Usporedba sustava PostgreSQL i MongoDB

Gazda, Sara

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Sara Gazda

USPOREDBA SUSTAVA PostgreSQL i MongoDB

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Sara Gazda

Matični broj: 0016156797

Studij: Informacijski i poslovni sustavi

USPOREBA SUSTAVA PostgreSQL i MongoDB

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Kornelije Rabuzin

Varaždin, lipanj 2024.

Sara Gazda

Izjava o izvornosti

Izjavljujem da je moj završni izvorni rezultat mojeg rada te da se u izradi istoga nisam koristila drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Rad prati usporedbu SQL i NoSQL sustava za upravljanje bazama podataka. U teorijskom dijelu rada opisani su SQL i NoSQL sustavi, njihove razlike, glavne značajke i karakteristike te prednosti i nedostatci oba sustava. Detaljnom analizom prikazane su mogućnosti i optimalnosti korištenja određenog sustava sukladno prirodi podataka i svrsi obrade.

U praktičnom dijelu rada izrađena je usporedba između PostgreSQL, popularne relacijske baze podataka i MongoDB, često korištene NoSQL baze podataka. Korištenjem PostgreSQL i MongoDB sustava na primjeru obrade i rada s podacima prikazana je usporedba sustava za upravljanje bazama podataka te situacije u kojima je bolje odabrati jedan od sustava.

U zaključku završnog rada pružen je sveobuhvatan pregled prednosti i nedostataka korištenja SQL i NoSQL sustava, smjernice prilikom odabira odgovarajućeg sustava te konkretna usporedba dva često korištena sustava – PostgreSQL i MongoDB.

Ključne riječi: SQL baze podataka, NoSQL baze podataka, PosgtreSQL, MongoDB, sustav za obradu podataka, usporedba, prednosti, nedostatci

Sadržaj

1. Uvod

U današnjem digitalnom dobu, količina podataka koju generiraju i obrađuju raznovrsne aplikacije neprestano raste. Iako je potreba za učinkovitim i pouzdanim sustavima za pohranu podataka od početaka razvoja tehnologije izuzetno važna, sa znatnim porastom količine podataka potreba za sustavima za upravljanje bazama podataka visokih performansi dodatno dobiva na težini.

Relacijske baze podataka koje upotrebljavaju SQL (engl. *Structured Query Language*) postale su standardni alat za pohranu i obradu podataka. Takve baze podataka temelje se upravo na relacijskom modelu podataka te se koriste za pohranu, organiziranje i manipuliranje strukturiranim podacima. S pojavom novih aplikacija i općenitog tehnološkog napretka, pojavljuju se podaci koje nije moguće prikladno i smisleno pohraniti u obliku strukturiranih podataka i rasporediti u tablice. Osim toga, enormnim porastom količine podataka raste i potreba za brzim i fleksibilnim pristupom, a upravo se ovdje pojavljuju NoSQL (engl. *Not Only SQL, No SQL*) baze podataka, popularno nazivane i nerelacijske baze podataka. NoSQL baze podataka predstavljaju alternativu SQL bazama podataka i ideja je da pružaju mogućnost pohrane i obrade nestrukturiranih i polustrukturiranih podataka. S obzirom na fleksibilnost NoSQL baza podataka, popularnost i korištenje istih je u porastu.

Ovaj rad istražuje razlike između tradicionalnih relacijskih baza podataka i NoSQL baza podataka, prikazujući usporedbu između PostgreSQL-a, kao predstavnika SQL baza podataka, i MongoDB-a, kao predstavnika NoSQL baza podataka. Na kraju, rad će donijeti zaključke o najboljim slučajevima upotrebe SQL-a odnosno NoSQL-a te prikazati prednosti i nedostatke obje vrste baza podataka.

2. Metode i tehnike rada

Za izradu ovog završnog rada korištena je relevantna literatura, udžbenici i istraživani su povezani članci navedeni u popisu literature na kraju rada.

Praktični dio rada izrađen je u nekoliko alata s ciljem prikazivanja i konkretiziranja teorije. U svrhu pojašnjenja relacijskih baza podataka i njihovih karakteristika, u programu draw.io izrađen je primjer ERA modela. PostgreSQl i MongoDB sustavi korišteni su za prikaz relacijskih i nerelacijskih baza podataka te je na istim primjerima napravljena konkretna usporedba dvije vrste baza podataka.

