UNIVERZITET
METROPOLITAN
BEOGRAD

IT250 - BAZE PODATAKA

Dokument baze podataka

Lekcija 13

PRIRUČNIK ZA STUDENTE

IT250 - BAZE PODATAKA

Lekcija 13

DOKUMENT BAZE PODATAKA

UVOD U LEKCIJU

Šta ćemo naučiti u ovoj lekciji?

Dokument baze podataka radi sa hijerarhijski struktuiranim dokumentima. Dokumenta najčešće sačinjavaju mape i liste, koje omogućavaju uspostavljanje prirodne hijerarhije – najčešće korišćenjem formata kao što su XML i JSON.

Proučavanje baza podataka sa aspekta XML standarda je aktuelna tema, jer se na takvom konceptu zasniva veliki broj e-business aplikacija. U XML okruženju, u bazu podataka se mogu smeštati podaci koji su struktuirani kao XML dokumenti ili iz nje ekstrahovati podaci u XML formatu. Samo čuvanje XML dokumenata u bazi podataka se može realizovati kroz XML-enabled baze podataka i Native XML baze podataka.
XML baze podataka su preteče danas mnogo češće korišćenih NoSQL baza podataka. NoSQL način razmišljanja je tehnološko obeležje promene tradicionalnog načina menadžmenta podataka. Popularnost NoSQL baza se krije u brzini koju nude programerima. NoSQL rešenja se lakše grade, ažuriraju i objavljuju od relacionih. Dizajn je okrenut brzim odgovorima i brzom dodavanju podataka.

Brzina se povećava na osnovu manje kvalitetnih rešenja u drugim poljima; neki od nedostataka NoSQL-a se ogledaju u manjoj konzistentnosti i većem vremenu potrebnom da se izvrši ažuriranje podataka. NoSQL rešenja treba primenjivati kada konzistentnost podataka nije od velikog značaja.

Poglavlje 1

XML enable baze podataka

ŠTA SU XML ENABLED BAZE PODATAKA?

U slučaju XML enabled baze podataka, XML se koristi za razmenu podataka između baze i neke aplikacije ili druge baze. U samoj bazi podataka se "ne vidi" XML dokument

U interakciji između XML-a i baza podataka, XML se može koristiti na dva načina:

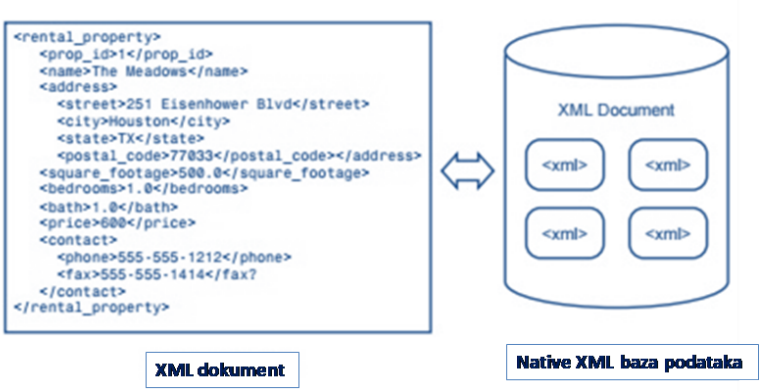
1. **u bazi podataka se može pamtiti sam XML dokument** (native baze podataka)
2. **XML se smešta u relacionu bazu podataka kao skup tabela eksplicitno dizajniranih za pamćenje XML dokumenata** (XML enabled baze podataka)

U XML enabled bazama podataka treba posmatrati dva procesa:

1. i*zdvajanje podataka iz baze i konstruisanje XML dokumenta – što je poznato kao publikovanje ili kompozicija.*
2. *izdvajanje podataka iz XML dokumenata i pamćenje tih podataka u bazi podataka – što je poznato kao isecanje ili dekompozicija.*

Bez obzira da li se XML dokumenti seku ili publikuju, treba naglasiti dve važne stvari:

1. U samoj bazi podataka XML se "ne vidi" – tj. XML dokument je potpuno izvan baze podatke**.** On se konstruiše na osnovu podataka koji se nalaze u bazi podataka ili se koristi kao izvor podataka koji se pamte u bazi.
2. Drugo, šema baze podataka se slaže sa XML šemom, što znači da je za svaku šemu baze podataka potrebna različita XML šema. Baza podataka koja na taj način koristi XML je poznata kao ***XML-enabled baza podataka.***



Slika 1.1 Native XML baza podataka [Izvor: Autor]

MAPIRANJE ŠEME BAZE PODATAKA U XML ŠEMU I OBRNUTO

Postoje tri vrste mapiranja a ovde će biti reči o mapiranju baziranom na tabelama

Kada se koristi XML-enabled baza podataka, neophodno je šemu baze podataka mapirati u XML šemu (ili obrnuto).

Mapiranje šeme baze podataka u XML šemu je moguće na tri načina:

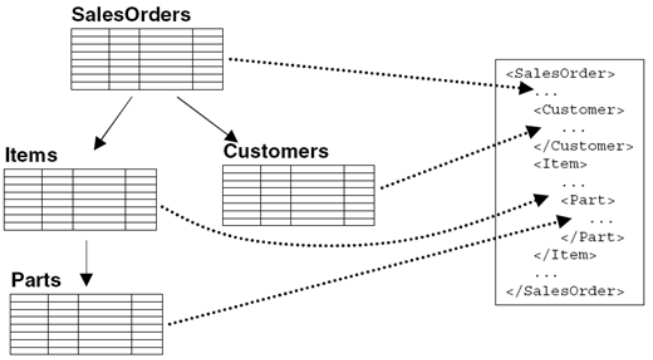
1. bazirano na tabelama (eng.table-based)
2. objektno-relacionim mapiranjem (eng.object-relational)
3. upitnim jezicima

Mapiranje bazirano na tabelama: *individualni redovi tabela se umeću u XML dokument, s tim što relacije primarni ključ/ strani ključ koje postoje u bazi podataka određuju kako će ti redovi biti umetnuti.* Tako stabla tabela u bazi podataka postaju stablo elemenata u XML dokumentu, što je prikazano na slici 1.2.

Primer mapiranja SQL šeme u XML šemu:

```
CREATE TABLE Korisnici (
    ID INT PRIMARY KEY,
    Ime VARCHAR(50),
    Prezime VARCHAR(50),
    Email VARCHAR(100)
);
```

```
<xs:element name="Korisnici">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Korisnik" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ID" type="xs:int"/>
            <xs:element name="Ime" type="xs:string"/>
            <xs:element name="Prezime" type="xs:string"/>
            <xs:element name="Email" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



Slika 1.2 Mapiranje bazirano na tabelama [Izvor: NM IT350 - 2020/2021.]

Primer mapiranja XML šeme u SQL šemu:

```
<Korisnici>
    <Korisnik>
        <ID>1</ID>
        <Ime>John</Ime>
        <Prezime>Doe</Prezime>
        <Email>john.doe@example.com</Email>
    </Korisnik>
    <!-- ... ostali korisnici ... -->
</Korisnici>
```

```
CREATE TABLE Korisnici (
    ID INT PRIMARY KEY,
    Ime VARCHAR(50),
    Prezime VARCHAR(50),
    Email VARCHAR(100)
);
```

OBJEKTNO-RELACIONO MAPIRANJE

Objekti koji sačinjavaju XML stablo se mapiraju u tabele, svojstva objekata u kolone a međusobne veze između objekata u relacije primarni ključ / strani ključ

U objektno-relacionom mapiranju XML dokument se posmatra kao serijsko stablo povezanih objekata kojim se na jedan očigledan način ukazuje na relacije koje postoje između njih. *Objekti iz XML dokumenta se u bazu podataka mapiraju tehnikom objektno-relacionog mapiranja, što podrazumeva da se objekti mapiraju u tabele, svojstva objekata u kolone a međusobne veze između objekata u relacije primarni ključ / strani ključ.* (slika 1.3)

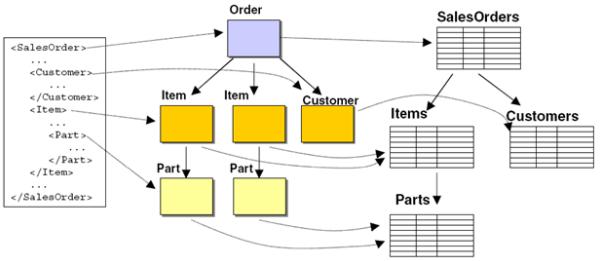
Mapiranje korisnika i adresa u SQL tabele - XML-stablo mapira se u SQL tabele:

```
Korisnici>
    <Korisnik>
```

```
        <ID>1</ID>
        <Ime>John</Ime>
        Doe</Prezime>
        <Adresa>
            <Grad>New York</Grad>
            <Drzava>USA</Drzava>
        </Adresa>
    </Korisnik>
    <!-- ... ostali korisnici ... -->
</Korisnici>
```

```
CREATE TABLE Korisnici (
    ID INT PRIMARY KEY,
    Ime VARCHAR(50),
    Prezime VARCHAR(50),
    Adresa_ID INT,
    FOREIGN KEY (Adresa_ID) REFERENCES Adrese(ID)
);

CREATE TABLE Adrese (
    ID INT PRIMARY KEY,
    Grad VARCHAR(50),
    Drzava VARCHAR(50)
);
```



Slika 1.3 Objektno-relaciono mapiranje [Izvor: NM IT350 - 2020/2021.]

Mapiranje porudžbine, proizvoda i korisnika:

```
<Porudzbine>
    <Porudzbina>
        <ID>1</ID>
        <Datum>2023-01-01</Datum>
        <Proizvod>
            <Naziv>Laptop</Naziv>
            <Cena>1200</Cena>
        </Proizvod>
        <Korisnik>
            <ID>1</ID>
            <Ime>John</Ime>
        </Korisnik>
    </Porudzbina>
```

```
    <!-- ... ostale porudžbine ... -->
</Porudzbine>
```

```
CREATE TABLE Porudzbine (
    ID INT PRIMARY KEY,
    Datum DATE,
    Proizvod_ID INT,
    Korisnik_ID INT,
    FOREIGN KEY (Proizvod_ID) REFERENCES Proizvodi(ID),
    FOREIGN KEY (Korisnik_ID) REFERENCES Korisnici(ID)
);

CREATE TABLE Proizvodi (
    ID INT PRIMARY KEY,
    Naziv VARCHAR(50),
    Cena DECIMAL(10, 2)
);

CREATE TABLE Korisnici (
    ID INT PRIMARY KEY,
    Ime VARCHAR(50)
);
```

PRIMER ZA OBJEKTNO-RELACIONO MAPIRANJE

Daje se nastavak primera

SQL Tabele:

```
CREATE TABLE Porudzbine (
    ID INT PRIMARY KEY,
    Datum DATE,
    Proizvod_ID INT,
    Korisnik_ID INT,
    FOREIGN KEY (Proizvod_ID) REFERENCES Proizvodi(ID),
    FOREIGN KEY (Korisnik_ID) REFERENCES Korisnici(ID)
);

CREATE TABLE Proizvodi (
    ID INT PRIMARY KEY,
    Naziv VARCHAR(50),
    Cena DECIMAL(10, 2)
);

