Univerzitet u Beogradu
Matematički fakultet

Baza podataka HyperGraphDB

Seminarski rad

ime i prezime	**Nikola Stanojević**
broj indeksa	1064/2012
predmet	Distribuirane i objektne baze podataka
školska godina	2013/2014
nastavnik	dr Saša Malkov
datum	01.12.2013

Sadržaj

1 Uvod

HyperGraphDB je softver za skladištenje podataka, opšte namene i otvorenog koda, zasnovan na moćnim formalizmima upravljanja znanjem poznatim kao usmereni hipergrafovi. Osim kao model stalne memorije dizajniran prvenstveno za upravljanje znanjem, veštačku inteligenciju i web projekte, ove baze mogu se koristiti i kao ugrađene (*en. embedded)* objektno orijentisane baze podataka za Java projekte različitog obima. Takođe mogu se koristiti i kao grafovske baze podataka ili kao ne-SQL relacione baze podataka.

HyperGraphDB je primarno, kao što joj i ime govori, baza podataka namenjena za smeštanje hipergrafova. Iako generalno pripada familiji grafovskih baza podataka, teško je kategorizovati HyperGraphDB bazu podataka kao još jednu standardnu bazu iz te familije. Prvenstveno zbog naprednog dizajna ove baze podataka koji omogućava upravljanje bogato struktuiranim informacijama sa visokom kompleksnošću.

Na primer, relacioni i objektno orijentisani stilovi upravljanja podacima se mogu oponašati. Kao grafovska baza HyperGraphDB ne nameće ograničenja i nudi mnogo veću uopštenost u odnosu na sve ostale grafovske baze podataka. Dizajn ove baze je prvenstveno minimalistički i njen cilj jeste da se razvije niz koncepata, kombinujući strukturu i interpretaciju na način koji omogućava softveru koji će nastati u budućnosti da dostigne kompleksnost stvarnog sveta bolje nego danas.

Ove grafovske baze našle su primenu prvenstveno u veštačkoj inteligenciji, bioinformatici i procesiranju prirodnog jezika zbog ogromne količine podataka koja se u ovim oblastima koristi.

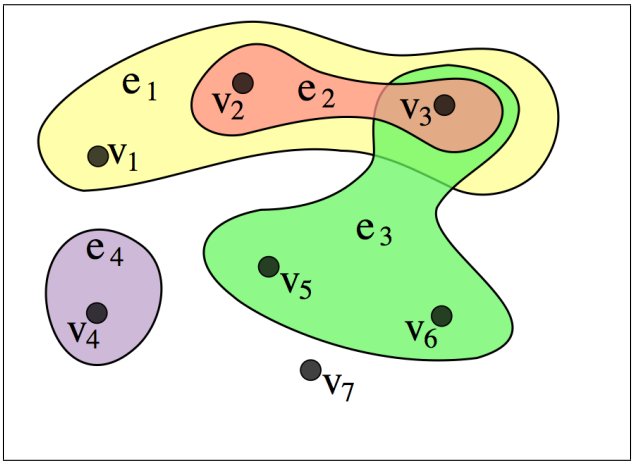
2 Hipergrafovi i model podataka baze HyperGraphDB

Standardna matematička definicija hipergrafa jeste da on predstavlja generalizaciju grafa, gde grane grafa mogu da povezuju bilo koji broj čvorova. Formalno, hipergraf je par (X,E) gde je X skup elemenata, čvorova, a E je skup podskupova od X, koji se nazivaju hipergrane. Grane grafa su predstavljene parovima čvorova koje one spajaju, dok su hipergrane proizvoljni skupovi čvorova, koji stoga mogu da sadrže proizvoljan broj čvorova.