3. Relacijske (SQL) baze podataka

Maleković i Rabuzin (2016) pojašnjavaju bazu podataka kao model za aplikacijsku domenu koja predstavlja dio realnog svijeta koji je od interesa za razmatranje. Uz to, baza podataka reprezentira dijelove realnog svijeta, a sastoji se od kolekcije podataka, skupa ograničenja te definiranih operacija.

Model podataka (MP) za aplikacijsku domenu (AD) nastaje modeliranjem aplikacijske domene, a isti je osnova za upravljanje bazama podataka (BP) pomoću sustava za upravljanje bazama podataka (SUBP). Pravilan model podataka sastoji se od strukturalne, integritetne i operativne komponente, što je moguće povezati s prethodno navedenom definicijom baza podataka gdje se spominju upravo kolekcije podataka, ograničenja i operacije. Strukturalna komponenta (S) modela podataka definira oblik u kojemu je prikazana kolekcija podataka, integritetna komponenta (UI) sastoji se od skupa ograničenja nad kolekcijom podataka unutar modela podataka dok operativna komponenta (O) predstavlja definirane operacije nad strukturama podataka. Iz navedenog, moguće je izvesti skraćeni zapis definicije modela podataka prema Malekoviću i Rabuzinu (2016); MP = (S, UI, O).

Važnost definiranja modela podataka leži i u tome što se prema modelima podataka nazivaju baze podataka i sustavi za upravljanje istima. Postoji više različitih modela podataka koji na specifičan način određuju strukturalne, integritetne i operativne komponente. Proširenjem relacijskog modela podataka moguće je definirati i modele poput temporalnih, deduktivnih i objektno-relacijskih (Maleković i Rabuzin, 2016).

Fokusiramo li se na relacijski model podataka, na temelju njega definirane su relacijske baze podataka te relacijski sustavi za upravljanje bazama podataka. U skraćenom zapisu definicije relacijskog modela podataka, RMP = (S, UI, O), strukturalna komponenta je skup relacija odnosno tablica, integritetna komponenta su ograničenja nad stanjima relacija, a operativna komponenta su definirani relacijski operatori. S obzirom na strukturalnu komponentu relacijskog modela podataka, jasno je da će podaci u RMP biti prikazani u tabličnom obliku ili, uzimajući u obzir razlike u terminologiji, u obliku relacija. U nastavku je prikazana baza podataka za primjer s ciljem pobližeg objašnjenja terminologije.

ID#	Prezime	Struka
1	Gazda	S1
2	Mako	S1
3	Seki	S2

Tablica 1. Tablica liječnika (autorski rad)

MBO#	Prezime	Broj_kartona
01	Koti	987
02	Roth	654
03	Luri	321

Tablica 2. Tablica pacijenata (autorski rad)

Prikazana baza podataka sastavljena je od dvije relacije naziva *liječnik* i *pacijent* u koje su uneseni podaci. U ovom se slučaju aplikacijska domena (AD) sastoji od šest objekata, konkretno tri liječnika i tri pacijenta, a za obje tablice definirana su obilježja koja nas zanimaju o liječnicima odnosno pacijentima.

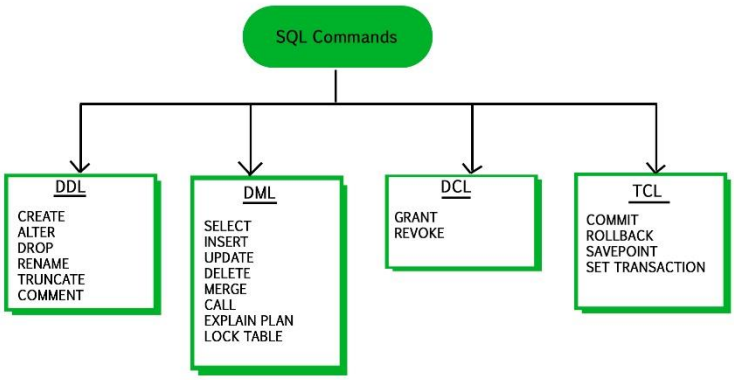
U zaglavlju tablica prikazani su skupovi atributa sh(liječnik) = {ID#, Prezime, Struka} i sh(pacijent) = {MBO#, Prezime, Broj-kartona} koji predstavljaju relacijsku shemu. Nadalje, relacije se sastoje od stupaca i redaka pri čemu se stupci nazivaju atributi, a druga terminologija za red jest slog. Analiziramo li prvi slog, ili prvi objekt od interesa, tablice liječnik možemo izvesti sljedeću činjenicu: prvi objekt na atributu ID poprima vrijednost 1, na atributu Prezime poprima vrijednost Gazda dok na atributu Struka poprima vrijednost S1. Činjenice izvedene iz relacija predstavljaju istinite propozicije, a moguće ih je izreći i bliže svakodnevnom jeziku. Na primjer, prvi slog relacije pacijent je trojka (01, Koti, 987) i u vezi s atributima predstavlja istinitu propoziciju koja može glasiti ovako: pacijent prezimena Koti s matičnim brojem osiguranika 01 upisan je pod brojem kartona 987.

