyzickými adresami v RAM paměti.

Prekreslit dva obrazky tohoto typu





https://neo4j.com/blog/native-vs-non-native-graph-technology/

https://thomasvilhena.com/2019/08/index-free-adjacency

https://medium.com/@dmccreary/how-to-explain-index-free-adjacency-to-your-manager-1a8e68ec664a

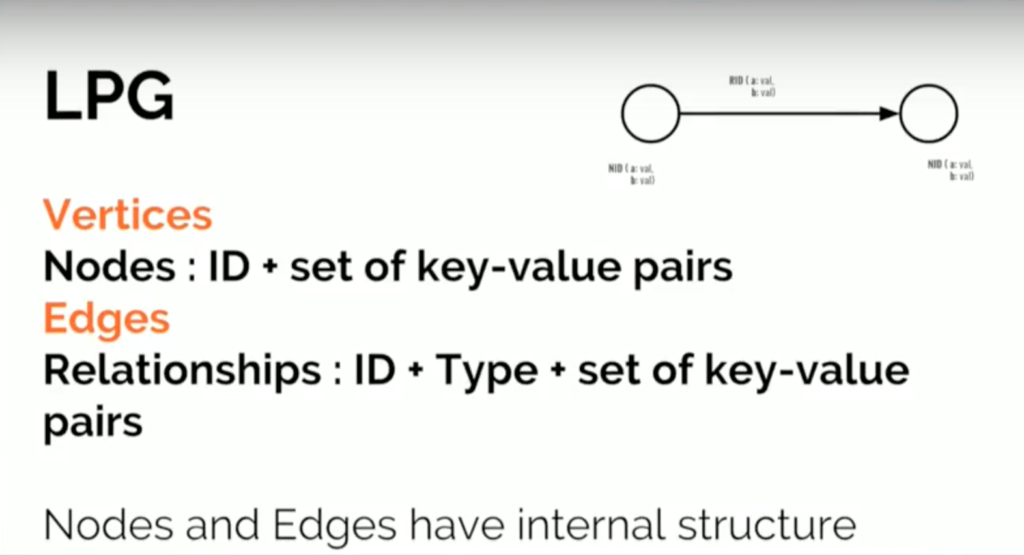
2.3 Grafové modely

Existují dva hlavní grafové modely – labeled propery graph a triplestore.

Labeled property graph (dále LPG) se zaměřuje primárně na ukládání a dotazování, naproti tomu modely typu triplestore jsou více orientované na efektivní provádění změn v datech. Pod modely typu triplestore spadá specifikace Resource Description Framework (dále RDF) spravovaný konsorciem W3C, který je de facto standardem této kategorie. Tento model budeme nazývat RDF store a budeme se jím dále zabývat v této kapitole a později ho využijeme v naší implementaci.

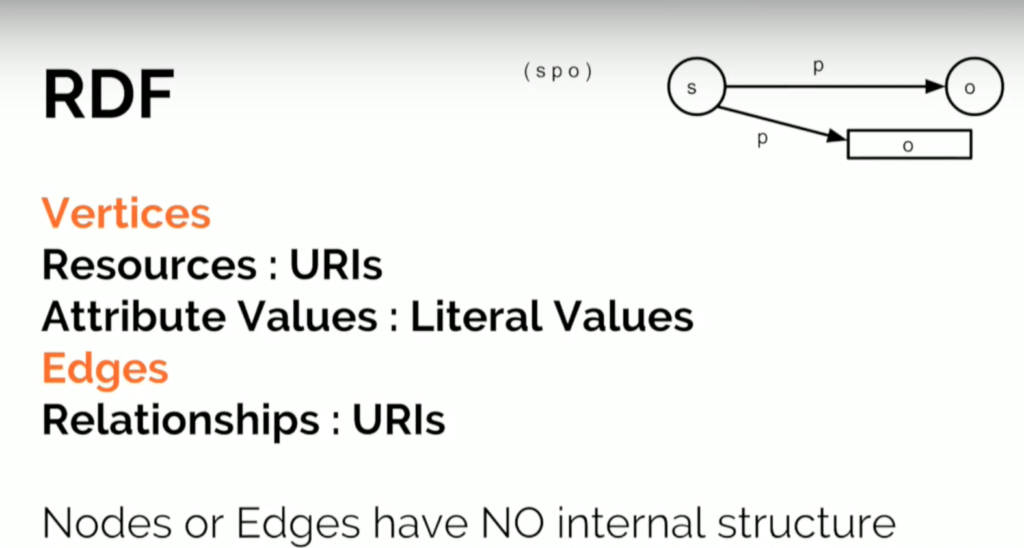
V LPG mají všechny uzly a hrany unikátní identifikátor a množinu vlastností, které je popisují. Unikátnost je důležitá, aby šlo každý prvek jednoznačně identifikovat. Dále je se všemi prvky spojena interní struktura, ve které jsou její vlastnosti obsaženy.