Hipergraf se takođe naziva sistemom skupa ili familijom skupova izvučenih iz univerzalnog skupa X. Hipergrafovi se mogu posmatrati kao incidentne strukture, i obratno. Mnoge teoreme koje se tiču grafova su primenjive i na hipergrafove. Remzijeva teorema je tipičan primer. Neke metode za proučavanje simetrija grafova se mogu proširiti na hipergrafove. Na primer, homomorfizam hipergrafova je preslikavanje iz skupa čvorova jednog hipergrafa u drugi, takvo da se svaka grana preslikava u drugu granu. Izomor-

fizam hipergrafova je inverzibilni homomorfizam. Automorfizam hipergrafova je izomorfizam jednog skupa čvorova u samog sebe, to jest, preimenovanje čvorova.

Za razliku od grafova, hipergrafove je teško nacrtati na papiru, tako da se oni obično izučavaju pomoću nomenklature teorije skupova pre nego korišćenjem vizuelnih prikaza iz teorije grafova.



Slika 1: Neusmereni hipergraf

U modelu podataka HyperGraphDB baze podataka , osnovna jedinica predstavljanja podataka je atom. Svaki atom se sastoji od pridružene torke atoma koja se naziva ciljnim skupom. Veličina ciljnog skupa definisana je kao arnost atoma. Atomi koji imaju vrednost arnosti 0 se nazivaju čvorovi (atomi koji ne pokazuju ni na sta), a atomi koji imaju vrednost arnosti veću od 0 se nazivaju grane (atomi koji pokazuju na nešto). Skup učestalosti atoma x je skup atoma koji imaju atom x kao člana njihovog ciljnog skupa (npr. broj grana koje sadrže x).

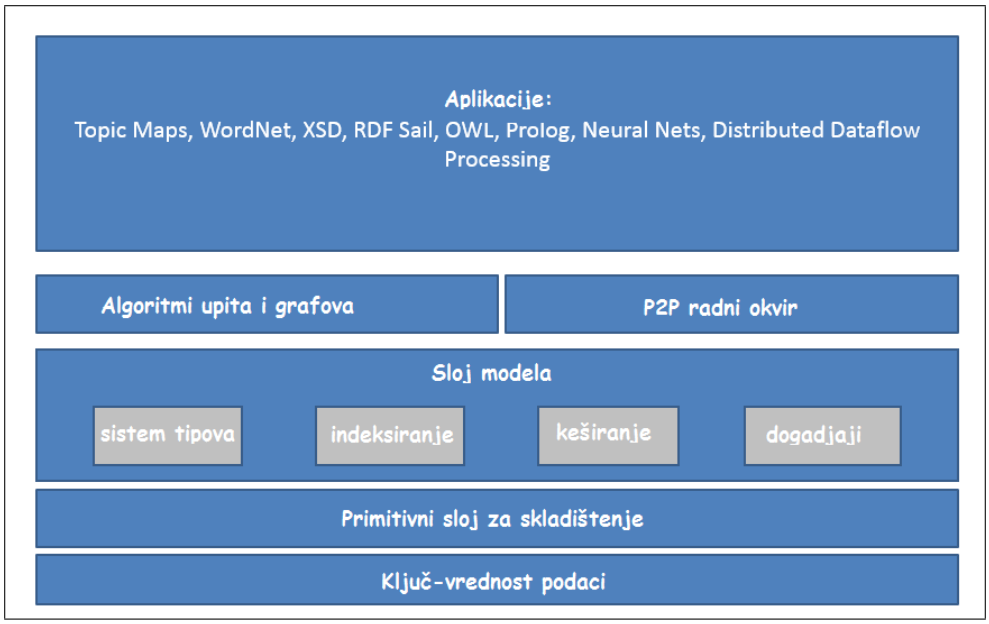
Takođe, svaki atom ima strogo tipiziranu pridruženu vrednost. Vrednosti su arbitražnog tipa, takođe i sami tipovi predstavljaju atome. Zajedno, atomi i vrednosti predstavljaju dva ortogonalna aspekta informacija koje su smeštene u HyperGraphDB bazu podataka. Atomi predstavljaju semantičke entitete koji formiraju strukturu hipergrafova, dok vrednosti predstavljaju tipizirane podatke koji mogu biti struktuirani. Vrednost atoma može biti promenjena ili zamenjena vrednošću različitog tipa, a da se pri tom zadrži identična struktura. Takođe moguće je da više atoma istovremeno deli istu vrednost. S druge strane, vrednosti su nepromenljivi entiteti.