3.1. SQL

SQL (engl. *Structured Query Language*) je standardizirani programski jezik kojim je moguće upravljati i manipulirati relacijskim bazama podataka. Pogodan je za kreiranje, ažuriranje, brisanje i provođenje drugih operacija nad relacijama, odnosno tablicama te se stoga koristi za rad s relacijskim bazama podataka.

Povijest SQL jezika seže u 1970-e godine kada je Edgar F. Codd predstavio koncept relacijskih baza podataka u radu „A Relational Model of Data for Large Shared Data Banks“. Objavom rada predstavljena je ideja da se povezanost i odnosi među podacima ne trebaju zasnivati na povezivanju ili ugnježđivanju podataka već na njihovim vrijednostima. Vođeni inovativnim radom, Chamberlin i Boyce razvili su SQL jezik s ciljem prevođenja složenih upita u učinkovita rješenja i pristupe problemu. Unatoč prvotnoj ideji i viziji, danas je jasno da strukturirani upitni jezik nije namijenjen isključivo postavljanju upita već sveobuhvatnom radu s bazama podataka (International Business Machines [IBM], bez dat.).

Jezik SQL moguće je brzo i lako razumjeti s obzirom da je usko povezan s engleskim jezikom i jednostavno se „prevodi“. SQL se sastoji od velikog broja naredbi koje mogu koristiti nešto različitu sintaksu ovisno o sustavu za upravljanje bazama podataka, ali u suštini imaju istu namjenu. Osnovne grupe naredbi prema Groffu i Weinberg su DDL, DML, ACL, TCL i programski SQL.



Slika 1. Grupe SQL naredbi (GeeksforGeeks, 22. svibnja 2024.)

DDL odnosno *Data Definition Language* služi definiranju strukture baze podataka i njime se kreira i upravlja objektima baze podataka. Neke od osnovnih naredbi ove grupe su CREATE, ALTER i DROP koje služe kreiranju, promjenama nad objektima i brisanju istih unutar baze podataka, pa tako i same baze podataka (DATABASE). Objekti u pitanju mogu biti tablice (TABLE), pogledi (VIEW), indeksi (INDEX) i slično.

Data Manipulation Language, DML grupa naredbi povezuje upravo naredbe koje se koriste u manipulaciji podataka unutar baza podataka. Naredbe unutar ove grupe omogućuju unos podataka u bazu, prikazivanje, brisanje i ažuriranje istih te predstavljaju moguće manipulacije nad podacima. Naredbom INSERT moguće je unositi nove redove, odnosno slogove, podataka u bazu; DELETE omogućuje uklanjanje podataka; SELECT služi prikazu podataka unutar objekata dok se naredba UPDATE koristi za modifikaciju i uređivanje postojećih podataka u bazi podataka.

ACL predstavlja *Access Control Language* i od iznimne je važnosti za sigurnost baza i podataka unutar njih. Korištenjem naredbi unutar ACL grupe moguće je upravljati pristupom podacima i kontrolirati sigurnost. Prema tome, omogućeno je dodjeljivanje prava za manipulaciju podacima što se izvršava naredbom GRANT te oduzimanje prava za rad s podacima od korisnika što je moguće naredbom REVOKE. Poradi očuvanja baze podataka i sigurnosti podataka važno je osvijestiti prava koja su dodijeljena pojedinom korisniku i po potrebi istima upravljati.

