Hibridne entitetno-relacijske in grafne podatkovne baze

Blaž Celarc[1] **in Luka Šajn**[2]

[1]*NC2G Solutions, Senožeti 74i, 1262 Dol pri Ljubljani,*
[2]*Fkulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana*
E-pošta: blaz@nc2g.eu, luka.sajn@fri.uni-lj.si

Abstract

Traditional relational databases have certain limitations that have been effectively addressed by the introduction of various types of NoSQL databases. Among these, Graph databases stand out as a prime example, with their design principles rooted in graph theory. Over the past two decades, Graph databases have experienced a significant surge in both popularity and commercial availability. These databases are particularly noted for their distinctive qualities, which make them exceptionally well-suited for searching and manipulating data that is represented in the form of a graph.

Graph databases leverage the concepts of nodes, edges, and properties to model complex relationships and networks in a way that is both intuitive and efficient. This structure allows for highly optimized queries that can traverse intricate connections quickly, which is a notable advantage over traditional relational databases. While relational databases are highly matured and widely adopted across various industries, they lack the inherent capabilities to efficiently perform out-of-the-box searches through graph-like data. To achieve similar performance levels on equivalent data, relational databases require extensive, hands-on implementation on a per-query basis, often involving complex joins and subqueries that can be cumbersome and less efficient.

Despite these clear benefits, there was no standardized language for working with Graph databases until April 2024. Prior to this, various graph database vendors developed their own query languages and APIs, leading to fragmentation and a steeper learning curve for developers and data scientists. The introduction of a standardized language has been a significant milestone, promising to unify the ecosystem, simplify development, and accelerate the adoption of Graph databases across a wider array of applications.

1 Uvod

Teorijo za relacijskimi podatkovnimi bazami je prvič opisal Edgar Codd leta 1970[1]. Na področju relacijskih podatkovnih baz je prisotnih več ponudnikov [2], ki se razlikujejo v implementaciji in ponujajo različne funkcionalnosti in zmogljivosti po različnih licencah.

Razširjenost relacijskih podatkovnih baz je botrovala uvedbi namenskega standarda za jezik SQL (‘Structured Query Language’), ki služi kot standardni jezik za delo z relacijskimi podatkovnimi bazami[3],nad katerim bdi ISO/IEC[4]. Učinki standardizacije so vidni predvsem v razširjenosti relacijskih podatkovnih baz po združbah in po številu strokovnjakov usposobljenih za delo z njimi. Posamezni SUPB-ji nadgrajujejo standardni SQL z njim specifičnimi ukazi, ki omogočajo izvedbo njim specifičnih nestandardnih operacij.

Omejitve relacijskih podatkovnih baz sprožijo raziskave in razvoj sistemov, ki te omejitve naslavljajo. Pojavijo se NoSQL rešitve, ki ne temeljijo več izključno na relacijskem pristopu in ne zadoščajo SQL standardu[5]. Različne omejitve, ki so botrovale razvoju teh SUPB, rezultira v različnih pristopih pri hrambi in manipulaciji podatkov v teh sistemih.

Eden izmed pristopov je uporaba teorije grafov za modeliranje podatkovne baze in rezultat uporabe pristopa so grafne podatkovne baze. Te se pojavijo šele po letu 2000 in so relativno nove v primerjavi z relacijskimi podatkovnimi bazami. Obstaja več ponudnikov grafnih podatkovnih baz, ki podobno ponujajo različen nabor funkcionalnosti in zmogljivosti po različnih licencah[6]. Relativno nova razširjenost in popularnost grafnih podatkovnih baz je sprožila poskus standardizacije jezika za delo z njimi. ISO/IEC tako objavi prvi standard v aprilu 2024 za jezik GQL (‘Graph Query Language’) ISO/IEC 39075:2024[7], ki naj bi služil kot globalni standard za grafne SUPB.