Važno je primetiti da ovaj model ne nameće ni jedan određeni vid semantike načina smeštanja podataka. Ovaj model je polustruktuirani model opšte namene koji uključuje grafovske, relacione i objektno orijentisane aspekte. Shema HyperGraphDB je tipovni sistem koji može da se usavršava dinamički, gde se ograničenja integriteta podataka

mogu izazvati samom implementacijom tipova.

3 Dvoslojna arhitektura

Jedan od osnovnih aspekata arhitekture HyperGraphDB baze podataka jeste organizacija podataka u dva sloja. Dvoslojna arhitektura HyperGraphDB baze podataka definiše primitivnu apstrakciju hipergrafovskog skladišta, sastoji se od primitivnog sloja za skladištenje i sloja modela.



Slika 2: Arhitektura HyperGraphDB baza

Primitivni sloj za skladištenje je podeljen u dva asocijativna niza:

Niz veza: ID $\rightarrow$ Lista $<$ ID $>$
Niz podataka: ID $\rightarrow$ Lista $<$ bajt $>$

Drugim rečima, primitivni sloj se sastoji od grafa ID-eva i sirovih podataka. Podrazumevana reprezentacija identifikatora je kriptografski jaka verzija 4 univerzalno jedinstvenih identifikatora (*en. universally unique identifier (UUID)*). Verzija 4 koristi šemu koja se oslanja samo na slučajne brojeve.

Algoritam postavlja broj verzije zajedno sa dva rezervna bita. Svi ostali bitovi (ostalih 122 bita) postavljaju se korišćenjem slučajnog ili pseudoslučajnog generatora brojeva. Verzija 4 ima oblik:

xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx

Broj 4 označava verziju UUID, svako od x je neka heksadekadna cifra, a svako od y je jedna od cifara: 8, 9, A, ili B (npr. f47ac10b-58cc-4372-a567-0e02b2c3d479).

Ovakva reprezentacija olakšava upravljanje identifikatorima u velikim distribuiranim okruženjima, gde se praktično eliminiše šansa za koliziju. Svaki čvor (*en. peer)* može napraviti nove ID-eve nezavisno. Osnovni zahtev za ključ-vrednost perspektivu skladištenja je da indikatori podržavaju više uređenih vrednosti po pojedinčnom ključu.

U sloju modela se čuva apstrakcija atoma hipergrafa, zajedno sa sistemom tipova, keširanjem, indeksiranjem i upitima. Sloj modela je implementiran formalizovanjem primitivnog sloja za skladištenje na sledeći način:

$$\begin{aligned}
&\text{AtomID} \rightarrow [\text{TypeID,ValueID,TargetID,...,TargetID}] \\
&\text{TypeID} \rightarrow \text{AtomID} \\
&\text{TargetID} \rightarrow \text{AtomID} \\
&\text{ValueID} \rightarrow \text{List} < \text{ID} > \mid \text{List} < \text{byte} >
\end{aligned}$$

Drugim rečima svaki ID predstavlja ili atom hipergrafa ili vrednost atoma. U prethodnom, atom se čuva kao torka čiji prvi argument predstavlja tip atoma, drugi predstavlja vrednost, a ostatak argumenata predstavljaju ciljni skup atoma. Takođe vrednost može da se čuva direktno kao niz bajtova ili može biti sačuvan preko niza identifikatora. Raspored identifikatora atoma određen je osnovnim radnim okvirom (*en. framework)* dok je određivanje rasporeda vrednosti podataka dodeljeno implementaciji tipova.