Preformulovat



Naproti tomu RDF model je založen na trojicích a existují v něm dva různé typy: objekt a literál. Oba se však v grafu budou jevit jako uzel. To tedy znamená, že na rozdíl od LPG, uzly ani hrany nemají interní strukturu, ale jsou pouze definovány unikátní značkou přímo hodnotou u literálů.

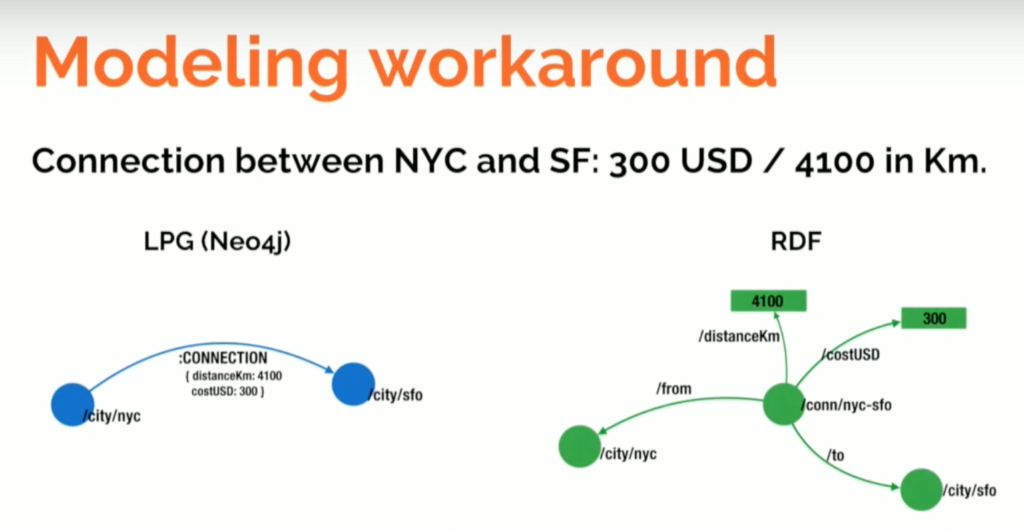
Preformulovat



Další rozdíl spočívá v tom, že RDF neumožňuje unikátně specifikovat vztahy stejného typu. Řekněme že existují entity osoba a museum - poté je v LPG můžeme několikrát propojit stejně pojmenovaným vztahem "navštívil". To nám potom umožní dotázat se na jeho kardinalitu. V RDF se všechny další stejně pojmenované vztahy budou ignorovat a neposkytnou žádné další informace.

V poslední řadě je se zaměřme na důsledky toho, že RDF neposkytuje žádné interní struktury pro vztahy. Představme si, že by jsme chtěli reprezentovat jízdu vlakem z Prahy do Brna, která stojí 1000 Kč a má vzdálenost 200km. Protože tento vztah obsahuje několik vlastností, povede to k topologicky velice rozdílným grafům, jak můžeme vidět na obrázku

predelat



V RDF je tedy nutné všechny násobné vlastnosti rozdělit do jednotlivých objektových a literálových uzlů.