O hibridni podatkovni bazi govorimo, ko združujemo vsaj dva SUPB-ja. Z uvedbo NoSQL rešitev so se hibridni sistemi podatkovnih baz pojavili praktično takoj, saj je del podatkov vsebovan v relacijski podatkovni bazi, del pa v namenski NoSQL podatkovni bazi. Težave se pojavijo pri organizaciji in delu s podatki, saj se moramo pri operacijah zavedati obstoja podatkov v ločenem sistemu. Posledično je zagotavljanje lastnosti ACID (atomarnost, konsistenca, izolacija, trajnost) otežkočeno oziroma nemogoče. Upoštevati je potrebno tudi dejstvo, da nekatere podatkovne baze teh lastnosti nimajo[8]. Sisteme za upravljanje podatkovnih baz lahko združimo na več različnih načinov, vodilo pri združevanju pa je doseg najoptimalnejše podatkovne baze, ki bo vsebovala kar največ pozitivnih lastnosti obeh pristopov in kar najmanj negativnih lastnosti ter pomanjkljivosti obeh SUPB.

Zaradi navedenih razlogov se odločimo za razvoj hi-

bridne grafno-relacijske podatkovne baze. Pristopov, ki jih lahko uberemo za arhitekturno zasnovo je mnogo in vsak izmed njih ima prednosti in slabosti. Če stremimo po optimizaciji skupne velikosti podatkovnih baz in optimizaciji časa potrebnega za pridobivanje določenih podatkov, bomo verjetno razmišljali v smer, ki v grafno podatkovno bazo seli le tiste podatke in strukture, ki lahko imajo grafno ureditev. V kolikor stremimo le po distribuiranem delovanju sistema in možnosti sočasnega izvajanja, bo izgled in obseg grafnega dela podatkovne baze bistveno drugačen.

En pristop (in ne nujno najboljši) pri načrtovanju hibridne podatkovne baze sledi spodnjim korakom:

1. Generiranje entitetno-relacijskega modela za celotno podatkovno bazo,
2. Identifikacija struktur, ki jih je moč optimalno preslikati v grafno podatkovno bazo,
3. Preslikava identificiranih struktur v grafno podatkovno bazo,
4. Identifikacija ključev, ki služijo povezavi med relacijskim in grafnim modelom,
5. Odstranitev nepotrebnih (podvojenih) podatkov iz relacijskega in grafnega modela.

Ta pristop se naslanja predvsem na dejstvo, da relativno lahko prepoznamo strukture, ki so lažje in bolj optimalno predstavljive v grafnem kot pa v relacijskem SUPB. Dober primer je entiteta, ki ima vsaj eno referenco sama nase. Logično gledano, nam taka entiteta predstavlja nek graf (naj bo to drevo ali pa že kar usmerjen graf).

2 Primer sledenja proizvodnji izdelkov

Za lažjo ilustracijo problematike poglejmo recimo podjetje, ki se ukvarja s proizvodnjo izdelkov. Podjetje vodi zalogo komponent iz katerih sestavlja polizdelke in izdelke. Pri sprejemu novih komponent v skladišče se zabeležijo vsi potrebni podatki, ki omogočajo identifikacijo proizvajalca, lota in morebitne serijske številke komponente. V proizvodnem procesu, se takšne komponente porabljajo za izdelavo izdelkov, ki jih prav tako beležimo kot komponente (v kolikor gre za končni izdelek, lahko le tega označimo, da omogočimo beleženje zaloge pripravljene za prodajo). S tem omogočimo popolno sledljivost posameznih komponent v nekem prodanem izdelku, kar med drugim omogoča zgodnje odkrivanje napak v kolikor pride do odpoklica vhodnih komponent zaradi napake (vemo, kaj smo iz teh komponent sestavili) oziroma odkrivanje skupnih imenovalcev pri reklamacijah, ki jih sprožijo kupci izdelanih izdelkov (katere komponente so skupne izdelkom, ki so bili reklamirani, kateri proizvodni procesi so skupni reklamiranim izdelkom).