Primećuje se da vrednosti ValueID formiraju rekurzivnu strukturu omogućavajući smeštanje kompleksnih struktura koje su ortogonalne u odnosu na apstrakciju atoma grafa. Zbog toga je potrebna efikasnija implementacija sloja modela. Osnovni indeksi koji su potrebni radi efikasnije implementacije ovog model su:

$$\begin{aligned}
&\text{IncidenceIndex} : \text{UUID} \rightarrow \text{SortedSet} < \text{UUID} > \\
&\text{TypeIndex} : \text{UUID} \rightarrow \text{SortedSet} < \text{UUID} > \\
&\text{ValueIndex} : \text{UUID} \rightarrow \text{SortedSet} < \text{UUID} >
\end{aligned}$$

IncidenceIndex mapira atom hipergrafa sa nizom svih grana koje pokazuju na taj atom. Takođe IncidenceIndex mapira atom hipergrafa sa skupom svih njegovih atoma instanci. ValueIndex mapira *top-level* vrednost strukture sa skupom atoma koji imaju tu vrednost.

4 Tipiziranost

HyperGraphDB baze podataka imaju mogućnost skladištenja proizvoljnih tipova podataka kao atoma na neutralan način u odnosu na programske jezike. Specijalno kao ugrađena (*en. embedded)* baza podataka mapira vrednosti podataka na jedinstvenom jeziku i formira stalno skladište. Stoga potrebno je da pokrije tipovne konstrukcije koje se mogu naći u različitim programskim jezicima. Ovo se postiže preko globalnog, proširivog mehanizma tipova koji je u veoma bliskoj interakciji sa nivoom skladištenja. U teoriji tipova u računarstvu, tipovi su formalizovani tako što je rečeno šta se sa njima može raditi.

Osnovni pojam je jednakost: kada su dva elementa datog tipa jednaka? Drugi osnovni koncept jeste kreiranje novih tipova od postojećih tipova koji su slični. Da bi se napravio novi tip, jednom od sličnih tipova potreban je konstruktor tipova. Konačno, pojam podtipa, koji je srodan obuhvatanju ili zamenljivosti, obično se uvodi nezavisno. S druge strane, jednakost se može shvatiti kao dvosmerno obuhvatanje: dve instance iste vrste su jednake ako i samo ako se mogu obuhvatiti međusobno.

Na osnovu svega prethodnog dobijamo sledeći minimalistički dizajn:
-Tip je atom sposoban da skladišti, kreira, i uklanja runtime reprezentacije svojih instanci i formira primitivni sloj za skladištenje.
-Tip je sposoban, kada mu se daju neke njegove dve instance, da kaže da li se jedna od njih može zameniti drugom. (relacija zamene).

Drugim rečima, tipovi su atomi kojima je dodeljena specijalna uloga koja dozvoljava sistemu da održava veze između tipova i instanci. Ovo je formalizovano korišćenjem sledećeg Java interfejsa koji mora implementirati svaki atom tip:

```
public interface HGAtomType {
    Object make(ID valueId,
                List<ID> targetSet,
                Set<ID> incidenceSet);
    ID store(Object instance);
    void release(ID valueId);
    boolean subsumes(Object general, Object specific);
}
```

Pošto su vrednosti po definiciji nepromenljive, ne postoji metod ažuriranja. Važno je primetiti da metod kreiranja (*en. make method)* uzima i ciljni skup i skup učestalosti kao dodatne parametre (pored identifikatora vrednosti u primitivnom skladištu). Zaista, reprezentacija atoma i jeste funkcija koja obuhvata oba ova niza. To znači da atom može biti označen različitim odnosima koji ga definišu prilikom izvršavanja.

Metod kreiranja daje mogućnost tipu da proizvodi objekte. Konstruktor tipa je jednostavno atom koji vraća jednu HGAtomType instancu korišćenjem njegovog metoda kreiranja. Sistem tipova u sebi sadrži niz predefinisanih tipova koje je moguće dodatno konfigurisati a koji pokrivaju standardne jednostavne vrednosti kao što su brojevi i niske. Takođe sistem tipova sadrži neke standardne konstruktore tipova za strukture zapisa (*en. record)* , liste i mape. Takvi predefinisani tipovi se jednim imenom nazivaju top tipovima, što takođe predstavlja i njihov zajednički tip.