<https://dzone.com/articles/rdf-triple-stores-vs-labeled-property-graphs-whats>

2.4 Porovnání s relačními databázemi

Od osmdesátých let minulého století, kdy byly položeny základy relačního databázového modelu jsou relační databáze de facto standardem pro ukládání velkých objemů dat. Mají ale svoje nevýhody – vyžadují striktní schéma a nutnost normalizace dat vytváří omezení na typy vztahů, na které je možno se dotázat. Také zvyšující se objem dat, který je nutno zpracovávat způsobuje problémy v relačních databázích.

Hlavním důvodem pro normalizaci dat v relačních databázích je podpora vlastností ACID (atomicita, konzistence, izolovanost, trvalost) u transakcí. Normalizace vede k odstranění duplikátů z databáze a zachování datové konzistence. To ve výsledku vyústí v to, že data jsou rozdělena do velkého počtu tabulek. Tento přístup byl zaveden k dosažení rychlých přístupů po řádcích, problémy však nastanou při vytváření komplexních vztahů mezi uloženými daty. Přestože takové vztahy mohou být analyzovány v relačním modelu, vede to však k vytvoření velice složitých dotazů, které ve výsledku povedou k provedení mnoha join operací přes různé atributy a tabulky. Další věc, která se musí zohlednit je omezení vycházející z cizích klíčů, což vede k z dalšímu zpomalení komplexních dotazů.

Grafové databáze tedy v porovnání s relačními databázemi nabízejí rychlejší práci s daty , které se dají dobře reprezentovat asociativními datovými strukturami a obvykle poskytují přirozenější mapování do struktur aplikací založených na objektově orientovaném modelu. Dále disponují lepší škálovatelností a ani u velkých datových sad nepotřebují pro dotazování join operace, které jsou jak již bylo řečeno velmi drahé obzvláště u rozsáhlých tabulek. Grafové databáze také méně závisejí na předem daném schématu a dají se snadno modifikovat a rozšiřovat a to je dělá vhodnými pro aplikace, které pracují s často měnícími se daty nebo schématy.

Naproti tomu, relační databáze obyčejně poskytují lepší výkon při opakování jedné operace nad rozsáhlými daty díky vhodnější formě jejich uložení.

Přestože grafové databáze zaznamenali v poslední době nárůst popularity oproti relačním, neměl by grafový model sám o sobě sloužit jako důvod pro nahrazení nových nebo již existujících relačních databází. Grafové databáze by se měli brát v úvahu pokud existují konkrétní důvody, které budou mít za následek snížení časové složitosti nebo latence pro daný systém.

2.4.1 Příklady – cervena barva se nahradi kurzivou nebo uvozovky???

Po teoretickém srovnání vlastností grafových a relačních databází se nyní podíváme na konkrétní instance pro lepší ilustraci.

Vezměme v úvahu následující situaci: chceme vyhledat všechny uživatele z tabuky users, jejichž telefonní číslo začíná předčíslím "420". V relačním systémech by toho bylo realizováno nejprve prohledáním všech záznamů v tabulce, která obsahuje telefonní čísla. U každého pak ověříme, zda obsahuje požadované předčíslí. Tento krok je obzvláště v tabulkách s velkým počtem sloupců časově náročný, proto relační databáze zavádí indexování. Data jsou uložena v menších podtabulkách, které obsahují pouze vybraná data a unikátní klíč. Pokud jsou telefonní čísla indexovány, tak hledání proběhne pouze nad touto menší tabulkou, ze které se získají klíče vyhovujících záznamů. Poté proběhne jejich vyhledání v hlavní tabulce. Tabulky jsou obyčejně fyzicky uloženy v takovém formátu, aby indexování bylo co nejefektivnější.

Jak již bylo řečeno v teoretické části, relační databáze sami o sobě neobsahují pevné vztahy mezi záznamy, místo toho jsou data provázány uložením klíče záznamu z jedné tabulky v jiné tabulce. Vezměme v úvahu tabulku emails, která obsahuje seznam emailových schránek. Taková tabulka pak může obsahovat sloupec userPK, který obsahuje primární klíč z tabulky users.