2.1 Relacijska podatkovna baza

Tako funkcionalnost v relacijski podatkovni bazi predstavimo z entitetnim tipom *'Artikel'*, *'Vsebovan'*. Vsaka vhodna komponenta (pridobivanje zaloge) se zabeleži kot artikel, ki pa nima nobenih komponent. Vsak izdelan izdelek, se prav tako zabeleži v tabeli *'Artikel'*, komponente uporabljene pri izdelku pa v *'Vsebovan'*. V primeru MSSQL lahko entiteto *'Artikel'* predstavimo kot tabelo tipa *NODE* in entiteto *'Vsebovan'* kot tabelo tipa *EDGE*.

2.2 Grafna podatkovna baza

Grafna podatkovna baza podatke hrani drugače. Jedro grafnih podatkovnih baz so vozlišča, robovi (povezave) in lastnosti. Vsako izmed vozlišč ima eno ali več oznak. Za naš primer je smiselno za vsak artikel imeti vozlišče z oznako *Artikel*. Vsakemu izmed vozlišč lahko določimo opcijske dodatne oznake (komponente, polizdelek, končni izdelek). Razmerja med vozlišči določimo s povezavami med vozlišči. V Neo4J lahko povezave hranijo tudi dodatne podatke, ki bi v relacijski bazi predstavljale dodatne entitete (*'ArtikelKomponente'*) oz. atribute. Neo4j za poizvedovanje in manipulacijo s podatki uporablja jezik Cypher. Zagotovo bo zanimivo spremljati implementacijo GQL jezika v Neo4j, katerega avtorji pozdravljajo standardizacijo na področju grafnega poizvedovalnega jezika[9]. V primerjavi z relacijskimi SUPB-ji, Neo4j prav tako zagotavlja vse lastnosti ACID [10]

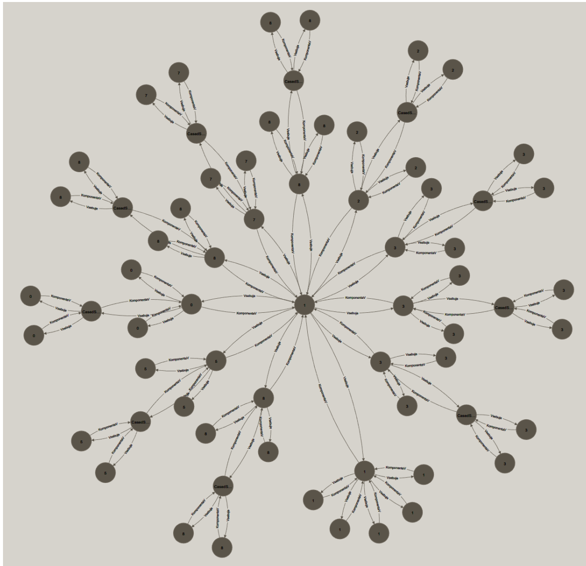
2.3 Testni podatki

Za lažjo ilustracijo poizvedb, vzemimo primer masovne proizvodnje meteoroloških postaj. Postaje ob dokončanju tovarniško testiramo po neki metodologiji, ki odkriva morebitne defekte. Pri izdelavi potrebujemo izdelana tiskana vezja (napajanje (PSU), krmiljenje (MB), 7 različnih senzorjev (SEN) in komunikacijo (RxTx)) in tudi njihova ohišja.

Vsaka izmed postaj uporablja vse tipe tiskanih vezij, torej držimo v inventarju 10 različnih tipov vezij, ki so dobavljeni v 10 različnih variacijah v 10 različnih lotih po 300 artiklov. Vsak tip vezja ima pripadajoče ohišje (spodnji del ohišja, pokrov in tesnilka). Vezje najprej namestimo v spodnji del ohišja in tak polizdelek pošljemo na naslednjo postajo, kjer se aplicira tesnilka in zgornji del ohišja. Sestavljanje vezij poteka vzporedno, kjer se ustvarja zaloga različnih tipov sestavljenih senzorjev. Ločeno poteka vstavljanje kombinacije vezji v spodnji del ohišja vremenske postaje in povezovanje z namenskim komunikacijsko-napajalnim kablom. Kot zadnji korak se namesti tesnilka vhoda in tesnilka pokrova ter pokrov in kreiranje zapisa za izdelano postajo. V tem testnem scenariju imamo skupno 30.000 izdelanih postaj, vseh vozlišč je 2.010.000, ki imajo 1.980.000 robov. Vsaka izmed postaj sestoji iz 67 komponent, ki so medsebojno povezane z 66 povezavami tipa *'Vsebuje'* in 66 povezavami tipa *'KomponentaV'*. Izvorna koda za generiranje testnih podatkov in struktur je na voljo na naslovu https://nc2g.eu/erk2024[11].