Transaction Control Language upravlja transakcijama koje se u suštini sastoje od više SQL naredbi. Svaka SQL naredba predstavlja implicitnu transakciju, a kada je potrebno kreirati eksplicitne transakcije koriste se TCL naredbe. Naredbom BEGIN označava se započinjanje eksplicitne transakcije dok COMMIT potvrđuje transakciju i trajno čuva rezultate naredbi unutar eksplicitne transakcije. Često korištena naredba prilikom uporabe transakcija je ROLLBACK koja prekida trenutnu transakciju te poništava promjene koje su napravljene tokom izvođenja transakcije.

U nastavku je prikazana razrada SQL-a s primjerima sintakse za naredbe temeljem „Bakus-Naur-Form“ [BNF] notacije. BNF notacija je formalna notacija kojom se definira gramatika određenog jezika, a koristi skup simbola kojima će biti prikazane SQL sintakse (Rabuzin, 2011.)

3.1.1. Kreiranje korisnika

U današnje vrijeme velike dostupnosti podataka putem interneta, osiguranje sigurnosti baze podataka i zaštita samih podataka sve više postaje imperativ. Za osiguranje podataka unutar baze nema jedinstvenog rješenja već je potrebno doprinositi manjim akcijama koje povećavaju ukupnu razinu sigurnosti. Ključno je zaštiti podatke od neautoriziranog pristupa gdje na važnosti dobivaju korisnici, objekti u bazi i ovlasti koje definiraju što korisnik smije raditi s pojedinim objektima (Rabuzin, 2011).

Identifikacija odnosno autentikacija korisnika prvi je korak u zaštiti podataka unutar baza, a njezin cilj je odrediti je li osoba ona za koju se predstavlja. Autentikacija se temelji na potvrđivanju identiteta korisnika što je moguće provesti korisničkim imenom i lozinkom, tokenom ili biometrijskim podacima. Autentikaciju je moguće koristiti na razini SUBP gdje se kreiranje korisnika radi naredbom CREATE USER, a korisnik se može obrisati naredbom DROP USER.

S obzirom da je naredba CREATE USER u suštini samo drugo ime za naredbu CREATE ROLE, sintaksa kreiranja korisnika i opcije koje se korisniku mogu dodijeliti jednake su kao kod sintakse kreiranja uloge koja je navedena u idućem poglavlju četvrte razine. Ipak, razlika između naredbe CREATE USER i CREATE ROLE je u opciji LOGIN. Naime, opcija LOGIN, koju podrazumijeva naredba CREATE USER, omogućuje korisniku spajanje na bazu dok opcija NOLOGIN, koju podrazumijeva naredba CREATE ROLE, ne omogućuje spajanje na bazu već je korisna za dodjelu ovlasti. Prema tome, u situacijama kada postoji više korisnika u sustavu i potrebno im je dodijeliti jednake ovlasti, praksa je za to koristiti uloge.

Autorizacija predstavlja upravo dodjeljivanje ovlasti identificiranim korisnicima, a same ovlasti, poput SELECT, INSERT ili UPDATE; predstavljaju sposobnost izvršavanja određene naredbe nad određenim objektima baze podataka. Za dodjeljivanje ovlasti koristi se naredba GRANT, dok je ovlasti moguće oduzeti naredbom REVOKE; sintakse su prikazane u nastavku (Rabuzin, 2014; Maleković i Rabuzin, 2016).

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES |
TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES |
TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM role_specification [, ...]
    [ GRANTED BY role_specification ]
    [ CASCADE | RESTRICT ]
```

3.1.1.1. Kreiranje uloga

Kako je ranije opisano, s obzirom da je uloga entitet koji je vlasnik objekata u bazi i ima određene ovlasti, umjesto pojedinačnog dodjeljivanja ovlasti velikom broju korisnika, moguće je dodijeliti korisnike kreiranoj ulozi. Prije dodjeljivanja korisnika ulozi, potrebno je kreirati ulogu naredbom CREATE ROLE čija se sintaksa nalazi u nastavku te korištenjem naredbi GRANT i REVOKE dodijeliti ili oduzeti ovlasti ulozi.

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid

DROP ROLE [ IF EXISTS ] name [, ...]
```

Postavljanje u ulogu moguće je naredbom `SET ROLE <naziv_uloge>`, a nakon kreiranja uloge, sljedećom naredbom je moguće dodijeliti korisnike kreiranoj ulozi:

```
GRANT role_name [,…] TO role_name [,..][WITH ADMIN OPTION]
```