Aby bylo možné spojit uživatele a jejich emailové schránky, systém nejdříve musí nejdříve načíst primární klíče vybraných uživatelů, ty pak vyhledat ve sloupci userPK v tabulce emails a vyextrahovat jejich informace. Tato join operace je časově náročná a při provádění komplikovaných dotazů se pravděpodobně bude muset provést vícekrát, což způsobí multiplikativní nárůst složitosti. Nakonec je nutné výsledky seřadit a přerovnat do formátu jaký požaduje dotaz.

Grafové databáze naproti tomu explicitně ukládají vztahy mezi záznamy. Místo toho aby emailová schránka byla nalezena přes cizí klíč v tabulce emails, záznam o uživateli přímo obsahuje ukazatel, který přímo ukazuje na záznam emailové schránky. Tedy jakmile máme načtený záznam o uživateli, je možné z něj přímo přejít na jakýkoliv jiný s ním spojený záznam, bez nutnosti jakéhokoliv vyhledávání. Tím odstraníme nutnost použití join operací.

Teď můžeme spojit oba předchozí příklady do jednoho a podívat se na jeho provedení. Mějme dotaz, který chce vybrat všechny emailové adresy, které patří uživatelům s předčíslím "420". Databázový systém, nejprve provede konvenční vyhledání všech vyhovujících uživatelů, ale poté získá všechny emailové schránky náležící těmto uživatelům přes přímé vazby mezi nimi a tím se vyhne kontrole porovnání všech položek v tabulce emails. Zde je tedy přímo patrné urychlení, které grafové databáze nabízí.

Znatelné navýšení výkonu však nastane ve chvíli, kdy vyhledání, které jsou nutné v dotazu provést zahrnuje entity, které jsou od sebe vzdáleny o dvě a více úrovní. Řekněme například, že emailové schránky jsou nabízeny několika poskytovateli. Pokud by jsme teď chtěli najít všechny uživatele s emailovými schránkami od konkrétního poskytovatele, zabralo by to relační databázi dvě join operace spolu s další režií na formátování výsledku.

V grafové databázi vyhledáme validní uživatele, a poté dle daných vztahů nejprve přejdeme do tabulky emails a z té následně do tabulky obsahující provozovatele. Vždy se pracuje s daty, které jsou ve v vztahu s množinou uživatelů, tedy pouze se zlomkem celkového objemu dat. Z této množiny poté odfiltrujeme výsledky, které nevyhovují. Pokud provedeme analýzu časové složitosti, tak zjistíme, že všechny kroky provedení tohoto dotazu spadají do logaritmické nebo konstantní časové složitosti. Tedy celková složitost takového dotazu je tedy logaritmická.

Vlastnosti prvků v grafové databázi poskytují další vrstvu abstrakce, která vylepší efektivitu dotazů. Řekněme, že existuje škola, do které chodí osoby. Poté osobám můžeme přiřadit vlastnosti, jako je "učitel", "žák", "školník"a další. To poté umožní rychlejší vyhledání například všech učitelů nebo žáků. Vlastnosti však můžeme přiřadit i vztahům. Vztah mezi učitelem a žákem může být pojmenován "učí", což zase umožní například rychlejší vyhledání všech žáků, které učitel vyučuje. Ekvivalentní SQL dotaz by k realizaci těchto úkonů potřeboval načíst data z tabulky propojující žáky a učitele. I když vlastnosti prvků zvyšují efektivitu dotazů, jejich hlavní výhoda je přidání sémantiky pro konečné uživatele.

Relační databáze je tedy výhodnější zvolit pro data s plochou strukturou a s entitami položenými blízko u sebe, naproti tomu grafové databáze jsou vhodné pro hustě propojená data. To je v této době dělá populárními například pro ukládání uživatelů na sociálních sítích, kde velice efektivně modelují jejich vztahy mezi sebou.