2.4 Poizvedbe nad podatki

Za ilustracijo razlik med zgornjimi podatkovnimi bazami spišimo poizvedbo, ki nam odgovori na vprašanje, koliko postaj ima vgrajeno napajalno enoto variacije 1 v kombinaciji s kontrolno enoto variacije 2 in komunikacijsko enoto variacije 3?



Slika 1: Grafna predstavitev komponent in povezav v individualni postaji

2.4.1 *Relacijski pristop*

V vsakem primeru smo prisiljeni v implementacijo neke vrste rekurzivnega algoritma, ki bo ugotovil kdaj (in kje) v strukturi se pojavijo določene komponente. Velja ome-

```
DECLARE @TIP_ARTIKLA INT = 11, @VARIACIJA INT = 2;
WITH IskalnoDrevo AS (
    SELECT FKArtikelParentId, FkArtikelKomponentaId from Vsebovan WHERE
    FKArtikelKomponentaID IN (SELECT ArtikelId from dbo.Artikel WHERE
        FKArtikelTipId = @TIP_ARTIKLA AND variacija = @VARIACIJA) UNION ALL
    SELECT V.FKArtikelParentId, V.FKArtikelkomponentaId from Vsebovan V
    JOIN IskalnoDrevo PT ON v.FkArtikelKomponentaId = PT.FKArtikelParentId
) SELECT DISTINCT FKArtikelParentId FROM IskalnoDrevo
WHERE FkArtikelParentId IN (Select ArtikelId from Artikel WHERE Final = 1)
```

Slika 2: Rekurzivna poizvedba

niti, da zgornja poizvedba vrača le tiste postaje, ki vsebujejo le eno kombinacijo tipa in variacije komponente. Če želimo vrniti le postaje, ki ustrezajo zgornjim omejitvam, moramo definirati dve dodatni poizvedbi 'tipa' *IskalnoDrevo*, kjer filtriramo po ustreznih tipih in variacijah. Tako spisana poizvedba na testnih podatkih traja v povprečju 9 sekund. Časovna in prostorska zahtevnost se tako povečuje neobvladljivo.

2.4.2 *Grafni pristop*

Napram relacijskemu pristopu, bomo na to vprašanje odgovorili s spodnjo poizvedbo:

```
1 MATCH (i:Izdelek)-[:Vsebuje*]→(p:PSU{variacija: 1})
2 MATCH (i)-[:Vsebuje*]→(m:MB{variacija: 2})
3 MATCH (i)-[:Vsebuje*]→(r:RxTx{variacija: 3})
4 return count(i)
```

Slika 3: Grafna poizvedba

Pozornost namenimo eni izmed prvih vrstic v tej poizvedbi. Vsaka izmed teh vrstic je v celoti ekvivalentna kodi v sliki 2. Ta poizvedba v povprečju traja okoli 90ms. Z večanjem števila podatkov, se časovna in prostorska zahtevnost bistveno ne povečuje oziroma je obvladljiva.

3 Hibridna podatkovna baza

Do sedaj smo pokazali, da obstajajo primeri, kjer je grafna podatkovna baza precej bolj fleksibilna in hitrejša kot obravnavana relacijska podatkovna baza. Posledično lahko z gotovostjo trdimo tudi, da je taka grafna podatkovna baza lažje obvladljiva kot tradicionalna relacijska baza.

Nekaj razlogov za ohranjanje relacijske podatkovne baze:

1. Večji nabor strokovnjakov s tehničnimi veščinami za razvoj in vzdrževanje relacijskih baz,
2. Preprostejša shema pri klasičnih ne-rekurzivnih entitetah.
3. Preverjeno delovanje in podpora s strani razvijalcev.
4. Močno standardizirano okolje.
5. Dolga tradicija in uveljavljenost relacijskih podatkovnih baz.
6. Večji izbor vzdrževalnih in razvojnih orodji.