CREATE TABLE Korisnici (
    ID INT PRIMARY KEY,
    Ime VARCHAR(50)
);
```

Ovi primeri ilustruju kako se XML struktura može mapirati u SQL tabele, a međusobne veze između objekata se ostvaruju putem stranih ključeva.

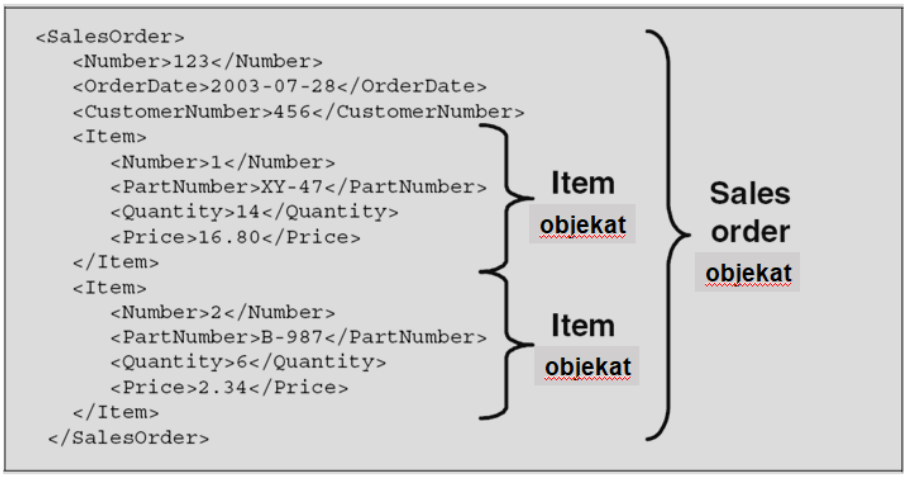
Važno je napomenuti da u stvarnim projektima, ORM alati, kao što su Hibernate za Java ili Entity Framework za .NET, često automatski obavljaju ovo

mapiranje, čime olakšavaju razvoj i održavanje aplikacija.

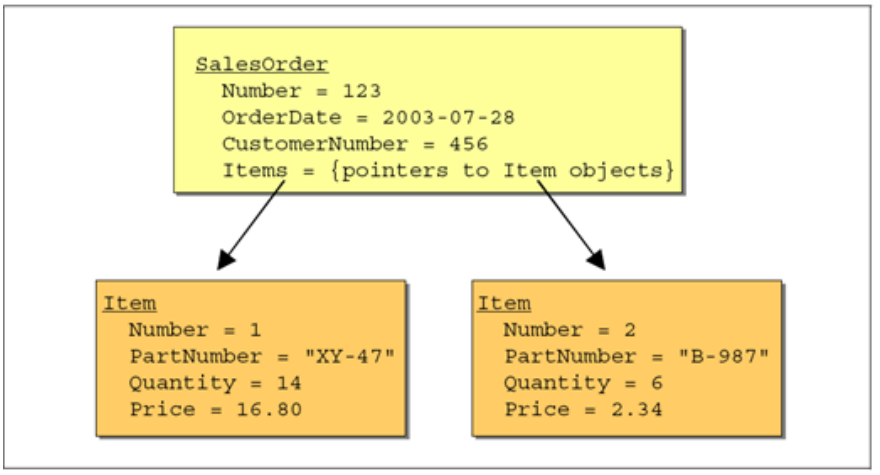
PRIMER OBJEKTNO RELACIONOG MAPIRANJA

Na osnovu XML dokumenta koji sadrži porudžbenicu se primenom objektno relacionog mapiranja dobijaju tabele SalesOrder i Items

Tako se XML dokument sa slike 1.4 koji sadrži porudžbenicu, korišćenjem objektno-relacionog mapiranja može videti kao serijsko stablo objekata prikazano na slici 1.5, a zatim se ono može mapirati u tabelu **SalesOrder** i tabelu **Items**, slika 1.6.



Slika 1.4 XML dokument kojim je opisana porudžbenica [Izvor: Autor]



Slika 1.5 Serijsko stablo objekata [Izvor: Autor]

Tabela 1. SalesOrder

Number	Date	Customer
123	2007-08-07	456

Tabela 2. Item

SONumber	Number	PartNumber	Quantity	Price
123	1	"xy-47"	14	14.56
123	2	"ab-567"	6	25

Slika 1.6 Tabele dobijene primenom objektno relacionog mapiranja [Izvor: Autor]

Poglavlje 2

Native XML baze podataka

XML MODEL PODATAKA

Opisuje koji su delovi XML dokumenta logički značajni.

Native XML baza podataka je baza podataka koja:

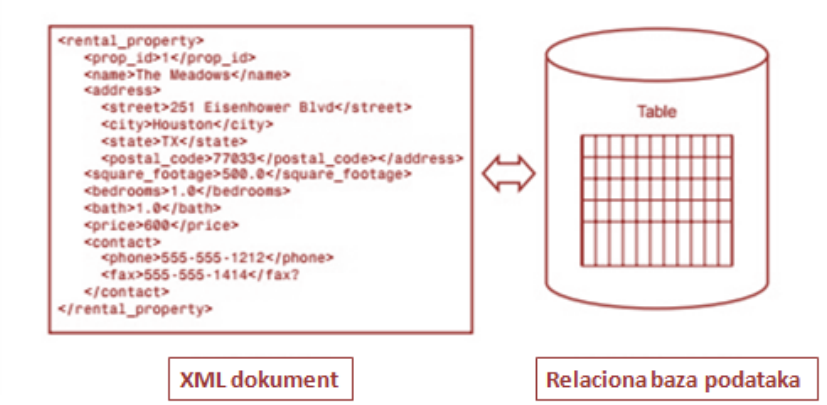
1. Definiše XML model podataka. Minimalni model uključuje elemente, atribute, tekst i redosled dokumenata
2. XML dokument koristi kao osnovnu logičku jedinicu za pamćenje podataka

Može koristiti bilo koju fizičku strategiju za pamćenje

XML modeli podataka se koriste kao **osnova za XML upitne jezike i definišu minimalnu količinu informacija koju native XML baze podataka moraju da pamte.**

Na primer: svi XML modeli podataka uključuju elemente, atribute, tekst i redosled dokumenata, dok neki modeli uključuju stvari kao što su reference na entitete i sekcije CDATA a drugi ne.

Native XML baze podataka su slobodne da definišu svoje sopstvene XML modele jer u vreme kada su se one prvi put pojavile, nisu postajali standardizovani XML modeli. Tako među njima postoje native XML baze koje se zasnivaju na info setu, DOM objektima, XPath modelu podataka i XQuery modelu podataka. Mnoge native XML baze podataka danas podržavaju XPath 1.0 i njihov model podataka s tendencijom da mnoge od njih u budućnosti koriste XQuery model podataka.



Slika 2.1 Mapiranje XML dokumenata u polja baze podataka [Izvor: Autor]

POTREBA ZA NATIVE XML BAZAMA PODATAKA

Postoji kada treba upravljati: dokumentima, polu-struktuiranim podacima, transakcijama čije izvršenje traje dugo ili kada se radi o arhiviranju dokumenata

Najbolji način da se razume potreba za native XML bazama podataka je analizirati primere iz realnog okruženja.

Na primer:

1. Upravljanje dokumentima kao što je korisnička dokumentacija, marketinške brošure, Web stranice itd. Native XML baze podataka posebno odgovaraju ovim slučajevima jer se XML modeli podataka dobro uklapaju u ovu vrstu dokumenata.

2. Polustruktuirani podaci koji se takođe dobro uklapaju u XML model podataka iz dva razloga. Prvo, XML model podataka može smestiti razgranatu šemu bez gubljenja prostora. Drugo, XML model podataka je proširiv, pa se ne zahteva fiksna šema.

3. Transakcije čije izvršenje traje dugo. Mnoge transakcije elektronske trgovine danas koriste XML dokumente kao način za razmenu informacija između različitih delova aplikacije. Pošto ove transakcije često mogu da traju nekoliko nedelja (npr, svaki deo mora da bude potvrđen od strane nekog ovlašćenog lica), native XML baze podataka su dobar način za pamćenje jednog stanja transakcije dok je ona u fazi izvršenja, čak i kada se krajnje stanje te transakcije pamti u relacionoj bazi podataka. Native XML baza podataka dozvoljava da se nad tekućim stanjem transakcije postavi upit korišćenjem XML upitnih jezika, a omogućava i korišćenje drugih XML alata kao što je XSLT transformacija.

4.Arhiviranje dokumenata. Mnoge kompanije, kao što su one u farmaceutskoj i finansijskoj industriji, moraju da arhiviraju dokumente jer ih na to obavezuju pravni propisi. Ako su to XML dokumenti, tada je native XML baza podataka prirodni izbor za arhiviranje, jer njena podrška za XML upitni jezik dozvoljava da se dokumenti koriste kao izvor istorijskih podataka koji se često koriste pri analizi trendova.

OSNOVNA JEDINICA ZA PAMĆENJE PODATAKA

U native XML bazama to je dokument

Osnovna jedinica za pamćenje podataka je najmanja grupa logički povezanih podataka koja se pamti. U relacionim bazama podataka to je red, a u native XML bazama osnovna jedinica za pamćenje podataka je dokument. Obzirom da je bilo koji fragment dokumenta koji je označen kao jedan element potencijalno XML dokument, izbor šta će činiti jedan XML dokument je čisto stvar odluke dizajnera.

Na primer:

*1. **Knjiga pisana u XML-u.** Mada će se mnogi ljudi složiti da knjiga treba da bude podeljena na više dokumenata, a da je takođe smešno da za svaki paragraf postoji poseban dokument,*

nije jasno šta je idealna veličina dokumenta. Na primer, treba li svaka knjiga da sadrži poglavlje? Sekciju? Podsekciju? To u potpunosti zavisi od knjige i uslova pod kojim će se ona koristi.

*2. **Medicinski podaci zapamćeni kao XML dokumenti.** Poseban XML dokument može postajati za svakog pacijenta, za svakog pružaoca medicinskih usluga (doktor, bolnica, klinika itd.) i za svaku osiguravajuću kuću. Na taj način je šema baze podataka do izvesnog stepena normalizovana, mada ta normalizacija nije kompletna u meri u kojoj se to može postići u relacionoj bazi podataka.*

*3. **Search engine XML dokumenata**. U ovom slučaju, nisu pravljeni pokušaji da se nađe idealna veličina dokumenta već se svaki dokument jednostavno pamti u bazi podataka bez obzira da li je njegova veličina idealna.*

NAČIN NA KOJI SE MOGU IMPLEMENTIRATI NATIVE XML BAZE PODATAKA

Pamćenjem celih dokumenata, parsiranjem XML dokumenata u fiksne skupove tabela, DOM stabla ili indeksirani skup heš tabela

Native XML baze podataka se mogu implementirati na jedan od sledećih načina. Pamćenjem

1. Celih XML dokumenata u CLOB strukturama relacionih baza podataka i indeksiranjem individualnih vrednosti elemenata i atributa.
2. **Parsiranih XML dokumenata u fiksnim skupovima tabela** (elementi, atributi, tekst itd.) u relacionoj bazi podataka.
3. Parsiranih XML dokumenata kao DOM stabla u objektno-orijentisanoj bazi podataka.
4. Parsiranih XML dokumenata u indeksiranom skupu heš tabela

Od načina na koji je native XML baza podataka implementirana umnogome zavise njene performanse. Najznačajniji faktor koji utiče na performanse native XML baza podataka je da li dokument zapamćen u celini ili u parsiranoj formi.

Na primer, dokument koji je zapamćen u indeksiranom CLOB-u, je zapamćen ceo. Dokument koji je zapamćen kao DOM stablo u objektno orijentisanoj bazi podataka je zapamćen u parsiranoj formi, a samo parsiranje se izvodi u trenutku insertovanja kada se pamte individualni elementi i atributi.

Da bi se sagledalo kako to utiče na performanse, mogu se razmotriti sledeće klase upita:

1. Upiti kojima se pretražuje čitav dokument a koji se baziraju na indeksiranim vrednostima. Na takve upite bolje performanse pruža baza podataka koja pamti ceo dokument, jer ne postoji potreba da se dokument ponovo kreira tj. da se izvrši njegova ponovna serijalizacija, što je slučaj kada je dokument zapamćen u parsiranoj formi. Upiti kojima se pretražuju indeksirane vrednosti. Za datu ekvivalentnu šemu indeksiranja, ovakve upite izvršavaju jednako sve baze podataka, jer se oni mogu rešiti jednostavno pretraživanjem indeksa.

2. Upiti kojima se pretražuju neindeksirane vrednosti ili fragmenti dokumenata. Ovaj upit bolje izvršavaju baze podataka koje pamte prethodno parsirane dokumente, jer baze koje pamte cele dokumente moraju da svaki dokument parsiraju kako bi ovakav upit rešile.
3. Upiti koji ažuriraju dokumenta. Ove upite najbolje izvršavaju baze podataka koje pamte prethodno parsirana dokumenta, jer one mogu da ažuriraju, insertuju ili brišu pojedine čvorove. Baze podataka koje pamte cela dokumenta ne moraju samo da parsiraju i modifikuju dokument; one moraju takođe da pretraže čitav dokument da bi ga zapamtile posle ažuriranja.

Poglavlje 3

XQuery

ČEMU SLUŽI XQUERY?

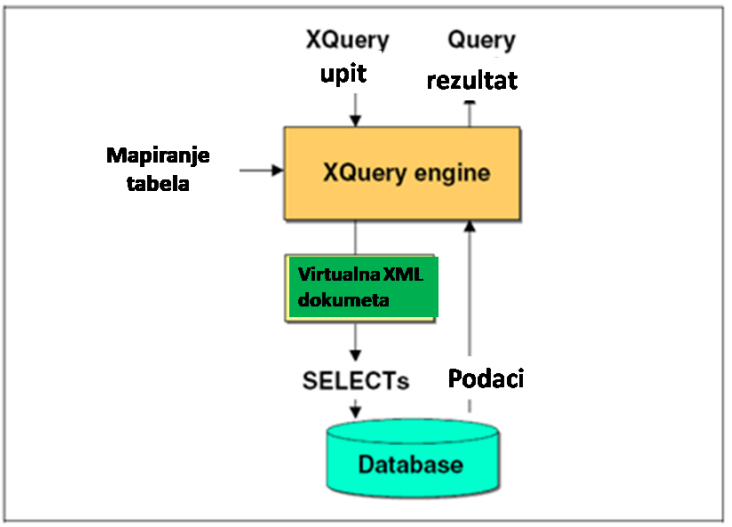
XQuery je dizajniran za upite nad XML dokumentima

XQuery je XML upitni jezik koji je definisan od strane W3C i koji još uvek nije završen. Međutim, već postoji više od dvadeset implementacija XQuery-a.

On je dizajniran za upite nad XML dokumentima a ne nad relacionim bazama podataka. Da bi bio implementiran nad relacionom bazom, neophodno je prvo mapirati relacionu bazu na jedan ili više virtualnih dokumenata. Najlakši način da se to uradi je mapirati svaku tabelu u poseban dokument korišćenjem table-based mapiranja. Tako se XQuery naredba za upit nad virtualnim dokumentom može mapirati u SELECT naredbu za upit nad tabelom (slika 3.1)

Primer jedne XQuery naredbe glasi:

```
FOR $c IN document("customers")/row/Customer
    RETURN
        <Customer>
            <Name>{$c/name}</Name>
            <ID>{$c/id}</ID>
        </Customer>
```



Slika 3.1 Implementacija XQuery jezika nad relacionim bazama podataka [Izvor: Autor]

PRIMER XQUERY

XQuery koristi funkcije za izdvajanje podataka iz XML dokumenta

Dokument "books.xml":

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Irish</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

Kako izabrati čvore iz XML dokumenta "books.xml"?

XQuery koristi funkcije za izdvajanje podataka iz XML dokumenta:

Funkcija doc() se koristi za otvaranje file-a "books.xml" file:

doc("books.xml")

XQuery koristi tzv. path izraze za navigaciju (prolazak) kroz elemente u XML dokumentu. Sledeći path izraz se koristi za selektovanje svih elemenata naslova (engl. title) u "books.xml" file:

doc("books.xml")/bookstore/book/title

(/bookstore selektuje elemente bookstore, /book selektuje sve elemente book ispod elementa bookstore, a /title selektuje sve elemente title ispod elemenata book). XQuery-em će se izdvojiti sledeće:

```
<title lang="en">Everyday Irish</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

PRIMER XQUERY-A SA PREDIKATIMA; FLWOR

XQuery koristi predikate za ograničavanje izdvojenih elemenata iz XML dokumenata; FLWOR se može koristi umesto izraza path sa predikatima.

XQuery koristi predikate za ograničavanje izdvojenih elemenata iz XML dokumenata.

Sledeći predikat je iskorišćen za selektovanje svih elemenata book ispod elementa bookstore koji imaju element price sa vrednošću koja je manja od 30.

doc("books.xml")/bookstore/book[price<30]

Gornji XQuery će izdvojiti sledeće elemente:

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Izraz:

doc("books.xml")/bookstore/book[price>30]/title

predstavlja selektovanje svih elemenata book ispod elementa bookstore koji imaju element price sa vrednošću koja je veća od 30.

Sledeći FLWOR izraz će selektovati isto što i path izraz:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

Rezultat će biti:

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Sa izrazom FLWOR rezultati se mogu sortirati

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

pa ćemo dobiti izlaz:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

POREĐENJE XQUERY-SQL

Upitni jezik za pretraživanje Native XML baze XQuery za XML je isto što je i SQL za relacione baze podataka

XQuery i SQL su dva različita jezika za upravljanje podacima, iako imaju sličnosti u smislu radnji koje obavljaju (upiti nad podacima).

Navodimo nekoliko ključnih razlika i poređenje između XQuery i SQL:

Namena:

SQL: SQL je standardizovan jezik za rad sa relacionim bazama podataka. Koristi se za definisanje, manipulaciju i upite nad tabelarno organizovanim podacima.

XQuery: XQuery se koristi za upite nad XML podacima. Namena mu je izdvajanje, transformacija i manipulacija podacima u XML formatu.

Podaci:

SQL: Rad s tabelarno organizovanim podacima (relacijama). Podaci su organizovani u redove i kolone.

XQuery: Rad s XML dokumentima. Podaci su organizovani u hijerarhijskoj strukturi stabla.

Sintaksa:

SQL: SQL ima svoju sintaksu optimizovanu za rad s tabelama, JOIN-ovima, indeksiranjem i drugim relacionim konceptima.

XQuery: XQuery ima sintaksu prilagođenu za rad s XML strukturama. Koristi se za navigaciju kroz XML elemente, selektovanje čvorova i transformaciju podataka.

Upiti:

SQL: SQL upiti su fokusirani na setove podataka i često koriste agregacije, JOIN-ove, i uslove.

XQuery: XQuery upiti su fokusirani na izdvajanje i manipulaciju XML elemenata i atributa. Koriste se za filtriranje, transformaciju i konstrukciju XML dokumenata.

Primena:

SQL: Često se koristi u aplikacijama koje koriste relacione baze podataka (MySQL, PostgreSQL, SQL Server, itd.).

XQuery: Često se koristi u aplikacijama koje rade s XML podacima, kao što su web servisi, aplikacije za razmenu podataka u XML formatu, itd.

Tipovi podataka:

SQL: Rad s tipovima podataka poput INTEGER, VARCHAR, DATE, itd.

XQuery: Rad s XML tipovima podataka, uključujući elemente, atribute, tekst, sekvence i druge.

DODATNO O POREĐENJU XQUERY-SQL

Daju se dodatni komentari o poređenju XQuery i SQL

Transakcije:
SQL:SQL često podržava transakcije koje omogućavaju grupisanje više upita u jednu koherentnu celinu.
XQuery: XQuery ne podržava transakcije na isti način kao SQL, jer je često korišćen za jednostavnije operacije nad XML podacima.

Dakle, SQL i XQuery su specifični jezici koji su prilagođeni za rad sa različitim vrstama podataka. Kada radite s relacionim bazama podataka, koristićete SQL, dok će se XQuery koristiti kada radite s XML podacima.

Poglavlje 4

NoSQL baze podataka

DA LI SU RELACIONE BAZE POGODNE ZA ČUVANJE POLUSTRUKTUIRANIH PODATAKA?

Dok su relacione baze podataka prvobitno namenjene za smeštanje tabelarnih struktura, one se nisu dobro pokazale pri pokušaju modeliranja ad-hoc relacija između polustruktuiranih podataka

Istorijski, mnoge web aplikacije se zasnivaju na relacionim bazama podataka. Ali i proteklih godina, bili smo suočeni sa porastom količine podataka, veoma brzim promenama i različitim strukturama podataka koje su se teško mogle čuvati u relacionim bazama podataka.
Programeri su pokušali da povezane, polustruktuirane skupove podataka čuvaju u relacionim bazama podataka. Ali dok su relacione baze podataka prvobitno namenjene za smeštanje tabelarnih struktura –za šta su one posebno dobre – one se nisu najbolje pokazale pri pokušaju modeliranja ad-hoc relacija koje se pojavljuju u realnom svetu. U jeziku koji se koristi za pristup relacionim bazama podataka, relacije ne postoje, one se samo javljaju za vreme modeliranja kao način za spajanje tabela.

Relacioni model može postati neefikasan u slučaju spajanja velikog broja tabela, kada su redovi tabela slabo popunjeni ili kada u njima ima previse null vredsti. Povećanjem broja veza u relacionim bazama podataka znači i povećanje broja spajanja između tabela što smanjuje performanse i otežava prilagođavanje postojećih baza podataka promenama iz okruženja.

Vreme izvršenja upita raste sa veličinama tabela i porastom broja join-a (tzv. join pain). Ulažući napor da se ovo spreči i izađe na kraj sa velikim skupovima podataka, u svetu NoSQL-a je usvojeno nekoliko alternativa relacionim bazama podataka. Iako vešti u baratanju sa velikim skupovima podataka, ovi alternativni modeli imaju tendenciju da budu manje skupi od relacionih. (Osim graf modela koji su mnogo skuplji).
Ali, količina nije jedini problem modernih web sistema. Pored toga što se suočavamo sa velikom količinom podataka, danas se podaci menjaju veoma brzo. Stopa kojom se podaci menjaju tokom vremena se naziva brzina (eng. velocity).

ZAŠTO RELACIONE BAZE NISU POGODNE?

Kod relacionih baza podataka se javlja problem čvrsto povezanih domena

Kod relacionih baza podataka se javlja problem čvrsto povezanih domena. Da bi razumeli cenu ili performanse povezanih (rekurzivnih) upita u relacionim bazama podataka,

pogledajmo ne tako jednostavne primere iz domena socijalnih mreža. Slika 4.1 prikazuje jednostavan upit za spajanje tabela kako bi se predstavilo prijateljstvo između različitih osoba.

Postavlja se pitanje ko je prijatelj Bobu?" to je jednostavno kao što se vidi u *primeru 1.*

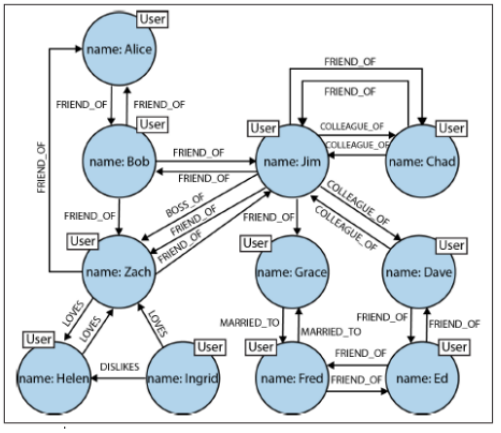
```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
ON PersonFriend.FriendID = p1.ID
JOIN Person p2
ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Primer 1. SELECT naredba kojom se mogu naći Bobovi prijatelji
Odgovor je Alice i Zach. Ovo nije posebno skup i težak upit, jer je u njemu ograničen broj slogova korišćenjem filtera WHERE Person.person='Bob'.

U primeru 2, postavljamo obrnuto pitanje:Ko je prijatelj sa Bobom ? Odgovor na ovaj upit je Alice; na žalost, Zach ne smatra Boba za svog prijatelja.

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
ON PersonFriend.PersonID = p1.ID
JOIN Person p2
ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Primer 2: SELECT naredba kojom se može naći odgovor: Ko je prijatelj sa Bobom?



Slika 4.1 Modeliranje veze friends i friends-of-friends u relacionim bazama podataka [Izvor: NM IT350 - 2020/2021.]

Ovaj obrnuti upit je još uvek lako implementirati, ali sa strane baze podataka on je skuplji, jer se sada moraju uzeti u obzir svi slogovi u tabeli PersonFriend. Možemo dodati indeks, ali i to još uvek neće rešiti naš problem.

SITUACIJE U KOJIMA RELACIONE BAZE NISU POGODNE

Da bi se dobile hijerarhijske veze koje postoje u SQL-u treba koristiti rekurzivni join, koji upit čini sintaksno mnogo složenijim

Stvari postaju mnogo problematičnije kada pitamo “ko su prijatelji mojih prijatelja?” Da bi se dobile hijerarhijske veze koje postoje u SQL-u treba koristiti rekurzivni join, koji upit čini sintaksno mnogo složenijim, kao što je prikazano u *primeru 3.*