Jako finální demonstraci provedeme srovnání jednoduchých dotazů v grafových a relačních databázích. Předpokládejme, že existují dvě tabulky: people, která obsahuje sloupce person\_id, person\_name a friend se sloupci friend\_id, person\_id(cizí klíč z tabulky people). Nyní provedeme dotaz na všechny Karlovi přátele.

Standardní relační dotaz v jazyce SQL:

SELECT p2.person\_name

FROM people p1

JOIN friend ON (p1.person\_id = friend.person\_id)

JOIN people p2 ON (p2.person\_id = friend.friend\_id)

WHERE p1.person\_name = 'Karel';

Cypher - grafový dotazovací jazyk :

MATCH (ee:person)-[:FRIEND-WITH]-(friend)

WHERE ee.name = "Karel"

RETURN ee, friend

SPARQL – rozšířený grafový dotazovací jazyk pro RDF grafovou databázi standardizovaný W3C:

PREFIX ex: <http://example.com/>

SELECT ?name

WHERE { ?s ex:name "Karel" ;

ex:knows ?o .

?o ex:name ?name .

}

SPASQL – hybridní dotazovací jazyk, který rozšiuje SQL pomocí SPARQL:

SELECT people.name

FROM (

SPARQL PREFIX ex: <http://example.com/>

SELECT ?name

WHERE { ?s ex:name "Karel" ;

ex:knows ?o .

?o ex:name ?name .

}

) AS people ;

prepsat do latexu se spravnnym zvyraznenim syntaxe

3. Resource Description Framework

Resource Description Framework (dále RDF) , česky systém popisu zdrojů je datový model standardizování konsorciem W3C . Poskytuje obecnou metodu pro popis dat definováním vztahů mezi datovými objekty. Byl navržen v roce 1999 a specifikace 1.0 byla publikována v roce 2004. Právě tento model budeme využívat pro reprezentaci grafových dat v tomto projektu.

RDF datový model je založen na principu tvoření výroků o datech ve formě trojic, které mají formu subjekt – predikát – objekt. Subjekt označuje zdroj a objekt aspekty nebo vlastnosti zdroje. Predikát poté vyjadřuje vztah mezi subjektem a objektem.

Jako příklad můžeme uvést výrok "Honza měří 180 cm". V RDF trojici subjekt reprezentuje "Honza" a objekt "180 cm". Predikát "měří" poté definuje vztah mezi nimi. Jak můžeme vidět, tento přístup se výrazně liší od standardního modelu entita-atribut-hodnota. V něm by byl záznam uložen jako entita "Honza", atribut "výška v cm" a hodnota "180".

Kolekce těchto trojic se poté obecně nazývá triplestore - v našem případě konkrétně RDF store a reprezentuje značený orientovaný multigraf . To znamená, že databázový systém postavený na RDF, má všechny benefity a negativa, které plynou s teorie grafů a grafových databází, jaké již byly zmíněny v kapitole o grafovém modelu

3.1 Serializace

RDF poskytuje několik různých serializačních formátů, které zahrnují například:

RDF/XML – formát založený na XML, první standardizovaný formát pro RDF

JASON-LD – formát založený na JSON

Turtle – kompaktní formát, snažící se o jednoduchost čtení pro uživatele

N-triples – vychází z Turtle, je jednoduší pro rychlejší generování a čtení

RDF/XML bývá někdy také označován zkratkou RDF, protože byl prvním definovaným standardem, je však nutné nezaměňovat RDF/XML serializační formát se samotným abstraktním RDF modelem.

3.2 Identifikace

Subjekt v RDF trojici je buď prázdný uzel nebo je použit uniform resource identifier (dále URI). Oba tyto stavy reprezentují zdroj. Zdroje reprezentované prázdnými uzly se nazývají anonymní a nedají se přímo identifikovat. Predikát je reprezentován také přes URI a identifikuje zdroj reprezentující vztah. Objekt naproti tomu může mít tři nabývat třech stavů. První je opět URI, druhý je prázdný uzel a třetí je řetězcový literál. Ten představuje jedinou možnost pro popis vlastností