3.1.2. Kreiranje baze podataka

Kreiranje baze podataka može se razlikovati od sustava do sustava, a u PostgreSQL sustavu kreira se naredbom CREATE DATABASE pri čemu mogućnost kreiranja baze podataka ima samo uloga SUPERUSER ili s privilegijom CREATEDB. Sintaksa kreiranja baze podataka dana je u nastavku, a odabir baze za korištenje moguć je naredbom `\c <naziv_baze_podataka>`. Nadalje, brisanje baze podataka moguće je naredbom `DROP DATABASE [IF EXISTS] <naziv_baze_podataka>` gdje IF EXISTS sprječava vraćanje greške ukoliko baza navedenog imena ne postoji.

```
CREATE DATABASE name
    [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ STRATEGY [=] strategy ]
           [ LOCALE [=] locale ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ ICU_LOCALE [=] icu_locale ]
           [ ICU_RULES [=] icu_rules ]
           [ LOCALE_PROVIDER [=] locale_provider ]
           [ COLLATION_VERSION = collation_version ]
           [ TABLESPACE [=] tablespace_name ]
           [ ALLOW_CONNECTIONS [=] allowconn ]
           [ CONNECTION LIMIT [=] connlimit ]
           [ IS_TEMPLATE [=] istemplate ]
           [ OID [=] oid ]
```

3.1.3. Kreiranje tablica

Unutar baze podataka, najvažniji objekt same baze su tablice u koje su pohranjeni podaci. Korištenjem naredbe CREATE TABLE kreira se tablica u bazi podataka, a vlasnik objekta (tablice) je korisnik koji je izvršio samu naredbu. Iako je korištenjem naredbe ALTER TABLE moguće naknadno mijenjati strukturu tablice dodavanjem i brisanjem stupaca ili ograničenja, poželjno je obratiti pozornost na redoslijed kreiranja tablica pri čemu se prvo kreiraju tablice koje ne sadrže vanjske ključeve. Sama sintaksa kreiranja tablice izuzetno je opširna te je za potrebe ovog rada reducirana na oblik s najčešće korištenim naredbama.

```
CREATE TABLE table_name ([
{column_name data_type [DEFAULT default_expr] [column_constraint […]]
|table_constraint} [, ...]
```

```
])
```

Brisanje tablice moguće je sljedećom naredbom:

```
DROP TABLE [ IF EXISTS ] name [,...] [ CASCADE | RESTRICT ]
```

3.1.4. Tipovi podataka

Tipovi podataka određuju koje podatke je moguće spremiti u pojedini stupac, odnosno prilikom kreiranja tablice za svaki stupac određuje se kojeg je tipa. PostgreSQL podržava puno tipova podataka, a među najkorištenijima su numerički tipovi (INTEGER, DECIMAL); znakovni tipovi (VARCHAR(N), TEXT); vremenski tipovi (TIMESTAMP) te ostali tipovi podataka u koje se svrstavaju npr. monetarni (MONEY) i logički (BOOLEAN).

U radu s podacima u SUBP moguća je pojava potrebe konverzije podataka iz jednog u drugi tip. Eksplicitne konverzije podataka moguće su naredbom `CAST(<source expression> AS <result type>)`.

3.1.5. Integritet baze podataka

U idealnoj situaciji, podaci u bazi podataka su konzistentni i ispravni. S obzirom da nije moguće u svakoj situaciji jamčiti ispravnost podataka, očuvanje konzistentnosti tj. zadovoljavanje ograničenja dobiva na važnosti. Definiranje tipa pojedinog stupca jedna je od dimenzija očuvanja konzistentnosti te je osiguravanje konzistentnosti moguće definiranjem ograničenja nad stupcem ili tablicom.

Definiranje ograničenja na razini stupca radi se nakon definiranja tipa podataka, a ograničenja na razini stupca su PRIMARY KEY, NULL (NOT NULL), UNIQUE, CHECK (<uvjet>), te vanjski ključ u obliku REFERENCES <tablica>[stupac]. Ograničenja na razini tablice pišu se nakon definiranja stupaca tablice, a slična su ograničenjima na razini stupca uz neke izmjene sintakse; UNIQUE (stupac1, stupac2…), PRIMARY KEY (stupac1, stupac2…), CHECK (<uvjet>), FOREING KEY (stupac1, stupac2…) REFERENCES <tablica>[stupac]. Prilikom odabira načina definiranja ograničenja, treba obratiti pozornost na sintaksu ograničenja. Na primjer, ukoliko je vanjski ključ jednostavan (sastoji se od jednog stupca) moguće ga je definirati kao ograničenje nad stupcem, ali ako se radi o složenom vanjskom ključu koji je sastavljen od više stupaca potrebno ga je definirati kao tablično ograničenje. Zaključno, ograničenja baza podataka definiraju se s ciljem dizajniranja i implementiranja baza podatka koje osiguravaju točnost, dosljednost i sigurnost podataka te je važno obratiti pozornost na njih.