```
SELECT p1.Person AS PERSON,
p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
ON pf2.PersonID = pf1.FriendID
JOIN Person p2
ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice'
AND pf2.FriendID <> p1.ID
```

Primer 3. Alisini prijatelji prijatelja

Ovaj upit je složen bez obzira što se odnosi samo prijatelje Alisinih prijatelja i ne ide dublje u Alisinu socijalnu mrežu

KARAKTERISTIKE NOSQL BAZA PODATAKA

NoSQL baze podataka nisu bazirane na šemi; u bazi se umesto relacija čuvaju agregacije informacija; pamte se na jeftinijem hardvru i visoko su distribuirane

Četiri osnovne funkcionalnosti u kojima se ogledaju razlike između NoSQL baza podataka i standardnih SQL sistema su:

1.Nepotrebnost šeme: Šema baze podataka je opis svih mogućih podataka i njihovih tipova koji se u bazi mogu naći. NoSQL nije baziran na šemi, tako da šema nije neophodna. Korisnici ovim dobijaju fleksibilnost u načinu na koji strukturiraju podatke.

2. Relacionalnost: U relacionim bazama pravimo mnogo relacija između tabela u kojima čuvamo podatke. Na primer, lista transakcija može da bude povezana sa korisnicima. Sa NoSQL bazama ova informacija je sačuvana kao agregacija, jedinstveni slog koji sadrži sve u vezi transakcija, uključujući i podatke o korisniku.

3. Jeftiniji hardver: Neke baze su dizajnirane da rade samo na najboljim serverima, NoSQL baza je drugačija i omogućava korisnicima da korišćenjem jeftinijih servera i spajanjem više njih postignu izuzetne performanse.

4. Visoka distribuiranost: Distribuirane baze mogu da skladište podatke na više od jednog uređaja. Sa NoSQL-om, klaster servera može da sadrži jednu veliku bazu podataka. NoSQL proizvođači olakšavaju rad svojim korisnicima (programerima koji koriste NoSQL). Tehnologija može brzo biti prihvaćena samo ako je razvojni timovi vide kao jeftinu alternativu. U procesu razvoja se mogu brže i efikasnije prevazići postojeći problemi. Mogućnost prilagođavanja hardveru i brzo izbacivanje novih servisa su osnovne karakteristike NoSQL implementacija. Sa NoSQL je moguće mnogo brže primeniti iterativni razvoj i dodavanje novina u odnosu na DBMS. Primenom NoSQL principa se ne smanjuje samo cena i olakšava život programerima, već se rešava i niz menadžerskih problema.

Poglavlje 5

Konzistentnost NoSQL baza podataka

ŠTA JE KONZISTENTNOST?

ACID i BASE konzistentnost

NoSQL baze podataka se odlikuju BASE konzistentnošću.

Konzistentnost **je atribut baze kojioznačava kakav pogledna podatke ima baza podataka tj. da li svako čitanje posle upisivanja odmah prikazuje upisanu vrednosti podataka ili ne**. Visoka konzistentnost je potrebna kod proizvoda koji koriste transakcije.

Konzistentnost se često deli u dva nivoa:

1. ACID konzistentnost: predstavlja atomske transakcije i obično znači da će svaki upit nakon upisa videti izmenjene podatke.
2. BASE konzistentnost: **znači da će upisani podaci pre ili kasnije početi da se prikazuju u upitima.**

Ne postoji jedinstveno mišljenje o tome koje je rešenje je bolje, istina je da nijedno nije uvek idealno i da je fleksibilnost u izboru rešenja jako bitna za razvoj visoko kvalitetnog softvera.

ACID KONZISTENTNOST

Generalni skup principa za transakcione sisteme dok ih većina NoSQL baza podataka ne podržava

U svetu relacionih baza podataka smo familijarni sa ACID transakcijama kojima se garantuje sigurno okruženje u kojem se radi sa podacima. ACID je generalni skup principa za transakcione sisteme i nije vezan jedino za baze, već se javlja i u mnogo drugih oblasti.

ACID znači:

1. Atomičnost (eng. Atomic): Sve operacije u transakciji moraju biti uspešne ili se transakcija mora poništiti (rolled back).
2. Konzistentnost (eng. Consistent): Po završetku transakcije, baza podataka je u konzistentnom (ispravnom) stanju.
3. Izolovanost (eng. Isolated): Akcije transakcija se sekvencijalno izvršavaju jedna nakon druge. Nije moguće nesekvencijalno, naizmenično izvršenje akcija transakcija.

4. Trajnost (eng. Durable): Rezultat primene transakcija je trajan, osim ako ne postoji greška.

Ove osobine znače da kada se transakcija završi, njeni podaci su konzistentni i stabilni na disku. Ovo je dobro za one koji rade na razvoju aplikacija, ali zahteva sofisticirano zaključavanje koje može prouzrokovati nedostupnost podataka i obično predstavlja težak uzorak (eng. pattern) za većinu slučajeva korišćenja.

Neki sistemi NoSQL baza podataka kao što su Foundation DB, MarkLogic, i Neo4j predstavljaju značajnije izuzetke u smislu nepodržavanja ovih principa. Sa MongoDB, svi podaci u bazi mogu da budu blokirani dok se upis ne desi i na taj način se blokiraju sva čitanja iz baze sve dok se upis ne završi.

BASE KONZISTENTNOST

BASE znači da baza nema ACID garancije; Sistem je primenjiv na socijalnim mrežama

U NoSQL svetu, ACID transakcije se ne koriste obzirom da kod njih ne postoji striktan zahtev za trenutnom konzistentnošću, osvežavanjem podataka i tačnošću, kako bi se postigli neki drugi benefiti. Umesto korišćenja ACID, pojavio se termin BASE kao popularan način za opisivanje svojstva optimističnije strategije čuvanja podataka.

BASE konzistentnost **znači da**baza nema ACID garancije. Ovo je dosta interesantno kod sistema koji su implementirani kao veliki klasteri, neki delovi sistema mogu da sadrže samo podatke za čitanja koji bivaju povremeno ažurirani. BASE ima sledeće značenje:

1. Osnovna raspoloživost (eng. Basic availability): Ima se utisak da se sve vreme vrši pamćenje podataka
2. Soft-state: Pamćenje ne mora da bude konzistentno sa upisom, niti različita ažuriranja moraju biti međusobno konzistentna sve vreme.
3. Konačna konzistentnost (eng. Eventual consistency): Konzistentnost se javlja u nekom kasnijem trenutku

BASE svojstva su značajno labavija u odnosu na ACID garancije i između njih se ne može uspostaviti direktno preslikavanje. U BASE pamćenju podataka, podaci postaju raspoloživi, ali se ne garantuje konzistentnost u trenutku upisa. BASE pamćenje ne obezbeđuje striktnu sigurnost, podaci postaju konzistentni kasnije, možda u vreme čitanja, ili će biti konzistentni ali samo za neke procese. **Prednost BASE pristupa je što se transakcije sporije komituju, što znači i brže čitanje.**

Mane ovoga su što klijenti koji se konektuju da čitaju replike podataka mogu da vide informacije koje više nisu validne. Ovakav sistem je primenljiv na socijalnim mrežama, ukoliko napravite post na facebook-u, a vaši prijatelji to vide tek kroz nekoliko minuta, gubitak nije veliki, a poboljšanja u performansama će biti velika.

ACID ILI BASE KONZISTENTNOST?

ACID ili BASE biramo uzimajući u obzir brzinu izvršenja transakcije i bezbednost podataka.

Kao što možemo očekivati NoSQL proizvođači mogu različito da se odnose prema svojim bazama i da zauzmu karakterističan stav u određenim oblastima.

1. Mnoge NoSQL baze imaju kao cilj da koriste ACID, iako su trenutno bazirane na BASE sistemu, što pokazuje koliko je ACID garancija bitna za enterprise sisteme.
2. Mnoge kompanije koriste BASE konzistentne proizvode kada testiraju svoje ideje jer su besplatne, a ACID konzistentnost koriste za produkcione sisteme za koje plaćaju.
3. Najlakši način za određivanje da li ćemo koristiti ACID je da pogledamo kako naši upisi i čitanja funkcionišu. Ukoliko čitanja sa kašnjenjem ne zadovoljavaju naše poslovne potrebe, mora se koristiti ACID konzistentnost.
4. Za finansijske servise potreba za ACID konzistentnošću je očigledna. Pored finansijski ključnih aplikacija, takođe je neophodno razmotriti korišćenje ACID-a i u zdravstvu, vojnim i sličnim sistemima.

Zaključak je da ACID ili BASE konzistentnost biramo uzimajući u obzir brzinu izvršenja transakcije i bezbednost podataka.

UVOD U NOSQL BAZE PODATAKA - VIDEO

Video tutorijal o uvodu u NoSql baze podataka

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 6

Modeli za čuvanje podataka u NoSQL bazama

ČUVANJE DOKUMENATA PO ID-U

Na najjednostavnijem nivou, dokumenti se čuvaju i pretražuju po ID. U opštem slučaju, pamćenje i nalaženje podataka se bazira na indeksima koji olakšavaju pristup dokumentima.

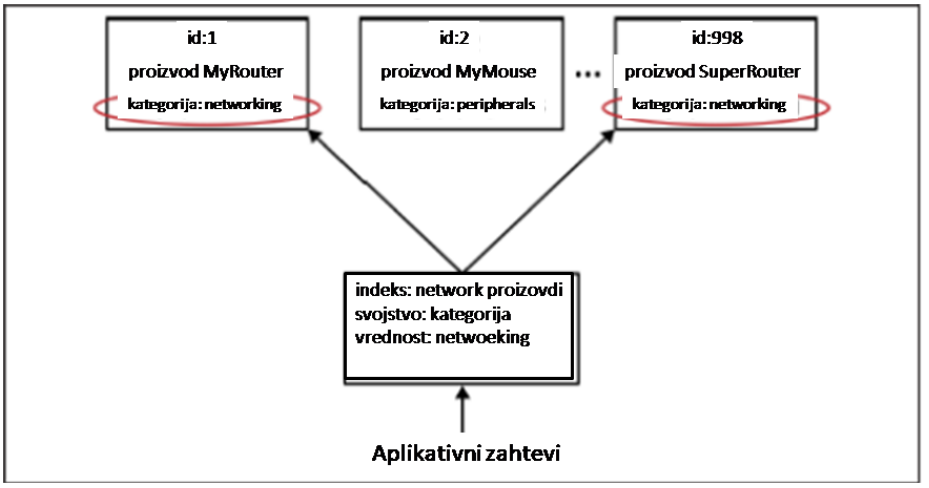
U NoSQL bazama podataka korite se sledeći model podataka:

1. ***čuvanje dokumenata po ID-u,***
2. ***čuvanje ključeva-vrednosti (eng.*** Key-Value Stores***),***
3. ***čuvanje familija kolona (eng.*** Column Family***) i***
4. ***graf baze podataka***

Čuvanje dokumenata po ID-u: Na najjednostavnijem nivou, dokumenti se mogu čuvati i pretraživati po ID. U opštem slučaju, pamćenje podataka se bazira na indeksima koji olakšavaju pristup dokumentima na osnovu njihovih atributa. Na primer, indeksi se mogu koristiti za predstavljanje različitih tipova proizvoda kako bi se nudili potencijalnim prodavcima, kao što je prikazano na slici 6.1.

Generalno, kod čuvanja dokumenata po ID-u u NoSQLbazama podataka indeksi se koriste da bi se pretražio skup povezanih dokumenata za potrebe neke aplikacije. Slično indeksima u relacionim bazama podataka, indeksi pri pamćenju dokumenata omogućavaju brže čitanje ali sporiji upis dokumenata.