Na primer, konstruktor tipa za strukture zapisa upravlja tipovima zapisa, koji dalje upravljaju konkretnim zapisima. Jedinstveni tip kao što je zapis ili lista može da skladišti svoje komponente oslanjajući se rekurzivno na ostale HyperGraphDB tipove. To znači da HGAtomType implementacija može da rukuje kako vrednostima na nivou atoma tako i vrednostima na čitom (kompozitnom) nivou vrednosti.

Strukture u stilu zapisa sa imenovanim delovima su tako česte da je u ovom slučaju definisan i apstraktni interfejs za njih koji se naziva HGCompositeType koji kompleksne vrednosti vidi kao višedimenzione strukture gde se svaka dimenzija može identifikovati svojim imenom. Takođe ovaj interfejs ima i pridruženu HGProjection implementaciju koja je u mogućnosti da manipuliše vrednostima duž tih dimenzija. Takvim tipovima koji upravljaju složenim strukturama je dozvoljeno i da ih deli u odvojene identitete u skladištu i obezbeđuju funkcije kao što su indeksiranje, postavljanje upita, ili brojanje referenci za te delove. Dodatno, oni mogu da ih skladište kao kompaktne delove (*en. blobs)* kojima se može pristupati samo sa nivoa atoma.

Konačno, tipovi u HyperGraphDB objedinjuju mnoge aspekte koji se često mogu pronaći u sistemima tipova različitih jezika u računarstvu. Oni imaju ulogu da proizvode objekte kao što to rade klase u objektno orijentisanim jezicima. Oni takođe definišu strukturna svojstva atoma. Na kraju, oni imaju i semantičku ulogu prilikom ograničavanja korišćenja atoma i primene određenih svojstava konzistentnosti i integriteta koje su tipične kako u sistemima tipova tako i u sistemima baza podataka.

Kombinacija ovih aspekata omogucava HyperGraphDB bazama podataka da prirodno integrišu različite oblike formalizama.

5 Indeksiranje

Nastavljajući u duhu refleksivne, otvorene arhitekture HyperGraphDB baze podataka, indeksiranje je takođe proširivo. Implicitno indeksiranje atoma nije opciono već je od suštinskog značaja za efikasnu podršku semantici sloja modela. Na primer, indeksiranje skupa učestalosti (*en. incidence set)* je od velikog značaja za performanse prilikom postavljanja upita nad grafovskim strukturama. Korišćenjem indeksa važni

dodatni indikatori se čuvaju kako na primitivnom tako i na sloju modela.

Na primitivnom sloju postoje indikatori koji se stvaraju prilikom implementiranja tipova. Ovakvi indikatori se kreiraju direktno od sloja za skladištenje podataka, a njima se upravlja interno preko tipa. Na primer, osnovna implementacija primitivnih tipova održava indeks između primitivnih podataka i identifikatora njihovih vrednosti, što izaziva deljenje vrednosti između atoma na nivou skladištenja. Ovo deljenje vrednosti se ne vrši kao podrazumevano za kompleksne tipove jer kod njih ne postoji univerzalni način da se tako nešto uradi efikasno ukoliko se ne uvede pojam primarnog ključa implementiranjem odgovarajućeg konstruktora složenih tipova.

Na sloju modela, indikatori se mogu koristiti kako sa grafovskom strukturom tako i sa vrednosnom strukturom atoma. Međutim oni su uvek povezani sa tipovima atoma. Ova povezanost je manje ograničavajuća nego što na prvi pogled to izgleda, jer se indeks primenjuje na sve podtipove tipa koji je sa njima u vezi. Indikatori se registruju sa sistemom implementirajući HGIndexer interfejs. Instance HGIndexer-a se skladište i kao atomi, koji mogu imati stalne meta-podatke za planiranje upita, algoritme aplikacija itd. U sledećoj tabeli nalaze se neki od predefinisanih indeksa:

ByPartIndexer	Indeksira atome složenim tipovima zajedno sa datim dimenzijama
ByTargetIndexer	Indeksira atome prema specifičnom cilju (npr. prema poziciji u ciljnoj torci)
CompositeIndexer	Komponuje neka dva indeksa u jedan zajednički
LinkIndexer	Indeksira atome prema celoj njihovoj ciljnoj torci.
TargetToTargetIndexer	Za datu granu, indeksira jedan od njenih ciljeva drugim.