3.1.6. Primarni i vanjski ključevi

Primarni ključ je jedinstveni identifikator svakog reda, odnosno zapisa u bazi podataka. Prema tome, primarni ključevi služe jednoznačnoj identifikaciji instance tipa entiteta. Prilikom definiranja atributa tablice, potrebno je odrediti jedan primarni ključ koji može biti jednostavan ili složen – sastavljen od više atributa. Složeni primarni ključ karakterističan je za slabe entitete ternarnih veza, a može se koristiti i u situacijama kada jednostavan primaran ključ ne definira svaku instancu tipa entiteta u potpunosti jednoznačno. Glavne karakteristike primarnih ključeva, bili oni jednostavni ili složeni, jesu jedinstvenost i ne poprimanje NULL, a oba ograničenja automatski su definirana korištenjem ograničenja PRIMARY KEY. Glede jedinstvenosti, nužno je da svaki redak tablice, koji ujedno predstavlja i instancu tipa entiteta, ima jedinstvenu vrijednost u stupcu primarnog ključa. Osim toga, nijedan zapis unutar tablice ne smije u stupcu primarnog ključa poprimiti NULL jer u tom slučaju primarni ključ instance tipa entiteta uopće nije definiran i nije ga moguće jednoznačno identificirati. Osim ispunjavanja navedenih ograničenja, poželjno je obratiti pozornost i na nepromjenjivost primarnog ključa što u suštini znači da se vrijednosti primarnog ključa ne bi trebale naknadno mijenjati jer se koriste upravo za identifikaciju zapisa.

Nadalje, za potrebe povezivanja tablica i ispunjavanja poslovnih pravila baze podataka potrebno je kreirati i vanjske ključeve. Vanjski ključevi također mogu biti sastavljeni od jednog ili više stupca te njihova vrijednost odgovara vrijednosti primarnog ključa tablice koju referenciraju. Prilikom definiranja vanjskih ključeva, nužno je obratiti pozornost na povezivanje tablica preko primarnih i vanjskih ključeva za potrebe kojeg tip podatka vanjskog ključa mora odgovarati tipu podatka primarnog ključa na kojeg se referencira. Moguće je da su vanjski ključevi sastavni dio primarnih ključeva, no to ne mora biti slučaj; vanjski ključevi mogu biti stupci odvojeni od primarnog ključa u jednoj relaciji.

Kao što je rečeno, svaka relacija nužno mora imati definiran primarni ključ, no to nije slučaj i za vanjski ključ. Ovisno o tipu veze među tablicama koje definiraju poslovna pravila određuju se tablice koji sadrže vanjski ključ. Primjeri implementiranja primarnih i vanjskih ključeva bit će prikazani u sljedećem poglavlju prilikom definiranja unarnih, binarnih i ternarnih veza.

3.1.6.1. Entitetski integritet

Entitetski integritet je ograničenje nad bazom podataka usko vezano uz primarni ključ relacije. Entitetski integritet definira da primarni ključ, jedinstven ili složen, ne smije sadržavati NULL. Radi li se o jednostavnom primarnom ključu, to znači da stupac primarnog ključa za nijednu instancu tipa entiteta ne smije poprimiti NULL. Ako je u pitanju složeni primarni ključ,

niti jedna od sastavnica, odnosno stupaca/atributa, kojima je definiran primarni ključ ne smije poprimiti NULL (Maleković i Rabuzin, bez dat.).

3.1.6.2. Referencijalni integritet

Referencijalni integritet je ograničenje nad bazom podataka usko vezano uz vanjski ključ relacije koji se referencira na drugu relaciju, odnosno na primarni ključ druge relacije. Da bi ograničenje referencijalnog integriteta bilo ispunjeno, vrijednost koja se upisuje u stupac vanjskog ključa relacije mora odgovarati već postojećoj vrijednosti primarnog ključa relacije koju vanjski ključ referencira. Radi ispunjenja referencijalnog integriteta dobra je praksa obratiti pažnju na redoslijed kreiranja tablica i unošenja podataka.