Pošto zapamćeni dokumenti nisu fizički povezani, postoje mnogi optimistični mehanizmi kontrole konkuretnosti koji prilikom konkuretnog upisivanja sadržaja pojedinačnih dokumenata ne moraju koristiti striktne metode zaključavanja.



Slika 6.1 Indeksi upredmećuju skup entiteta u zapamćenim dokumentima [Izvor: Autor]

ČUVANJE KLJUČEVA-VREDNOSTI

Deluju kao velika distribuirana hashmap struktura podataka koja čuva i pretražuje nerazumljive vrednosti po ključu.

Čuvanje ključeva-vrednosti (eng. Key-Value) je sličan pamćenju familije dokumenata, i potiče sa Amazon's Dynamo baze podataka. On deluju kao velika distribuirana hashmap struktura podataka koja čuva i pretražuje nerazumljive vrednosti po ključevima. Prostor sa ključevima hashmap-a se širi preko velikog broja cilindara (eng. buckets) na mreži. Zbog tolerancije na greške, svaki cilindar se replicira na nekoliko mašina.

Sa tačke gledišta korisnika, **Key-Value način pamćenja jelak za korišćenje. Klijent čuva element podataka heširanjem nekog identifikatora koji je specifičan za domen problema (ključ).**

Heš funkcija je izrađena tako da omogućava podjednaku distribuciju kroz raspoložive cilindre omogućavajući tako da nijedna mašine ne postane žarište. **Pošto dobije ključ za heširanje, klijent može da koristi tu adresu da bi zapamtio vrednost u odgovarajućem cilindru. Sličan proces se koristi i pri pretraživanju zapamćenih podataka.**

U ovakvom modelu, aplikacije koje žele da pamte podatke ili ih pretražuju, potrebno je da znaju (ili izračunaju) odgovarajući ključ. Kako postoji veoma veliki broj mogućih ključeva u skupu ključeva, u praksi postoji tendencija da to budu prirodni ključevi iz domena problema: Username i email adresa, socijalni broj itd.

ČUVANJE FAMILIJA KOLONA

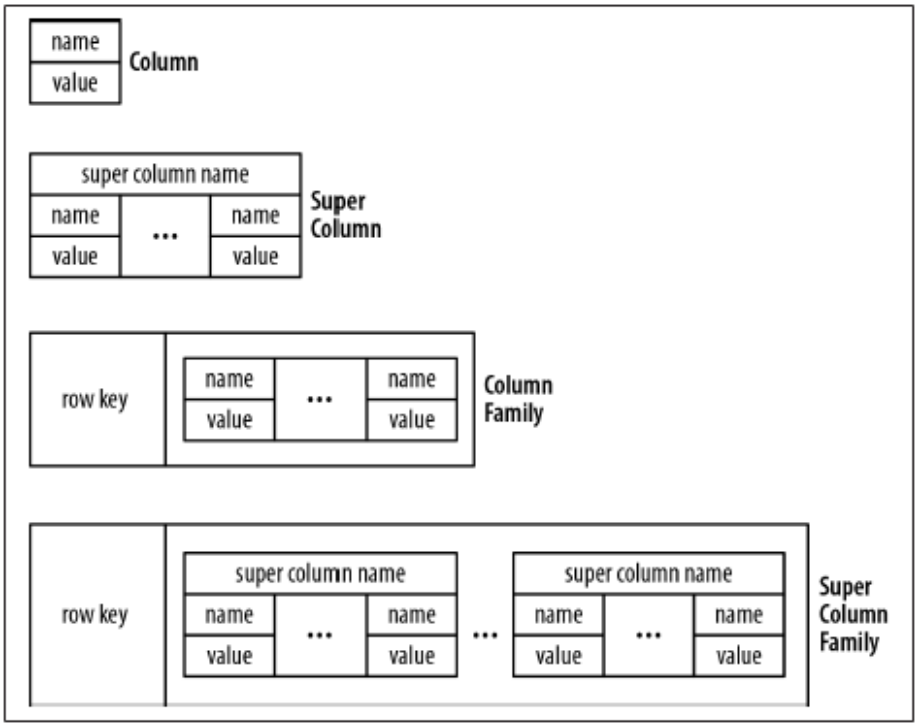
Kolone se pamte kao redovi tabele i kada red sadrži samo kolone, on je poznat kao familija kolona a kada red sadrži super kolonu, on je poznat kao super familija kolona

Čuvanje familija kolona (eng. Column Family) način čuvanja podataka je modelovan na Google's BigTable. Model podataka se bazira na retko popunjenim tabelama čiji redovi mogu da sadrže proizvoljne kolone, ključeve koji omogućavaju prirodno indeksiranje.

Na slici 6.2 se vide četiri zajednička bloka za izgradnju (eng. building blocks) koja se koriste u Column Family bazama podataka:

1. **Kolona** (eng. Column)
2. **Super kolona** (eng. Super Column) koja se dobija kombinacijom bilo kog broja kolona
3. **Familija kolona** (eng. Column Family) koja se dobija kada se više kolone zapamti kao red tabele koji ima svoj ključ
4. **Super familija kolona** (eng. Super column family) koja se dobija kada se više super kolona zapamti kao red tabele koji ima svoj ključ

Individualni redovi na taj način postaju važni u ovoj strukturi, jer oni omogućavaju umreženu hashmap strukturu u okviru koje denormalizujemo naše podatke.



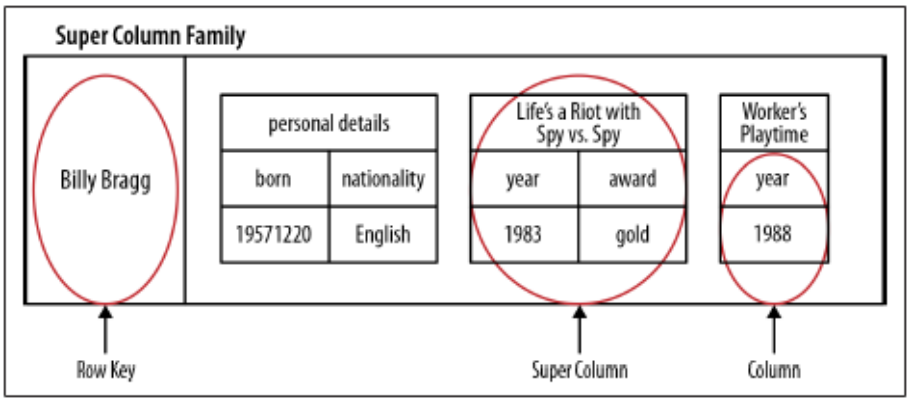
Slika 6.2 Četiri bloka za izgradnju column family čuvanja podataka [Izvor: NM IT350 - 2020/2021.]

KAKO SE ČUVAJU FAMILIJE KOLONA

U column family bazama podataka svaki red u tabeli predstavlja pojedinačan sveobuhvatni entitet

Na slici 6.3 je prikazano kako možemo mapirati zapis o umetniku i njegovim albumima u super column family strukturu – logički, što predstavlja ništa više nego mapiranje mapa.

U bazamapodataka zasnovanim na familiji kolona, svaki red u tabeli predstavlja pojedinačan sveobuhvatni entitet (npr. sve o umetniku). Te familije kolona su kontejneri za povezane delove podataka, kao što je ime umetnika i njegova diskografija.
U okviru familije kolona, pronalazimo vrednosti ključnih podataka, kao što su datum izdavanja albuma i datum rođenja umetnika.



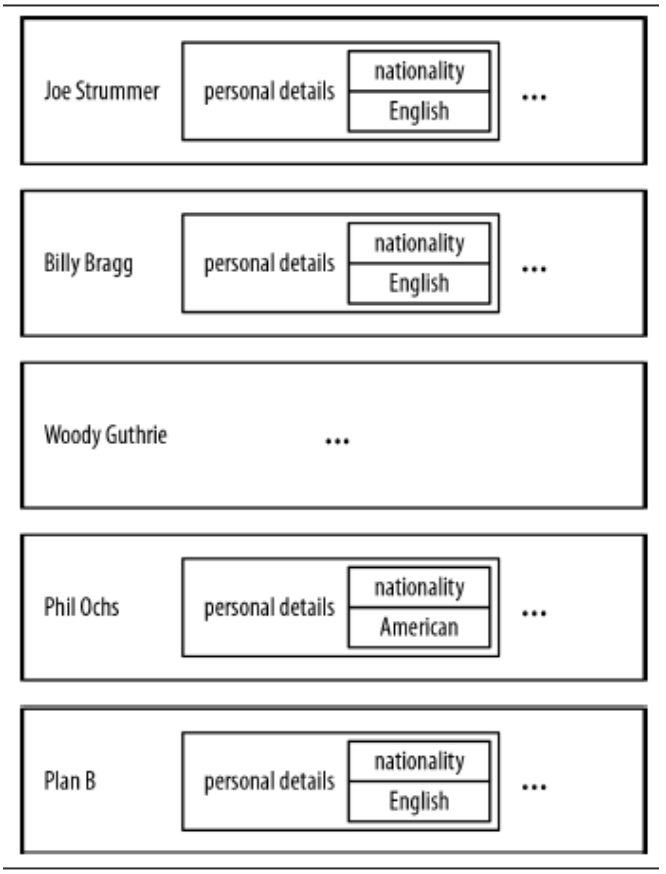
Slika 6.3 Pamćenje podataka u familiji super kolona [Izvor: NM IT350 - 2020/2021.]

O JOŠ JEDNOM NAČINU ZA ČUVANJE FAMILIJE KOLONA

Pogled orijentisan na redove se može okrenuti za 90 stepeni kako bi se došlo na pogled orijentisan na kolone.

Ovaj pogled orijentisan na redove (eng. row-oriented view) se može okrenuti za 90 stepeni kako bi se došlo na pogled orijentisan na kolone (eng. column oriented view).

Na primer, kao što se vidi sa slike 6.4 u stanju smo da nađemo sve redove gde je umetnik Englez. Odatle je lako izdvojiti kompletne podatke o umetniku iz svakog reda. Ovde se ne radi o povezanim podacima kao što je to slučaj u grafu, već se ovde samo daje neki uvid u skup povezanih entiteta.
Baze podataka zasnovane na familiji kolona se razlikuju od dokument baza podataka i načina čuvanja podataka ključevi - vrednosti, ne samo po svom izražajnijem modelu podataka već i po svojim operativnim karakteristikama.

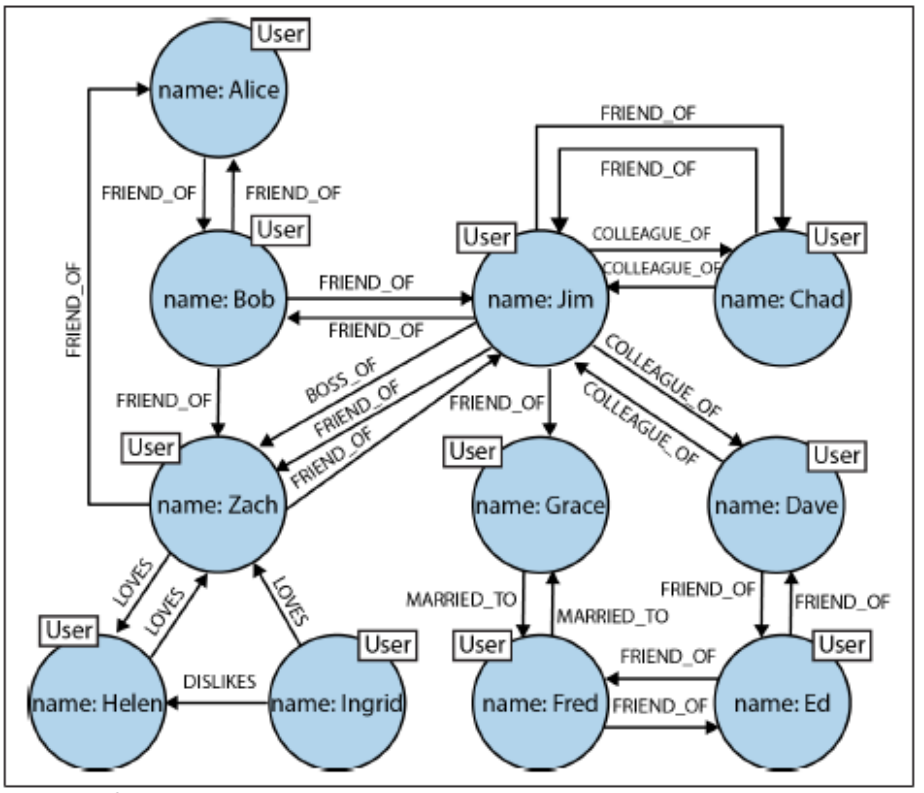


Slika 6.4 U column family bazama podataka ključevi formiraju prirodni indeks za redove [Izvor: NM IT350 - 2020/2021.]

GRAF BAZE PODATAKA

U graf bazama podataka, veze između entiteta ne prikazuju uniformnost na nivou domena već je domen promenljivo struktuiran

Nasuprot pamćenju podataka o kojem je do sada bilo reči, u graf bazama podataka (eng. Graph Databases), povezani podaci se pamte na diskovima kao povezani. Npr, razmotrimo socijalnu mrežu prikazanu na slici 6.5.
U ovoj socijalnoj mreži, kao i u mnogim slučajevima iz realnog okruženja veze između entiteta ne prikazuju uniformnost na nivou domena, već je domen promenljivo struktuiran (eng. variably-structured). Socijalna mreža je popularan primer gusto povezane, promenljivo struktuirane mreže koja je rezidentna i odgovara svim mogućim šemama. Naša jednostavna šema prijatelja može da raste u veličini i postaje sve bogatija.
Grafom se mreža može mnogo bolje predstaviti. Mi možemo videti ko koga voli (LOVES), koje je kome kolega (COLLEAGUE_OF), ko je kome šef (BOSS_OF), ko je s kim u braku (MARRIED_TO), takođe možemo videti ko se kome ne sviđa (DISLIKES veze).

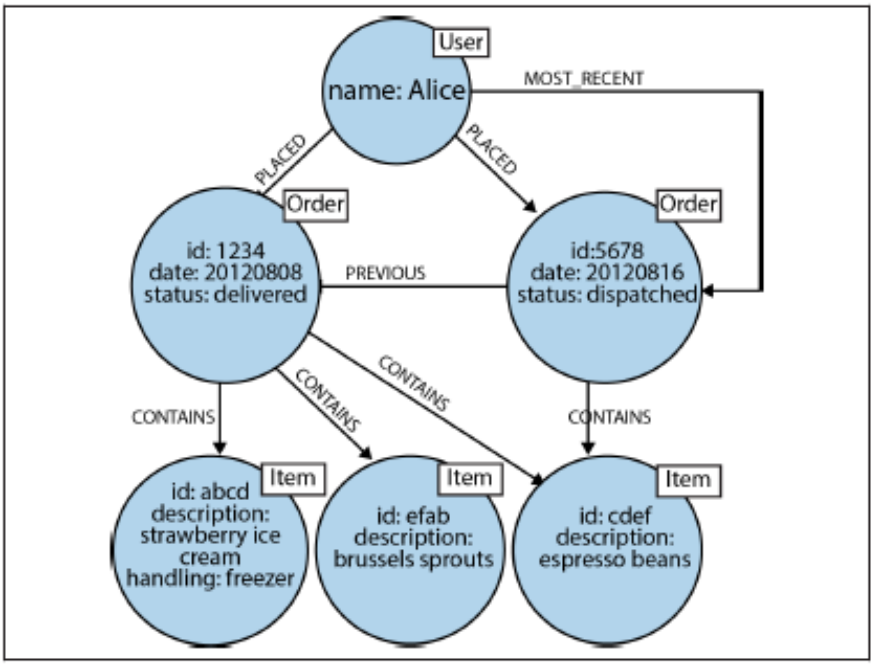


Slika 6.5 lako modeliranje veza prijateljstvo, kolega, šef, voli itd. korišćenjem grafa [Izvor: NM IT350 - 2020/2021.]

DA LI SE NOSQL BAZE MOGU KORISTITI UMESTO GRAF BAZA PODATAKA?

NoSQL baze podataka čuvaju skupove nepovezanih dokumenata/ vrednosti/kolona što ih takođe čini teškim za korišćenje u slučajevima povezanih podataka ili grafova.

Većina NoSQL baza podataka čuvaju skupove nepovezanih dokumenata/vrednosti/kolona. To ih čini teškim za korišćenje u slučajevima povezanih podataka ili grafova.



Slika 6.6 . Mala socijalna mreža realizovana korišćenjem agregatnog modela [Izvor: NM IT250 - 2020/ 2021.]

Ako proučimo agregatni model za spajanje podataka koji je prikazan na slici 6.6, možemo zamisliti relacije koje postoje između njih.

Pogledajmo referencu na narudžbinu:1234 u slogu čiji je user: Alise, možemo zaključiti da postoji relacija između user : Alice i order :1234. To nam daje pogrešnu nadu da korišćenjem ključeva i vrednosti možemo dobiti strukturu grafa.

Ali, u ovoj šemi strukturi postoji druga slaba tačka. Pošto u njoj ne postoje identifikatori koji ukazuju na veze unazad, gubimo mogućnost da izvršavamo druge interesantne upite nad bazom.

Poglavlje 7

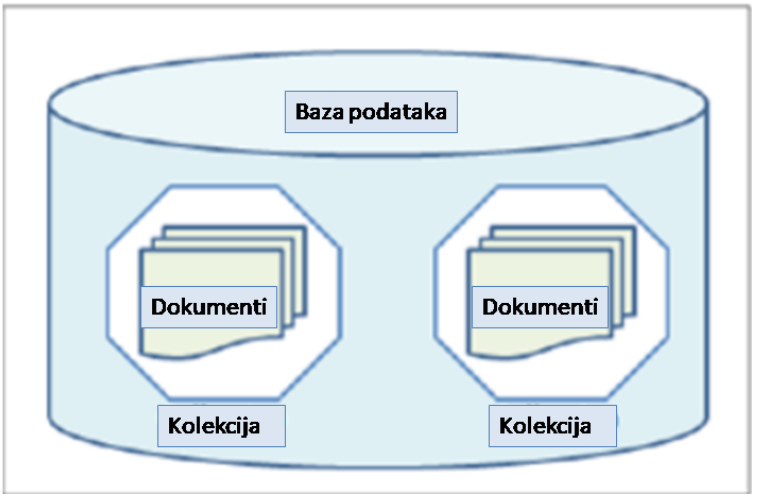
Formati podataka u NoSQL bazama podataka

OSNOVNI ELEMENTI NOSQL BAZA PODATAKA

Osnovni elementi su: baza podataka, kolekcija i dokument

Osnovni elementi u NoSQL bazama podataka su:

1. Baza podataka - čini je fizički kontejner strukture koja se zove kolekcija. Svaka baza ima svoj sopstveni skup file-ova na file sistemu. Jedan DB server obično ima više baza na sebi.
2. Kolekcija - slična RDBMS tabelama. Imena kolekcija su unikatna. Obično su kolekcije koje se sadrže u istoj bazi povezane, ali nije neophodno naznačiti kako su povezane, što predstavlja ključnu razliku u odnosu na RDBMS.
3. Dokumenti – Ovo je najjednostavnija jedinica podataka u NoSQL bazama podataka, obično se sastoji od skupa parova ključ-vrednosti. Za razliku od unosa u RDBMS, NoSQL dokumenti imaju dinamičku šemu. Dokumenti u jednoj kolekciji ne moraju da imaju uvek istu strukturu.



Slika 7.1 Dokument kao ključni element u NoSQL baza podataka [Izvor: Autor]

JSON FORMAT ZA ČUVANJE PODATAKA

U NoSQL bazama podataka, podaci se čuvaju u JSON formatu: to je tekstualni format koji je kompletno nezavisan od programskog jezika

JSON (eng.JavaScript Object Notation) je format za predstavljanje podataka, lak je za čitanje i pisanje, a s druge strane je lak i mašinama za generisanje i parsiranje. Baziran je na podskupu definicije JavaScrip standarada iz Decembra 1999. JSON je tekstualni format koji je kompletno nezavisan od programskog jezika, a koristi konvencije slične C-porodici jezika, uključujući C, C++, C#, Java, JavaScript, Perl, Python, i mnoge druge. Sve ove osobine čine JSON dobrim formatom za predstavljanje podataka.

JSON koristi dve strukture:

1. ***Kolekciju parova ime/vrednost koje se u različitim jezicima mogu zvati objekti, slogovi, strukture, rečnici…***
2. ***Raspoređenu listu vrednosti koja se često naziva niz, vektor, lista, ili sekvenca.***

Ove univerzalne strukture podataka se mogu razlikovati od jezika do jezika. JSON je nastao kao zamena za XML, ali podaci se mogu predstaviti na kraći način i lakše ih je za čitati.
Primer XML-a:

```
<employee>
    <firstName>Peter</firstName>
    <lastName>Jones</lastName>
</employee>
```

Primer JSON-a:

```
{"firstName":"John","lastName":"Doe"}
```

OBJEKAT

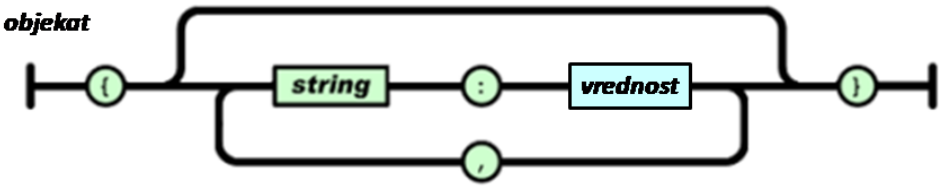
Predstavlja set ime/vrednost parova

Objekat je set ime/vrednost parova.

Definicija objekta počinje otvorenim vitičastim zagradama { i završava se zatvorenim vitičastim zagradama }. Svaki naziv prate dve tačke a ime/vrednost parovi su međusobno odvojeni zarezima.

Primer: predstavljanja radnika u firmi korišćenjem JSON objekata:

```
{"firstName":"John", "lastName":"Doe"}
```



Slika 7.2 Primer objekta [Izvor: Autor]

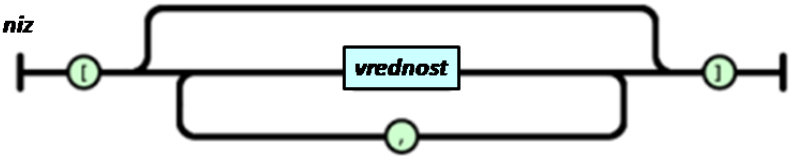
NIZ

Predstavlja kolekciju vrednosti

Definicija niz-a u JSON-u počinje otvorenom uglastom (srednjom) zagradom i završava se zatvorenom uglastom zagradom. Vrednosti u niz-u su odvojeni zarezom.

Primer: predstavljanja zaposlenih u firmi korišćenjem JSON nizova i objekata:

```
{"employees":[
{"firstName":"John", "lastName":"Doe"},
{"firstName":"Anna", "lastName":"Smith"},
{"firstName":"Peter", "lastName":"Jones"} ]}
```



Slika 7.3 JSON nizovi i objekti [Izvor: Autor]

Primer: predstavljanja zaposlenih u firmi korišćenjem XML-a:

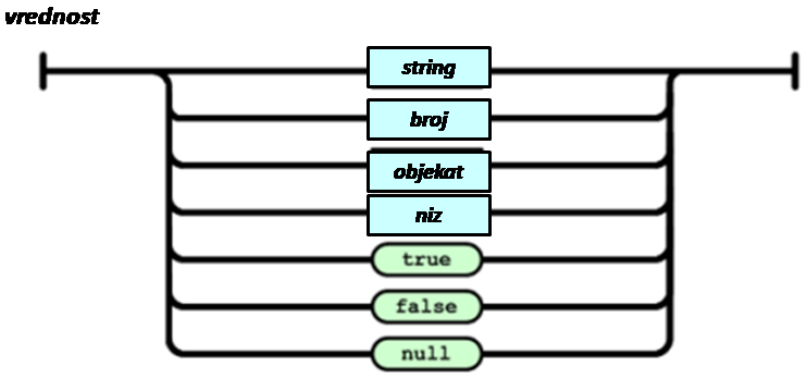
```
<employees>
    <employee>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
    </employee>