Upoštevajoč zgornje razloge, relacijske baze ne moremo kar tako odpisati in jo zamenjati z grafno, rešitev poiščemo v izdelavi hibridne podatkovne baze. Za naše potrebe torej modeliramo relacijsko podatkovno bazo za podatkovne strukture, nad katerimi želimo poizvedovati z uporabo klasičnega SQL. Podatkovne strukture, ki so v relacijskih bazah težje predstavljene in/ali zahtevajo potratne poizvedbe tako implementiramo v grafni podatkovni bazi.

S tem deloma podatkovno bazo modulariziramo, saj jasno ločimo funkcionalne dele med seboj. S prvimi težavami se srečujemo šele pri operacijah, ki zahtevajo združevanje podatkov, ki so prisotni tako v relacijski kot tudi v grafni podatkovni bazi. Generalnega vmesnika, ki bi omogočal prevod poljubnega SQL ukaza v sintakso GQL oz. Cypher še ni, zato se lahko kvečjemu odločimo za implementacijo aplikacijskega vmesnika, ki implementira točno določeno funkcionalnost. Implementacija posameznih operacij je tako razbita v aplikacijski logiki, ki praviloma upošteva specifike obeh SUPB in tako omogoča implementacijo optimalnih poizvedb in operacij.

3.1 Aplikacijski vmesnik

Aplikacijski vmesnik vsebuje metode, ki se nanašajo na točno določeno operacijo in le to tudi implementirajo. Ker trenutno ni mogoča avtomatska prevedba med SQL in GQL ukazi, generalne implementacije tega vmesnika ne moremo izvesti.

Uporabnik proži aplikacijske ukaze na strani vmesnika. Naloga vmesnika je implementacija in pravilno razpošiljanje ukazov in operacij med grafnim in relacijskim SUPB. Način razpošiljanja, vrstni red in število ukazov, ki se morajo za določeno operacijo izvesti je popolnoma v domeni vmesnika. V grobem lahko trdimo,

da implementacija vmesnika služi podobni nalogi, kot na primer implementacija kompleksnih poizvedb in operacij v PL/SQL sintaksi za Oracle podatkovne baze. Poenostavljeno, gre le za prenos logike iz SUPB specifične implementacije v kodno implementacijo, ki skrbi za pravilen (in optimalen) vrstni red klicev na ciljne SUPB. Posebno pozornost je potrebno posvetiti pri razbijanju podatkovnih modelov in deljenju podatkov med obema SUPB, saj s tem lahko kršimo lastnosti ACID.

3.2 Vmesnik na nivoju SUPB

V okolju MSSQL (od verzije SQL Server 2017 (14.x) in kasnejše) obstaja ukaz "CREATE TABLE ... AS [NODE — EDGE] ... "[12] katerega rezultat so tabele tipa "NODE" (vozlišče) oziroma "EDGE" (rob), vendar so te tabele primarno relacijske. Tabelam teh tipov eksplicitno definiramo atribute, po katerih je moč povpraševati kar z uporabo SQL in MSSQL specifičnih ukazov za delo z grafnimi tabelami. Poizvedovanje in delo nad temi strukturami je podobno kot pri jeziku Cypher. Bistvena razlika se pojavi pri hrambi teh podatkov (in posledično pri časovno-prostorski zahtevnosti), kjer prevlada Neo4j saj podatke hrani na disku v povezanih seznamih ("linked list")[13], vendar so to že specifike samega SUPB. V primerjavi s pristopom z uporabo aplikacijskega vmesnika izgubimo nekaj fleksibilnosti, vendar pa s tem zopet pridobimo vse lastnosti ACID brez, da bi za to morali poskrbeti sami.