3.1.7. Unos, ažuriranje i brisanje podataka

Nakon kreiranja tablica naredbom CREATE TABLE, potrebno je u njih unesti podatke što je moguće naredbom INSERT čija je sintaksa prikazana u nastavku. Prilikom korištenja naredbe potrebno je definirati u koju se tablicu unose podaci te eventualno navesti stupce u koje se podaci upisuju, a važno je obratiti pozornost da upisane vrijednosti odgovaraju tipu stupca. Ukoliko se ne navedu nazivi stupaca, podrazumijeva se da je redoslijed unesenih podataka i redoslijed samog spremanja u stupce.

Unos podataka u tablice moguć je red po red ili upisivanjem više redova odjednom, a korištenjem naredbe SELECT INTO moguće je kreirati tablicu koja sadrži podatke iz neke druge tablice.

```
INSERT INTO table_name [ AS alias ] [ ( column_name [,...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [,...] )
     [,...] | query }
    [ RETURNING * | output_expression [ [ AS ] output_name ] [,...] ]
```

Ukoliko se prilikom unosa podataka dogodi greška ili se podaci naknadno promijene, moguće je ažurirati unesene podatke korištenjem naredbe UPDATE. Unos novih, ažuriranih, vrijednosti moguć je u jednom redu, više redova ili u svim redovima tablice; a sintaksa naredbe za ažuriranje podataka prikazana je u nastavku.

```
UPDATE [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    SET { column_name = { expression | DEFAULT } |
          ( column_name [,...] ) = [ ROW ] ( { expression | DEFAULT }
          [,...] ) |
          ( column_name [,...] ) = ( sub-SELECT )
        } [,...]
    [ FROM from_item [,...] ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [,...] ]
```

Do sada je prikazana naredba DROP koja se koristi prilikom brisanja objekata baze podataka, a naredbom DELETE moguće je brisati redove odnosno podatke iz tablica. Klauzulom WHERE, koja je dio sintakse naredbe DELETE, određuje se prema kojem uvjetu će se brisati redovi pa se tako iz tablice može obrisati nula, jedan ili više redova.

```
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    [ USING from_item [, ...] ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [,...] ]
```

3.1.8. Agregirajuće funkcije

Agregirajuće funkcije, odnosno skupovne funkcije (engl. *set function*) služe za vraćanje vrijednosti na temelju skupa, a često se koriste uz klauzulu GROUP BY koja služi dijeljenju podataka u logičke grupe prema stupcu ili izrazu. Korištenjem skupovnih funkcija moguće je vratiti jednu:

AVG(); prosječnu vrijednost stupca,

COUNT(); broj redova,

MAX(); maksimalnu vrijednost stupca,

MIN(); minimalnu vrijednost stupca,

SUM(); sumira vrijednosti u stupcu (Rabuzin, 2011).

3.1.9. Upiti

Upitima je moguće dohvatiti određene redove iz jedne ili više tablica, a prema SQL standardu predstavljaju operacije kojim se referenciramo na tablice pri čemu nakon izvršavanja ponovno nastaje tablica (Rabuzin, 2011). Klauzula SELECT koristi se za zadavanje upita, a uz nju se redom koriste klauzule FROM, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT I OFFSET. Navedeno odgovara sintaksi naredbe za zadavanje upita:

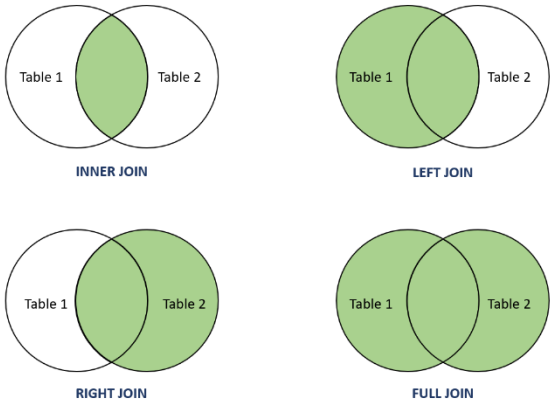
```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
    [ HAVING condition ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS {
     FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH
     TIES } ]
    [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF
    table_name [, ...] ] [ NOWAIT | SKIP LOCKED ] [...] ]
```

Podatke koji su prikazani izvršavanjem upita moguće je sortirati uzlazno ključnom riječi ASC (engl. *ascending*) ili silazno ključnom riječi DESC (engl. *descending*). Moguće je da se u rezultatima upita jedna vrijednost pojavi više puta što se izbjegava korištenjem opcije DISTINCT.