Ve webových aplikacích, kde je RDF populární například pro reprezentaci propojení osob na sociálních sítích, mohou RDF URI reprezentovat adresy, přes které se dá reálně přistoupit k datům na webu. Obecně ale platí, že RDF nejsou určeny pouze pro popis webových zdrojů, ale mohou nabývat jakýchkoliv hodnot a nemusí se přes ně dát nikam přistoupit. Proto se poskytovatelé i uživatelé musejí předem domluvit na jejich významu. Toto není přímo součástí RDF standardu, existuje však několik běžně používaných verzí. Jako příklad můžeme uvést URI,

http://example.com/bookstore#

která může označovat uzel reprezentující knihkupectví. URI je poté možné zkracovat například tak, že "http://example.com/", budeme zapisovat jako "ex:".

3.3 Dotazovací Jazyky

Jediným de facto požívaným dotazovacím jazykem je SPARQL, který doporučuje samotné W3C. Syntaxí SPARQL vychází z SQL a je navržen pro práci s grafovými databázemi. Jediný další zástupce, který stojí za zmínění je Versa, což je kompaktní jazyk nezávislý na SQL a implementovaný v Pythonu.

3.4 Příklad - lip strukturovat v latexu

Funkcinalitu RDF můžeme ilustrovat na jednoduchém příkladě, který popisuje osobu jménem Eric Miller.

Jednotlivci, například "Eric Miller" (subjekt) jsou identifikováni URI: "http://www.w3.org/People/EM/contact#me"

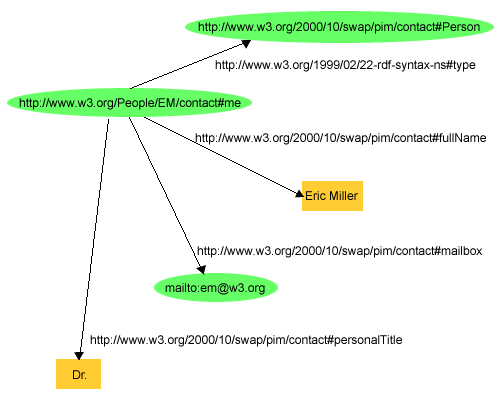
Druhy entit, například osoby jsou identifikovány URI: "http://www.w3.org/2000/10/swap/pim/contact#Person"

Vlastnosti těchto entit, například "název emailové schránky" (predikát) je identifikována URI:

"http://www.w3.org/2000/10/swap/pim/contact#mailbox"

Hodnoty těchto vlastností (objekty) jsou poté reprezentovány jednotlivými uzly. Objektový uzel označený URI "mailto:em@w3.org" popisuje konkrétní emailovou schránku, literální uzly následně obsahují řetězce znaků "Eric Miller" a "Dr." které reprezentují jméno a titul.

Asi nechat tento obrazek?



Příklad uložení v RDF/XML:

<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#" xmlns:eric="http://www.w3.org/People/EM/contact#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">

<contact:fullName>Eric Miller</contact:fullName>

</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">

<contact:mailbox rdf:resource="mailto:e.miller123(at)example"/>

</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">

<contact:personalTitle>Dr.</contact:personalTitle>

</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">

<rdf:type rdf:resource="http://www.w3.org/2000/10/swap/pim/contact#Person"/>

</rdf:Description>

</rdf:RDF>

introduction

https://www.w3.org/TR/rdf-primer/

4.SPARQL

SPARQL (čti "sparkl", zastupující rekurzivní akronym "SPARQL Protocol and RDF Query Language") je RDF dotazovací jazyk určený pro manipulaci a získávání dat v RDF formátu. Standardizuje ho konsorcium W3C a existuje několik implementací v různých jazycích. Navíc existují nástroje, které dokáží přeložit SPARQL dotazy do jiných dotazovacích jazyků, jako je například SQL.

Dotaz se skládá ze seznamů trojic (forma "subjekt-predikát-objekt", která vychází z RDF). Každý prvek této trojice je poté možno nahradit neznámou (v originále "wildcard"). Jako výsledek jsou potom vracený ty položky z databáze, které se shodují se vzorem definovaným v dotaze. K dispozici jsou také všechny běžně používané analytické a agregační funkce jako například SORT, SUM nebo MAX.