Tabela 1: Predefinisani indeksi u HyperGraphDB

6 Postavljanje upita

Postavljanje upita nad grafovskim bazama može imati jednu od nekoliko različitih formi. Jedna od njih jeste forma u kojoj se zadaje sekvenca atoma (forma obilaska). Takođe zadavanjem skupa ne nužno povezanih atoma koji su slični po nekom predikatu , kao u relacionim bazama, je još jedna od formi zadavanja upita. Konačno, treca kategorija je poklapanje šablona (*en. pattern)* grafovskih struktura.

Forma obilaska je definisana u vidu API iteratora. Očigledno, što je opštiji model hipergrafa to podrazumeva veću odgovarajuću generalizaciju obilaska. Postoji nekoliko

aspekata te generalizacije. Prvo, generalizovan je pojam susedstva čvorova. Susedni atomi se pronalaze kao kod standardnih grafova, ispitivanjem veza usmerenih ka datom atomu, npr. njegov skup učestalosti.

Za dati atom xi i skup tipiziranih torki grana koje na njega pokazuju, često je potrebno ispitati samo podskup tih grana. Pa ako je grana koja nas zanima $l = [x_1, ..., x_i, ..., x_n]$, susedni atomi mogu biti u podskupu od $\{x_{i+1}...x_n\}$ ili $\{x_{i-1}...x_1\}$ zavisno od tipa atoma i svojstava kao i od smera pretrage.

Dalje, u slučaju uniformnih strukturalnih oblika u grafu, jedan od njih može biti zainteresovan za pojam srodnosti, gde je uključeno više od jedne grane - znači y je susedan x-u ako su povezani na bilo koji način.

Svi ovi slučajevi se rešavaju na nivou API-a obezbeđivanjem algoritma pretrage (*en. traversal algorithm)* . Može se korititi bilo pretraga u širinu (*en. breadth-first search)* bilo pretraga u dubinu (*en. depth-first search)* . Takođe je obezbeđen i generator liste suseda - interfejs koji vraća listu atoma koji su susedni datom atomu.

Drugi deo generalizacije pretragom sastoji se od dodavanja dodatne dimenzije procesu, dozvoljavajući nekom od njih da prati incidentne grane zajedno sa susednim atomima.

Izrazi upita se sastoje od niza primitiva koje su prikazane u sledećoj tabeli:

eq(x), lt(x), eq("name", x), ...	Upoređuje vrednosti atoma
type(TypeID)	Tip atoma je TypeID
typePlus(TypeID)	Tip atoma je podtip od TypeID
subsumes(AtomID)	Atom je još opštiji od AtomID.
target(LinkID)	Atom pripada ciljnom skupu od LinkID.
incident(TargetID)	Atom pokazuje na TargetID.
link($ID_1, ..., ID_n$)	Ciljni skup atoma sadrži $\{ID_1, ..., ID_n\}$.
orderedLink($ID_1, ..., ID_n$)	Atom je torka u zadatoj formi.
arity(n)	Arnost atoma je n.

Tabela 2: Primitive koje se koriste u upitima

Takođe u upitima se koriste i standardni *AND, OR* i *NOT* operatori.

Većina aktivnosti u upitima odnosi se na izvršavanje operacija spajanja različitih indikatora koji su dostupni u upitu. Manji deo aktivnosti je keširanje čestih skupova atoma u uređene nizove ID-eva, što značajno olakšava njihovo pretraživanje. Čitave celine ili delovi većih čestih skupova se keširaju i na nivou skladištenja, ali oni ostaju

u formi B-stabala. Spajanja su najčešće ostvarena na par sortiranih skupova (najčešće na skupove UUID-eva) pomoću cik-cak algoritma (*en. zig-zag algorithm)* , gde skupovi podržavaju nasumični pristup na osnovu ključa, ili algoritam spajanja (*en. merge algorithm)* .