4 Zaključek

S standardizacijo jezika GQL za grafne podatkovne baze, ki služi kot protiutež že uveljavljenemu standardu SQL, se je zgodil pomemben korak pri standardizaciji in uveljavitvi grafnih podatkovnih baz. Na področju, ki do pred kratkim še ni imelo standardiziranega poizvedovalnega jezika, se je z uvedbo GQL standarda poenostavil razvoj prihodnjih grafnih SUPB, ki bodo reševali različne potrebe končnih uporabnikov oziroma bodo implementirani kot razširitev v klasičnih relacijskih SUPB po vzoru MS-SQL.

Grafne podatkovne baze predstavljajo pomembno razširitev že obstoječega nabora funkcionalnosti prisotnega v relacijskih podatkovnih bazah. Relacijske in grafne podatkovne baze so ekvivalentne, saj vse kar je predstavljivo v eni, je možno predstaviti v drugi. Pomembne razlike so le v delovanju in v objektivno boljšem obravnavanju določenih tipov podatkovnih struktur in podatkov v posameznih SUPB.

Pokazali smo, da predstavljeno Cypher poizvedbo lahko implementiramo tudi v SQL-u. Poizvedbi sta funkcionalno stvari reševali na identičen način, vendar se je časovna zahtevnost med obema sistema bistveno razlikovala. Prikazan način implementacije poizvedbe je le eden od mnogih, saj lahko poizvedbo rešujemo tudi proceduralno. V tem primeru, mora bazni programer bistveno bolje poznati delovanje podatkovne baze. Za optimalno rešitev naloge mora poznati različne preiskovalne algoritme in podatkovne strukture za delo z grafi.

Razvojniki se lahko poslužujejo implementacije hibridnih sistemov, katerega namen je čim bolj učinkovita izraba latentnih prednosti v sodelujočih SUPB. Implementacije teh hibridnih sistemov so zelo prilagojene reševanemu problemu in praviloma niso generalne zaradi močno optimiziranega algoritma s pomočjo katerega koordiniramo delovanje med SUPB-ji.

Zahvala: To delo je podprto z raziskovalnim programom ARIS Računalniški vid (P2-0214).

Literatura

[1] E. F. Codd. "A relational model of data for large shared data banks". V: *Commun. ACM* 13.6 (1970). ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: https://doi.org/10.1145/362384.362685.

[2] *Primerjava različnih SUPB*. URL: https://troels.arvin.dk/db/rdbms/ (pridobljeno 23. 7. 2024).

[3] *SQL Standard*. URL: https://www.iso.org/standard/76583.html (pridobljeno 23. 7. 2024).

[4] *Organizacija ISO/IEC*. URL: https://www.iso.org/ (pridobljeno 23. 7. 2024).

[5] Musa Garba in Hassan Abubakar. "A comparison of nosql and relational database management systems (rdbms)". V: *Kasu Journal Of Mathematical Sciences* 1.2 (2020), str. 61–69.

[6] Diogo Fernandes, Jorge Bernardino in sod. "Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB." V: *Data* 10 (2018), str. 0006910203730380.

[7] *Standard GQL*. URL: https://www.iso.org/standard/76120.html (pridobljeno 23. 7. 2024).

[8] Blessing E James in Prince Oghenekaro Asagba. "Hybrid database system for big data storage and management". V: *International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol* 7 (2017).

[9] *Neo4J komentar ob uvedbi GQL standarda*. URL: https://neo4j.com/press-releases/gql-standard/ (pridobljeno 24. 7. 2024).

[10] *Neo4J upoštevanje lastnosti ACID*. URL: https://neo4j.com/docs/cypher-manual/current/introduction/cypher-neo4j/#_acid_compliance (pridobljeno 31. 8. 2024).

[11] *Izvorna koda za vzpostavitev testnega okolja*. URL: https://nc2g.eu/erk2024 (pridobljeno 5. 9. 2024).

[12] *MS SQL grafne tabele*. URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-sql-graph?view=sql-server-ver16 (pridobljeno 31. 8. 2024).

[13] *Neo4J način shranjevanja podaktov*. URL: https://neo4j.com/docs/operations-manual/current/database-internals/store-formats/ (pridobljeno 31. 8. 2024).