    <employee>
        <firstName>Anna</firstName>
        <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName>
        <lastName>Jones</lastName>
    </employee>
</employees>
```

VREDNOSTI

Vrednosti u JSON-u mogu da budu string, broj, true, false ili null, ili objekat ili niz.

Vrednosti u JSON-u mogu da budu *string*, broj, **true**, **false** ili **null**, ili objekat ili niz. Ove strukture mogu da budu ugnježdene.

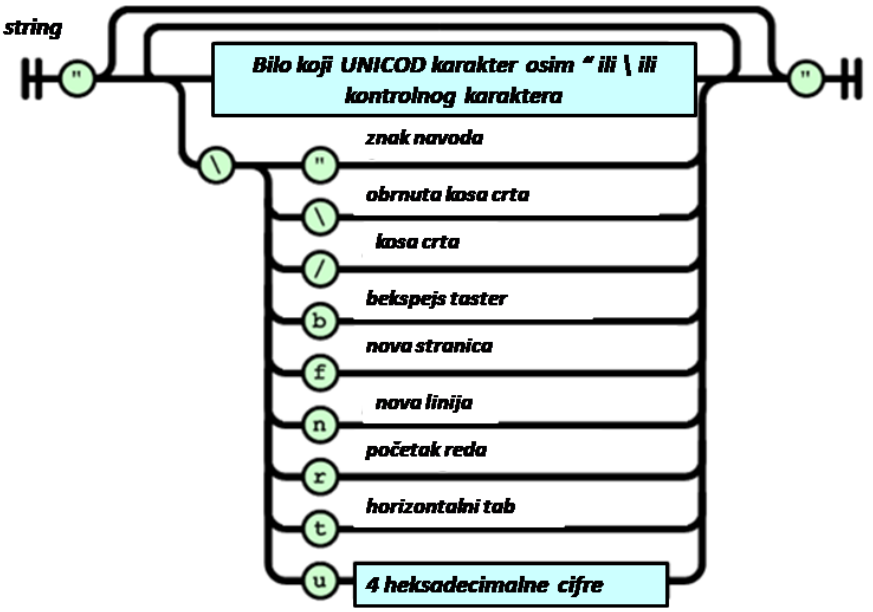


Slika 7.4 Vrednosti U JSON-u [Izvor: Autor]

STRING I BROJEVI

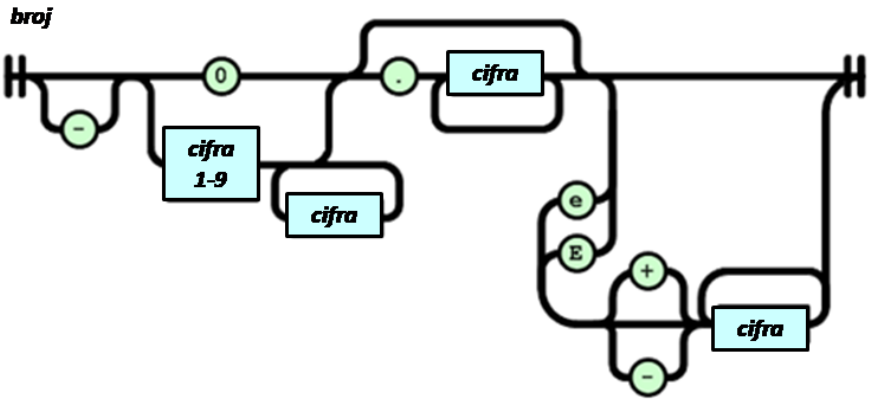
String je sekvenca UNICODE karaktera dok su brojevi slični brojevima u drugim jezicima (Java, C), razlika je u tome što u JSON-u nije moguće koristiti hexadecimalne brojeve

String je sekvenca UNICODE karaktera, u duplim navodnicima. Karakteri se predstavljaju kao string-ovi jednog elementa. String je jako sličan string-ovima u Javi i C-u.



Slika 7.5 String [Izvor: Autor]

Brojevi su slični brojevima u drugim jezicima (Java, C), razlika je u tome što u JSON-u nije moguće koristiti heksadecimalne brojeve.



Slika 7.6 Brojevi [Izvor: Autor]

DOKUMENT

Reprezentacije dokumenata variraju zavisno od programskih jezika, ali većina jezika ima strukture koje se prirodno poklapaju sa strukturom dokumenta

Reprezentacije dokumenata variraju zavisno od programskih jezika, ali većina jezika ima strukture koje se prirodno poklapaju sa strukturom dokumenta, neke od njih su *map*, *hash*, *rečnik*.

Postoji mnogo primera dokumenata koji razume NoSQL:

Primer 1: *{"name" : "Francesco", "age" : 44, "phone":"123-567-890"}*

Većina dokumenata je dosta kompleksnija i sadrže ugnježdene dokumente u sebi. Ovi denormalizovani modeli podataka omogućavaju aplikacijama da manipulišu povezanim podacima u sklopu jedne operacije, kao u primeru 2.

Primer 2: *{"name" : "Francesco", "age" : 44, "contact" : { "phone":"123-567-890"} }*

Svaki par u NoSQL bazi ima ključ koji mora da bude jedinstven u dokumentu. Ključevi u dokumentu su String-ovi. Bilo koji UTF-8 karakter može da bude uključen u ključu sa par izuzetaka:

1. Karakter \0 ne sme biti u ključu.
2. Karakteri . i $ ne smeju biti u ključu.

Kako bismo sačuvali dokument u kolekciju, moramo da izvršimo insert komandu i to možemo uraditi kroz komandu datu na primeru dokumenta:{ *"name" : "francesco", "age" : 44, "phone":"123-567-890" }*

to izgleda:

db.users.insert({"name": "francesco","age": 44, "phone": "123-567- 890"})

Kolekcije mogu da budu kreirane dinamički postavljanjem kroz insert komandu. Ova komanda će sačuvati naš prvi dokument.

U nekim NoSQL bazama (npr. MongoDB-u), postoji podrška za specijalne vrste kolekcija koje se zovu Capped kolekcije koje sadrže dokumente određene veličine što im omogućava veliku brzinu upisa i čitanja dokumenata u redu upisivanja. Capped kolekcije moraju da budu napravljene pre nego da se koriste.

Poglavlje 8

Pokazna vežba

NAČIN ORGANIZACIJE POKAZNIH VEŽBI

Vežba je organizovana kroz uvod deo i deo za samostalni rad studenata

Vežba je organizovana kroz uvod deo i deo za samostalni rad studenata.

1. U uvodnom delu pokaznih vežbi se daje pokazni primer koji studentima treba da pomognu u samostalnom rešavanju zadataka.
2. Zadatke koji su zadati za samostalni rad student samostalno rešava uz pomoć asistenta.

8.1 Pokazni primer - deo 1.

INSTALACIJA I POKRETANJE MONGODB (10 MIN.)

U nastavku će biti sagledan proces instalacije MongoDB –a za različite operativne sisteme.

Jedan od alata koji se može koristiti za rad sa NoSQL bazama podataka je MongoDB.

Instaliranje Mongo DB je obično lakše nego instalacija RDBMS-a jer je samo potrebno unzipovati arhiviranu bazu, i ukoliko je neophodno konfigurisati novu putanju do lokacije za podatke. U nastavku će biti sagledan proces instalacije za različite operativne sisteme.

Koraci instaliranja MongoDB-a na Windows OS-u

1. Download poslednje stabilne verzije MongoDB-a.
2. Pokrenuti instaler, unzipovati skinuti file.
3. MongoDB zahteva mesto gde će čuvati podatke na disku. Bez izmene podešavanja folder će biti na lokaciji c:\data\db. Izvršavanjem naredne napravićemo folder ’data’: **C:\mongodb-win32-x86_64-3.0.3>md data**
4. Potrebno je da MongoDB-i kažemo da se upali i pokaže na određeni folder. To možemo da uradimo tako što ćemo prvo da odemo u folder bin u instalacionom direktorijumu MongoDB-a:

C:\mongodb-win32-x86_64-3.0.3\bin> mongod.exe --dbpath "C:\mongodb-

win32-x86_64-3.0.3\data"

Ova komanda će pokrenuti MongoDB.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

MONGODB VIDEO TUTORIJAL (30 MIN.)

MongoDB u 30 minuta

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

UPOZNAVANJE SA MONGODB-OM (3 MIN.)

MongoDB dolazi sa JavaScript shell-om koji omogućava interakciju sa MongoDB instancom kroz komandnu liniju.

Shell predstavlja osnovni alat za izvršavanje administrativnih funkcija, praćenje pokrenutih instanci ili čak i za obično dodavanje dokumenata.

Kako bismo pokrenuli shell moramo pokrenuti Mongo izvršni file:
$ mongo
MongoDB shell version: 3.0.3
connecting to: test

Shell će automatski da pokuša da se konektuje na pokrenutu instancu MongoDB servera, moramo da se pobrinemo da pokrenemo mongod pre pokretanja shell-a.

Ukoliko ne odaberemo bazu na početku onda će shell da odabere default bazu koja se zove test. Kako bismo sve primere iz vežbi imali na istom mestu preći ćemo na bazu sampledb:
> use sampledb
switched to db sampledb

Naše prethodno iskustvo sa RDBMS-ima nas navodi da mislimo da je potrebno napraviti tabele i baze pre njihovog korišćenja, sa MongoDB-om ovo nije slučaj. Kolekcija, dokumenti i baze se kreiraju tek kada prvi put pokušamo da uradimo čuvanje podataka. Individualne baze, kolekcije i dokumenti mogu da se generišu za vreme rada naše aplikacije tek kada odlučimo šta nam je od podataka potrebno da sačuvamo.

Ukoliko želimo da pogledamo listu baza podataka koje su već napravljene možemo to da uradimo korišćenjem komande show dbs:
>show dbs
local 0.78125GB
test 0.23012GB

Baze podataka koje napravimo se ne prikazuju u listi sve dok ne sačuvamo jedan dokument u njih.

MONGO ALATI (2 MIN.)

MongoDB dolazi sa skupom komandnih alata koji mogu da budu korisni za administratore servera

MongoDB dolazi sa skupom komandnih alata koji mogu da budu korisni za administratore servera.

1. bsondump: prikazuje podatke u BSON formatu (BSON je fileformat koji predstavlja implementaciju JSON-a)
2. mongoimport: Importuje podatke iz JSON, TSV ili CSV i čuva ih u kolekciji.
3. mongoexport: Ispisuje postojeću kolekciju u CSV ili JSON formatu
4. mongodump/mongorestore: Ispisuje MongoDB podatke na disk u BSON formatu
5. mongostat: Prikazuje replike, klustere i skupove replika.

8.2 Pokazni primer - deo 2.

FIND METODA (5 MIN.)

Koristi za izvršavanje upita u MongoDB-u

Find metoda se koristi za izvršavanje upita u MongoDB-u. Ukoliko ne postoje argumenti metoda će vratiti sve dokumente u kolekciji koja je odabrana.

***Primer 1**:*

> db.users.find()

Odgovor baze će biti sličan sledećem:

{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco", "age" : 44, "phone" : "123-456-789" }

Kao što možemo primetiti u naš dokument je dodato id_polje. Ovo polje predstavlja primarni ključ. Svaki mongoDB dokument zahteva jedinstven identifikator, pa ukoliko ga ne dodelimo sami dokumentu MongoDB će ovo uraditi za nas.

Kako bismo mogli da vidimo razlike u našim upitima dodaćemo još jedan dokument u kolekciju:

> db.users.insert({"name": "owen","age": 32, "phone": "555-444-333"})

U ovom trenutku naša kolekcija poseduje dva dokumenta. Ovo možemo da proverimo pokretanjem komande count:
***>** db.users.count()*
2

KLJUČEVI I PRETRAŽIVANJE PO KLJUČEVIMA (5 MIN.)

Naredbe find i filter nam omogućavaju da pretražimo korisnike po vrednosti ključa

Ključevi dokumenata su ograničeni znacima navoda. Ovo nije obavezno ali se koristi kao dobra praksa jer povećava čitljivost samog koda.

Imajući dva dokumenta u kolekciji sada možemo da napišemo upit. Naredbe find i filter nam omogućavaju da pretražimo korisnike po vrednosti ključa.

Primer 2: *ukoliko želimo da nađemo korisnika sa imenom "owen" možemo primeniti naredni upit:*
> db.users.find({"name": "owen"})
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "age" : 32, "phone" : "555-444-333" }

Moguće je postaviti više uslova kao i sa standardnim WHERE i AND konstruktom u SQL-u:

Primer 3:
> db.users.find({"name": "owen", "age": 32})
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "age" : 32, "phone" : "555-444-333" }

ODABIR KOLONA (5 MIN.)

Projekcije se često primenjuju korišćenjem binarnih operatora nule i jedinice;

Prethodne upite koje smo napisali su jednaki SELECT * naredbi u SQL-u. Korišćenjem projekcija možemo da odaberemo podset polja koje želimo da dobijemo iz svakog dokumenta u skupu rezultata upita. Ovo može da bude korisno kada radimo sa velikim dokumentima kako bismo mogli da smanjimo količinu podataka koji moraju da budu poslati i deserijalizovani.

Projekcije se često primenjuju korišćenjem binarnih operatora nule i jedinice; binarni operator 0 znači da vrednost ključa ne sme da bude uključen u rezultat upita, dok vrednost 1 znači da vrednost za taj ključ mora da bude uključena u rezultat upita.

Primer 4: *kako da uključimo name i age u rezultat upita:*
> db.users.find({}, {"name": 1,"age": 1})
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco", "age" : 44 }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "age" : 32 }

Primer 5: *Podešavanjem vrednosti name i age na nulu, biće prikazana samo vrednost za broj telefona:*
> db.users.find({}, {"name": 0,"age": 0})
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "phone" : "123-456-789" }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "phone" : "555-444-333" }

Napomena: Ne smemo koristiti mešavinu operatora za select 0 i 1. Izuzetak od ovog pravila je _id koji može da se razlikuje od ostatka:

{_id: 0, name: 1, age: 1}

KORIŠĆENJE OPSEGA U NOSQL-U (5 MIN.)

Operatorima >, < i = u SQL-u su ekvivalentni operatorima gt, gte, lti lte u MongoDB-u

Vrlo često upiti moraju da imaju funkcionalnost da omoguće opseg podataka koje korisnik želi da vidi. U SQL dijalektima i jezicima za ovo se koriste operatori >, < i =.

Ekvivalentne operacije u MongoDB-u su **$**gt, **$**gte, **$**lt i **$**lte.

Primer 6: *kako možemo da nađemo korisnike čije godine su veće od 40:*
> db.users.find({ age: { $gt: 40 } })
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco", "age" : 44, "phone" : "123-456-789" }

Primer 7:
> db.users.find({ age: { $gte: 32 } })
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco","age" : 44, "phone" : "123-456-789" }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen","age" : 32, "phone" : "555-444-333" }

KORIŠĆENJE LOGIČKIH OPERATORA U UPITIMA (7 MIN.)

Primer operatora or, and i $not.

Skoro je nemoguće zamisliti skripting jezik bez logičkih operatora i MongoDB nije izuzetak. Postoji puno čestih logičkih operatora koji se u MongoDB-u zovu **$**or, **$**and i **$**not.

Ovde nećemo pokazati šta koji operator radi već ćemo pogledati njihovo korišćenje na konkretnim primerima

Primer 8: *Operatora $or:*

db.users.find({ or: [{ "age": { lt: 35 } }, { "name": "john" }] })

U ovom primeru smo selektovali korisnika čije godine su manje od 35 i ima ime john. Ukoliko jedan od uslova bude ispunjen dobićemo kao rezultat dokument:

{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen","age" : 32, "phone" : "555-444-333" }

Primer 9: *Korišćenjem operatora AND nećemo dobiti nijednog korisnika:*

db.users.find({ and: [{ "age": { lt: 35 } }, { "name": "john" }] })

Korišćenja operatora NOT možemo da dobijemo inverzan upita i vratimo dokumente kod kojih se te vrednosti ne poklapaju.

Primer 10: *Upit koji vraća sve korisnike kojima prezime ne počinje slovom f:*
> db.users.find({"name": {$not: /^f/} })
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "age" :32, "phone" : "555-444-333" }

Primer 11: *Korišćenje /expr/ operatora, omogućava izvršavanje operacija sličnih SQL izrazima LIKE.*

Primer za brojeve telefona koji sadrže 444:

> db.users.find({"phone": /444/})

UPDATING DOKUMENATA (3 MIN.)

Ako želimo da menjamo vrednosti polja dokumenata, koristi se $set