Ukrštanja su implementirana kao bidirekcioni iteratori, koji dozvoljavaju bektreking tokom izvršavanja sofisticiranih algoritama pretrage.

7 P2P distribuiranost

Distribuiranost podataka u HyperGraphDB bazama podataka je implementirana u sloju modela koristeći P2P (*en. Peer to peer)* radni okvir. Algoritmi za to su razvijeni koristeći komunikacione protokole ugrađene korišćenjem ACL (*en. Agent Communication Language)* FIPA standarda. ACL je baziran na teoriji govornog čina i definiše primitivne komunikacione elemente kao što su predlaganje, prihvatanje, informisanje, zahtevanje, upit itd.

Svaki agent održava skup razgovora implementirajući tokove razgovora (*en. workflow)* (najčešće preko jednostavnih mašina stanja). Sve aktivnosti su asinhrone pa su dolazne poruke prosleđene pomoću planera (*en. sheduler)* i procesuirane u izvršiocu niti (*en. thread pool).*

P2P arhitektura je često korišćena kod HyperGraphDB baza podataka kao mehanizam za distribuiranu reprezentaciju znanja. Pretpostavlja se da neće biti moguće postići stalnu dostupnost cele baze znanja na svakoj lokaciji, ili se to čak u nekim slučajevima smatra i nepoželjnim. Na primer, ako imamo replikacionu shemu sa konstantnim podacima baziranu na narednom algoritmu:

1. Na početku svaki agent javlja da je zainteresovan za određeni atom (što je definisano predikatom tipa boolean) svim svojim čvorovima (*en. peers).* .

2. Zatim svaki od čvorova osluškuje atom događaje od baze podataka i ako dođe do dodavanja, menjanja ili uklanjanja nekog od atoma, zainteresovani čvorovi bivaju odmah obavešteni o nastalim promenama tako što im se pošalje nazad informaciona poruka.

3. Da bi se obezbedila konzistentnost, lokalne transakcije su linearno uređene po broju verzije i povezane kako bi mogle da kontaktiraju sve zainteresovane čvorove.

4. Čvor koji primi obaveštenje o nekom od njemu potrebnih atoma mora da potvrdi prijem takve informacije i da odluči da li da pokrene transakciju lokalno ili ne. Ovde

ne postoji sistem dvofaznog potvrđivanja (*en. two-phase commit)* : jednom kada se događaj potvrdi od strane onoga kome je informacija upućena , onaj koji informaciju šalje može smatrati da je ona i procesuirana.

Ovaj pristup balansira između pristupa replikacije i particionisanja podataka na korisničkom nivou. Potpuno transparentna alternativa koja pokušava da prilagodi tačnu lokaciju atoma zasnovana na iskorišćenosti i meta podacima imala bi za rezultat nepredvidive i česte udaljene upite, uz potrebu za značajnom količinom dodatnog lokalnog prostora za skladištenje podataka.

8 Primer. Tipovi, podaci, upiti

Sledećim primerom pokazujemo kako HyperGraphDB može biti kreirana i kako se objekti tipa Knjiga dodaju u bazu podataka.

Sledeći kod predstavlja klasu kojom je definisan jedan objekat tipa knjiga:

```
public class Knjiga {

    private String naziv;
    private String autor;

    public Knjiga() {
    }

    public Knjiga(String naziv, String autor) {
        this.autor = autor;
        this.naziv = naziv;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public String getNaziv() {
        return naziv;
    }

    public void setNaziv(String naziv) {
        this.naziv = naziv;
    }

}
```

```
// Naredni kod predstavlja glavni program:

import java.util.List;
import org.hypergraphdb.HGHandle;
import org.hypergraphdb.HGQuery.hg;
import org.hypergraphdb.HyperGraph;


public class HGDBSimpleSample {
    public static void main(String[] args) {
        String databaseLocation = "/putanja/do/
            hypergraphdb";
        HyperGraph graph = null;
        try {
            graph = new HyperGraph(databaseLocation);
            Knjiga mojaKnjiga = new Knjiga("Romeo and
                Juliet", "William Shakespeare");
            HGHandle knjigaHandle = graph.add(
                mojaKnjiga);

            List<Knjiga> knjige = hg.getAll(graph, hg.
                and(hg.type(Knjiga.class), hg.eq("
                autor", "William Shakespeare")));
            for (Knjiga knjiga : knjige)
                System.out.println(knjiga.getNaziv());
        } catch (Throwable t) {
            t.printStackTrace();
        } finally {
            graph.close();
        }
    }
}
```