SPARQL definuje čtyři typy dotazů:

SELECT – Vrací nezměněná data z databáze, tedy formou tabulky

CONSTRUCT – Vrací data přetransformovaná do formátu RDF

ASK – Dotaz vracející hodnotu true nebo false

DESCRIBE – popisuje výsledek dotazu konstrukcí relevantního RDF grafu

Přesný popis formátu můžeme demonstrovat na jednoduchém dotazu typu SELECT, který má za úkol vrátit všechny hlavní města, která leží v Africe. Výsledek bude obsahovat názvy těchto měst a států, ve kterých leží.

PREFIX ex: <http://example.com/exampleOntology#>

SELECT ?capital

?country

WHERE

{

?x ex:cityname ?capital ;

ex:isCapitalOf ?y .

?y ex:countryname ?country ;

ex:isInContinent ex:Africa .

}

Příkaz začíná definicí prefixu, pomocí kterého zkrátíme URI a zpřehledníme tak zbytek dotazu. Poté následuje SELECT a seznam neznámých. Ty se označují symbolem "?". Náš příklad obsahuje neznámé "?capital" a "?country" – to budou výsledky, které nám dotaz vrátí a ve formě tabulky se dvěma sloupci. Nakonec dotaz obsahuje klíčové slovo WHERE. V jeho těle budou vloženy všechny vzory, které musí výsledky splňovat.

Pokud trojice končí středníkem, tak další trojice, které po ní následují nemusí obsahovat kompletní definici a prázdná místa budou doplněna z předchozí trojice. To znamená, že v našem příkladu má druhá trojice "ex:isCapitalOf ?y" stejný význam jako "?x ex:isCapitalOf ?y". Celý dotaz se poté dá formulovat čtyřmi trojicemi. Neznámá "?x" reprezentuje entitu město a "?y" entitu stát. První trojice vyjadřuje to, že u měst je nutné vrátit pouze vlastnost název a druhá poté specifikuje, že entita město musí být hlavním městem libovolného státu. Třetí opět vyjadřuje nutnost vrácení pouze vlastnosti název, tentokrát ale státu. Poslední pak zužuje výběr pouze na ty státy, které leží v Africe. Systém poté projde databází a vrátí ty prvky, které všechny vzory splňují.

SPARQL navíc obsahuje rozsáhlý soubor dalších specifikací, které definují další funkcionality mimo dotazování. Aktuální specifikace jazyka SPARQL poskytují například příkazy UPDATE, INSERT a DELETE, pomocích kterých se dají vkládat nové trojice nebo modifikovat již existující.

https://en.wikipedia.org/wiki/SPARQL

https://www.topquadhttps://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/rant.com/comparing-sparql-with-sql/

https://www.w3.org/TR/sparql11-query/#basicpatterns

<https://en.wikipedia.org/wiki/RDF_query_language>

5. HTML

HTML (Hypertext Markup Language) je v informatice název značkovacího jazyka používaného pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. HTML je hlavním z jazyků pro vytváření stránek v systému World Wide Web.

Jedná se o interpretační jazyk, to znamená, že zdrojový kód se nepřekládá do spustitelného tvaru, ale je prováděn interpreterem (webovým prohlížečem) na klientovi. Výhodou je, že kód je obyčejný textový dokument, který si můžeme prohlédnout a porozumět mu, a je nezávislý na hardwaru i operačním systému.

Jazyk HTML je tvořen elementy, což jsou objekty, které říkají něco o svém obsahu. Jméno elementu ohraničené ostrými závorkami < a > označujeme jako tag (značka). Může být párový nebo nepárový. Párový element má otevírací tag, např. <p> a uzavírací tag (před jméno elementu je předřazeno lomítko), např. </p>. Každý element může obsahovat určité atributy (parametry, které určují vlastnosti elementu). Atribut se zapisuje do otevíracího tagu vedle jména, ve formátu jméno\_atributu="hodnota". Tyto elementy mohou být v sobě vnořovány podle určitých pravidel (popsaných ve specifikaci jazyka - DTD).