Primer 12: *Update-ovanje korisnika koji ima ime owen i postavićemo da mu broj godina bude 39:*
> db.users.find({ age: { $gte: 32 } })
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco","age" : 44, "phone" : "123-456-789" }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen","age" : 32, "phone" : "555-444-333" }

Rezultat ove komande nas informiše da se update poklopio sa jednim dokumentom i da ga je modifikovao.

Možemo da primenimo komandu find kako bismo videli da su izmene uspele:
> *db.users.find()*
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco","age" : 44, "phone" : "123-456-789" }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "age" :39, "phone" : "555-444-333" }

Ukoliko se ne koristi **$**set operator, ne menjaju se vrednosti polja dokumenata, već se čuva ceo dokument, s tim što se zadržava isto _id polje.

OPCIJA UPSERT ZA AŽURIRANJE-KREIRANJE DOKUMENTA (5 MIN.)

Opcija upsert omogućava da update-ujemo dokument ukoliko postoji i kreiramo ga ako ne postoji

Update podržava dodatne opcije koje možemo da koristimo za kompleksniju logiku.

Primer 13: *Želimo da update-ujemo dokument ukoliko postoji i kreiramo ga ako ne postoji. U tom slučaju koristimo upsert i podešavamo njegovu vrednost na true, ovo možemo videti na narednom primeru:*
> db.users.update({user: "frank"}, {age: 40},{ upsert: true})
WriteResult({"nMatched" : 0,"nUpserted" : 1,"nModified" : 0,"_id" : ObjectId("55082f5ea30be312eb167fcb")})

Kao što možemo da vidimo upsert je bio izvršen i dokument sa age ključem je dodat:
> db.users.find()
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco","age" : 44, "phone" : "123-456-789" }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "age" : 39, "phone" : "555-444-333" }
{ "_id" : ObjectId("55082f5ea30be312eb167fcb"), "age" : 40 }

OPCIJA UPSERT ZA UKLJANJANJE DELA DOKUMENTA; OPCIJA PUSH ZA DODAVANJE VREDNOSTI ZA ODREĐENI KLJUČ (5 MIN.)

Uklanjanje dela dokumenta postižemo tako što korišćenjem unset operatora uklonimo određeni ključ iz dokumenta;Operacija push je suprotan operator od operatora unsert

Uklanjanje dela dokumenta možemo videti na narednom primeru.

Primer 13:
db.users.update({name: "owen"}, {$unset : { "age" : 1} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
Možemo izvršiti operaciju find kako bismo videli promene:
> db.users.find()
{ "_id" : ObjectId("5506d5988d7bd8471669e675"), "name" : "francesco","age" : 44, "phone" : "123-456-789" }
{ "_id" : ObjectId("5506eea18d7bd8471669e676"), "name" : "owen", "phone": "555-444-333" }
{ "_id" : ObjectId("55082f5ea30be312eb167fcb"), "age" : 40 }

Suprotni operator od operatora unset, je operator push koji dodaje vrednost za određeni ključ.

Primer 14: *kako da dodamo age ključ za korisnika sa name vrednosti owen:*
> db.users.update({name: "owen"}, {$push : { "age" : 39} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
Možemo postići isti rezultat korišćenjem $set-a na određenom polju koje se ne nalazi u dokumentu.

BRISANJE PODATAKA-REMOVE OPERATOR (5 MIN.)

Korišćenje remove operatora ima iste posledice kao TRUNCATE komanda u SQL-u.

Kao što smo već videli update operator je jako fleksibilan i dozvoljava kraćenje i pomeranje ključeva u kolekciji. Ukoliko je potrebno obrisati skup dokumenata onda moramo da koristimo remove operator.

> db.users.remove()

Najčešće će nam biti potrebno da selektivno uklanjamo dokumente iz naše kolekcije i to ćemo odraditi tako što ćemo da postavimo određena ograničenja našoj remo komandi.

Primer 15: *Evo kako možemo ukloniti korisnike starije od 40 godina:*
> db.users.remove({ "age": { $gt: 40 } })
WriteResult({ "nRemoved" : 1 })

Kao TRUNCATE naredba u SQL, samo uklanja dokumente iz kolekcije. Ukoliko želimo da odbacimo celu kolekciju moramo da koristimo drop metodu, kola uništava celu strukturu i sve povezane index-e.

> db.users.drop()

MONGODB NAREDBE - VIDEO (5 MIN.)

Video tutorijal - razmevanje MongoDB naredbi

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

8.3 Zadaci za samostalni rad

ZADACI ZA SAMOSTALNI RAD - 120 MIN.

Pisanje upita nad postojećom bazom podataka

U lekciji 13 postavljena je arhiva koja sadrži dodatni materijal koji sadrži JSON fajl sa podacima za bazu podataka "Restourants". Preuzmite navedenu arhivu i uvezite podatke u novu MongoDB bazu podataka.

Na osnovu podataka koji se nalaze u bazi podataka, napisati upite na osnovu sledećeg:
1. Napisati MongoDB upit kojim se prikazuju svi dokumenti u kolekciji restorana.
2. Napisati MongoDB upit kojim prikazuju imena svih restorana i kuhinja iz kolekcije.
3. Napisati MongoDB upit kojim se prikazuju sva polja sem id restorana.
4. Napisati MongoDB upit kojim se prikazuju imena svih restorana koji su u mestu (borough) Bronx.
5. Napisati MongoDB upit kojim se prikazuju imena prvih 5 restorana koji su u mestu (borough) Bronx.
6. Napisati MongoDB upit kojim se prikazuju restorani koji se nalaze u ulici koje u nazivu sadrže reči “ Ave” ili “Avenue”

7. Napisati MongoDB upit kojim se prikazuju restorani koji služe krofne
8. Napisati MongoDB upit kojim se prikazuju restorani koji služe picu ili italijansku kuhinju
9. Napisati MongoDB upit kojim se prikazuju restorani koji imaju ukupnu ocenu veću od 150
10. Napisati MongoDB upit kojim se prikazuju restorani koji se nalaze na longitudi manjoj od nule
11. Napisati MongoDB upit kojim se prikazuju restorani koji se nalaze na latitudi većoj od 40.5 I longitudi manjoj od 74.0
12. Napisati MongoDB upit kojim se prikazuju restorani koji imaju makar jednu ocenu C
13. Napisati MongoDB upit kojim se prikazuju restorani koji imaju samo ocene B
14. Napisati MongoDB upit kojim se prikazuju restorani koji imaju ocene A ili B I ukupan skor veći od 100
15. Napisati MongoDB upit kojim se prikazuju restorani na Menhetnu koji imaju skor veći od 200

Potrebno vreme za rešavanje zadataka 120 minuta.

Poglavlje 9

Domaći zadatak

DOMAĆI ZADATAK 13 - VREME IZRADE 150 MIN.

Rešiti zadatak prema postavljenom tekstu

Svaki student radi tri zadatka. Redni brojevi zadatka se računaju po formulama:
1. zadatak = (Broj_indeksa mod 15) + 1
2. zadatak = ((Broj_indeksa + Redni broj zadatka 1) mod 10) +1
3. zadatak = ((Broj_indeksa + Redni broj zadatka 2) mod 20) +1

Na osnovu JSON file-a, priloženog kao dodatni materijal uz 13. lekciju, napisati sledeće upite:
1. Napisati MongoDB upit kojim se prikazuju imena restorana kojima je skor veći od 90.
2. Napisati MongoDB upit kojim se prikazuju imena restorana kojima je skor veći od 80, ali manji od 100.
3. Napisati MongoDB upit kojim se prikazuju restorani koji se nalaze na latitutdi manjoj od -65.7 i imaju skor ocenu veću od 70.
4. Napisati MongoDB upit kojim se prikazuju restorani kojima ime počinje sa ’Mad’.
5. Napisati MongoDB upit kojim se prikazuju restorani koji u svom u imenu imaju slog ’mon’ bilo gde.
6. Napisati MongoDB upit kojim se prikazuju restorani čiji je ZIP kod u opsegu 10000 do 12000
7. Napisati MongoDB upit kojim se prikazuju restorani koji se nalaze u ulici koja u nazivu sadrži reč “Avenue”
8. Napisati MongoDB upit kojim se prikazuju restorani koji imaju samo ocene A ili B
9. Napisati MongoDB upit kojim se prikazuju restorani koji imaju skor veći od 20
10. Napisati MongoDB upit kojim se prikazuju restorani koji imaju skor u opsegu 15 do 25
11. Napisati MongoDB upit kojim se prikazuju restorani koji imaju makar jednu ocenu A i skor manji od 15
12. Napisati MongoDB upit kojim se prikazuju restorani koji služe kinesku hranu
13. Napisati MongoDB upit kojim se prikazuju restorani koji su a Menhetnu, a služe italijansku hranu
14. Napisati MongoDB upit kojim se prikazuju restorani koji služe sladoled
15. Napisati MongoDB upit kojim se prikazuju restorani koji u nazivu sadrže reč “Café”
16. Napisati MongoDB upit kojim se prikazuju restorani čiji je id u opsegu od 4030000 do 4050000
17. Napisati MongoDB upit kojim se prikazuju restorani koji imaju samo ocene “A”

NASTAVKA FORMULACIJE DOMAĆEG ZADATKA 13

Nastavak teksta zadataka za rešavanje

18. Napisati MongoDB upit kojim se prikazuju restorani čiji skor na ocenama veći od 50
19. Napisati MongoDB upit kojim se prikazuju restorani čiji je skor na ocenama u opsegu 50 do 150
20. Napisati MongoDB upit kojim se prikazuju restorani čiji je skor manji od 50
21. Napisati MongoDB upit kojim se prikazuju restorani koji u nazivu sadrže reč “restaurant” (case insensitive)
22. Napisati MongoDB upit kojim se prikazuju restorani koji u nazivu sadrže reč “Club” ili “Garden”
23. Napisati MongoDB upit kojim se prikazuju restorani koji se ne nalaze na Menhetnu ili u Bronksu
24. Napisati MongoDB upit kojim se prikazuju restorani u nazivu ulice sadrže reč “Street”
25. Napisati MongoDB upit kojim se prikazuju restorani koji služe košer hranu
26. Napisati MongoDB upit kojim se prikazuju restorani koji se nalaze u ulici koja u naziv sadrži broj
27. Napisati MongoDB upit kojim se prikazuju restorani koji se nalazi u zgradi čiji broj sadrži “-“
28. Napisati MongoDB upit kojim se prikazuju restorani koji nalazi u Queens-u, a čiji je ZIP kod u opsegu od 11000 do 11500
29. Napisati MongoDB upit kojim se prikazuju restorani koji imaju makar jednu ocenu čiji je skor manji od 10
30. Napisati MongoDB upit kojim se prikazuju 10 restorana prema ukupnom skoru koji se nalaze u Brooklyn-u ili Queens-u

Napomena: Rešeni zadatak šaljete kao tekstualni fajl. Prilikom slanja domaćih zadatka, neophodno je da ispunite sledeće:

- Subject mail-a mora biti IT250-DZbr (u slučaju kada šaljete domaći za ovu nedelju to je IT250-DZ13)
- U prilogu mail-a treba da se nalazi arhiviran projekat koji se ocenjuje imenovan na sledeći način: IT250-DZbr-BrojIndeksa-Ime Prezime. Na primer, IT250-DZ13-1234-VeljkoGrkovic
- Telo mail-a treba da ima pozdravnu poruku
- Arhivu sa zadatkom poslati na adresu predmetnog asistenta: milica.vlajkovic@metropolitan.ac.rs (studenti u Beogradu i online studenti) ili tamara.vukadinovic@metropolitan.ac.rs (studenti u Nišu).
- Svi poslati mail-ovi koji ne ispunjavaju navedene uslove NEĆE biti pregledani. Za sva pitanja ili nedoumice u vezi zadatka, možete se obratiti asistentu

ZAKLJUČAK

Šta smo naučili u ovoj lekciji?

U interakciji između XML-a i baza podataka, XML se može koristiti na dva načina:

- **u bazi podataka se može pamtiti sam XML dokument (native baze podataka)**
- **XML se smešta u relacionu bazu podataka kao skup tabela eksplicitno dizajniranih za pamćenje XML dokumenata (XML enabled baze podataka)**

U slučaju XML enabled baze podataka, XML se koristi za razmenu podataka između baze i neke aplikacije ili druge baze. U samoj bazi podataka se “ne vidi” XML dokument. Kada se koristi XML-enabled baza podataka, neophodno je šemu baze podataka mapirati u XML šemu (ili obrnuto). Ovde je bilo reči o objektno relacionom mapiranju.

Potreba za XML native bazama podataka postoji kada treba upravljati : dokumentima, polu-struktuiranim podacima, transakcijama čije izvršenje traje dugo ili kada se radi o arhiviranju dokumenata. Osnovna jedinica za pamćenje podataka u native XML bazama jeste dokument koji može predstavljati knjigu pisanu u XML-u, medicinski podaci zapamćeni kao XML dokumenti itd.

NoSQL baze podataka nisu bazirane na šemi; u bazi se umesto relacija čuvaju agregacije informacija; pamte se na jeftinijem hardveru i visoko su distribuirane. NoSQL ne podržavaju ACID već BASE garancije konzistentnosti što znači da neki delovi sistema mogu da sadrže samo podatke za čitanja koji bivaju povremeno ažurirani. Prednost BASE pristupa je što se transakcije sporije trajno pamte, što znači i brže čitanje.

Osnovni elementi NoSQL baza su:

- **Kolekcija** – Skup dokumenata. Kolekcija je slična RDBMS tabelama. Imena kolekcija su unikatna. Obično su kolekcije koje se sadrže u istoj bazi povezane ali nije neophodno naznačiti kako su povezane, što predstavlja ključnu razliku u odnosu na RDBMS.
- **Dokumenti** – Ovo je najjednostavnija jedinica podataka u NoSQL bazama podataka, obično se sastoji od skupa parova ključ-vrednosti. Za razliku od unosa u RDBMS, NoSQL dokumenti imaju dinamičku šemu. Dokumenti u jednoj kolekciji ne moraju da imaju uvek istu strukturu.

LITERATURA

Za pisanje ove lekcije korišćena je sledeća literatura

1. Jim Webber, Ian Robinson, Emil Eifrem, Graph Databases, O’Reilly Media, 2013
2. https://www.tutorialspoint.com/mongodb/