Razmotrimo sledeći kod i postavljanje upita u HyperGraphDB:

```
public class Ime {
    private String prezime;
    public String getPrezime() { return prezime; }
    public void setPrezime(String prezime) { this.
        prezime = prezime; }

    private String ime;
    public String getIme() { return ime; }
    public void setIme(String ime) { this.ime = ime;
        }

    public Ime(){};
}

// i ako dodamo neka imena:

            Ime a = new Ime();
            a.setPrezime("Mark");
```

```
            a.setIme("Stefan");
            graph.add(a);

            Ime b = new Ime();
            b.setPrezime("Hugo");
            b.setIme("Victor");
            graph.add(b);

            Ime c = new Ime();
            c.setPrezime("Chavez");
            c.setIme("Hugo");
            graph.add(c);

            Ime d = new Ime();
            d.setPrezime("Camus");
            d.setIme("Albert");
            graph.add(d);
```

Potrebno je napisati upit kako bi se pronasli svi koji 1) imaju 2) nemaju za prezime ili ime "Hugo".

```
List<Name> hugos = hg.getAll(graph,
                    hg.and(hg.type(Name.class),
                    hg.or(hg.eq("prezime", "Hugo"),
                    hg.eq("ime", "Hugo"))));

List<Name> noHugos = hg.getAll(graph,
                    hg.and(hg.type(Name.class),
                    hg.not(hg.or(hg.eq("prezime", "
                       Hugo"),
                    hg.eq("ime", "Hugo")))));

    for(Name n : hugos)
            System.out.println("Jeste Hugo: prezime:"
               +
            n.getPrezime() + " . ime: " + n.getIme());

    for(Name n : noHugos)
            System.out.println("Nije Hugo: prezime:" +
            n.getPrezime() + " . ime: " + n.getIme());
```

izlaz:
Nije Hugo: prezime: Mark . ime: Stefan
Jeste Hugo: prezime: Hugo . ime: Victor
Jeste Hugo: prezime: Chavez . ime: Hugo
Nije Hugo: prezime: Camus . ime: Albert

9 Zaključak

Model podataka koji se koristi u HyperGraphDB generalizuje klasične grafove u više dimenzija i kao takav nije uvršten u tipove skladišta bazirane na pokazivačkim strukturama. S druge strane, otvorena arhitektura i proširivi sistem tipova omogućava modelu lako obuhvatanje različitih formalizama bez obaveznog gubitka na perfomansama prilikom tog procesa. Dalji razvoj upitnog jezika za HyperGraphDB model, sistemi povezivanja za ostale programske jezike kao što je C++, podrška za ugnježdene grafove i algoritimi distribuiranja bazirani na uspostavljenim osnovama će u bliskoj budućnosti podstaći prihvatanje ovog modela podataka i u oblastima u kojima do danas nije korišćen.

10 Reference

1. Zvanični web sajt HyperGraphDB baze podataka

2. Directed hypergraphs and applications, Giorgio Gallo, Giustino Longo, Sang Nguyen, Stefano Pallottino

3. HyperGraphDB: model and applications, Borislav Iordanov

4. Dokumentacija za HyperGraphDB baze podataka

5. Graph databases, Ian Robinson, Jim Webber, Emil Eifrem

6. A Universally Unique IDentifier (UUID) URN Namespace, P. Leach, M. Mealling, R. Salz