HTML kód se standardně ukládá do souboru s příponou html. Na prvním řádku souboru se nachází definice DOCTYPE, poté následuje kořenový element <html>. Kdekoliv uvnitř dokumentu se může nacházet komentář. Uvnitř kořenového elementu se může nacházet pouze element <head>, který nese základní informace o dokumentu a element <body>, kde je vlastní obsah dokumentu. Základní struktura poté může vypadat takto:

<!-- toto je komentář; dále je definováno DTD HTML 4.01 Strict -->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<!-- kořenový element dokumentu s určením jazyka-->

<html lang="cs">

<!-- hlavička -->

<head>

</head>

<!-- tělo dokumentu -->

<body>

</body>

</html>

Elementy, které se při interpretaci dokumentu zobrazí jako jeho obsah, se vkládají do elementu body. Mohou být dvou základních obsahových typů – řádkové nebo blokové. Elementy se mohou do sebe vnořovat, nikdy se nesmí křížit. Do řádkového elementu není možno vnořit element blokový. Existují však také elementy, které se do výstupu nezobrazují. Většina se nachází uvnitř elementu <head>, jako <meta>, <style> a <link>, některé však i uvnitř body, jako třeba <script>.

https://www.samuraj-cz.com/clanek/zaklady-jazyka-html/

5.1 Tag <canvas>

Jedná se o paritní tag, který je speciálně určen pro vykreslování grafiky na webových stránkách. Ovládá se pomocí skriptovacího jazyka JavaScript a umožňuje dynamické změny po i po načtení stránky. Pokud takové změny provádíme v konstantních intervalech – například šedesátkrát za sekundu, můžeme vytvářet animace, čehož budeme využívat při vizualizaci grafové databáze. Jeho funkcionalita je dostupná ve všech běžně používaných prohlížečích.

Samotný tag může mít například takovýto tvar:

<canvas id="myCanvas" width="200" height="100"></canvas>

Ten vytvoří objekt canvas uvnitř webové stránky, jehož id je "myCanvas" a bude 200px široký a 100px vysoký. K tomuto objektu pak přistoupíme pomocí DOM struktury z jazyka JavaScript.

K tomu canvas obsahuje desítky již předimplementovaných funkcí, které pokrývají většinu základní funkcionality pro práci s 2D objekty. To zahrnuje vykreslování základních tvarů jako je bod, čára, kruh, mnohoúhelníky a další. U těch je poté měnit barvy výplní, ohraničení a jiné vlastnosti. Existuje zde i podpora pro vykreslování textu.

Par mensich obrazku toho co canvas umi

https://www.w3schools.com/html/html5\_canvas.asp

https://www.w3schools.com/tags/ref\_canvas.asp

<https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API>

<https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial>

6. REST API

REST (Representational State Transfer) je architektura, která umožňuje přistupovat k datům na určitém místě pomocí standardních metod HTTP. V tomto projektu ji budeme využívat pro výměnu dat mezi databází a webovou stránkou.

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.

REST implementuje čtyři základní metody, které jsou známé pod označením CRUD, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu.

GET (Retrieve)

Základní metodou pro přístup ke zdrojům je získání zdroje – metoda GET. Setkává se s ní každý uživatel webu – není to nic jiného než požadavek na stránku.

POST (Create)

Pro vytvoření dat slouží metoda POST, známá z HTML formulářů.

DELETE

Zdroj lze smazat pomocí volání URI HTTP metodou DELETE. Volání je obdobné jak u metody GET.

PUT (Update)

Operace změny je podobná operaci vytvoření (create, metoda POST), s tím rozdílem, že voláme konkrétní URI konkrétního zdroje, který chceme změnit a v těle předáme novou hodnotu

https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

//------------------konec casti, která se bude odevzdavat 15.1.

7. Implementace

7.1 JavaScript

7.1.1 DOM?

7.2 Apache

7.2 RDF4J

7.1.2 RDJF API

popis co se posila prez co