Upiti koji se zadaju naredbom SELECT mogu biti jednostavni, kod kojih je u FROM klauzuli samo jedna tablica, ili složeni koji dohvaćaju podatke iz više tablica. Ukoliko se radi o složenom upitu, tablice iz kojih se dohvaćaju podaci potrebno je ispravno spojiti pri čemu se koriste definirani primarni i vanjski ključevi. Samo povezivanje tablica može se raditi korištenjem klauzula FROM i WHERE no to je najstarija mogućnost spajanja tablica i manje se koristi. Nova sintaksa spajanja tablica koristi klauzulu JOIN, a osnovna prednost nove sintakse je odvojenost uvjeta za spajanje tablice i uvjeta za filtriranje redova (Rabuzin, 2011). Sintaksa klauzule JOIN za povezivanje tablica je sljedeća:

```
FROM tablica1 [NATURAL] join_type tablica2
[ON join_condition | USING (join_column [,…])]
```

Prilikom korištenja JOIN naredbe potrebno je odabrati vrstu spajanja (engl. *join type*), a ona ovisi o zahtjevima upita. Vrste spajanja su [INNER] JOIN, LEFT i RIGHT [OUTER] JOIN, FULL [JOIN te CROSS JOIN koji je često rezultat pogreške, a značenje pojedine vrste spajanja prikazano je na slici 2.



Slika 2. Vrste spajanja (AlphaCodingSkills, 27. lipnja 2024.)

3.1.10. Indeksi

Indeksi su objekt sustava za upravljanje bazama podataka koji služe bržem pretraživanju redova u tablicama odnosno poboljšavanju performansi. Podaci u tablicama nisu smješteni na određen način već na lokaciju gdje ima mjesta što značajno produžuje vrijeme pronalaska potrebnih podataka. S druge strane, podaci u indeksima, gdje se obično sprema jedan ili više stupaca, pohranjeni su određenim poretkom – često sortirani uzlazno. Prema

tome, prilikom pretraživanja indeksa moguće je izravno pozicioniranje i dohvat traženih podataka umjesto sekvencijalnog čitanja. Kada sustav pronađe traženu vrijednost (ključ) u indeksu, pomoću pohranjene adrese podatka može izravno pristupiti podatku u tablici i tako značajno poboljšati performanse baze podataka (Rabuzin, 2011). Sintaksa kreiranja i brisanja indeksa prikazana je u nastavku.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ]
    ON [ ONLY ] table_name [ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [
     opclass [ ( opclass_parameter = value [,... ] ) ] ][ ASC | DESC]
    [ NULLS { FIRST | LAST } ] [,...] )
    [ INCLUDE ( column_name [,...] ) ]
    [ NULLS [ NOT ] DISTINCT ]
    [ WITH ( storage_parameter [= value] [,... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]

DROP INDEX [ IF EXISTS ] name [,...] [ CASCADE | RESTRICT ]
```

3.1.11. Pogledi

Pogledi su objekti u bazi podataka koji su usko vezani uz upite pošto se u suštini baziraju na pohranu upita u katalog sustava. Pogledi se definiraju kao virtualne tablice koje ne sadrže podatke (nematerijalizirane), a prilikom izvršavanja pogleda izvršava se upravo upit koji je pohranjen u pogledu. Pogledi se koriste za prikaz stupaca u obliku jedne tablice, ograničavanje pristupa određenim podacima i agregiranje (grupiranje) podataka.

Kada govorimo o vrstama pogleda, sagledava se rezultat pogleda (upita) naspram početne tablice pa tako postoje horizontalni pogledi koji eliminiraju redove iz početne tablice, vertikalni pogledi koji eliminiraju stupce te spojni pogledi kojima se tablice spajaju (Rabuzin, 2011). Sintaksa kreiranja i brisanja pogleda prikazana je u nastavku.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name
[,...] ) ]
    AS query

DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

3.1.12. Transakcije

Kako navodi Rabuzin (2011), transakcije su niz naredbi čije je osnovno svojstvo atomnost - naredbe se izvršavaju sve u kompletu ili se ne izvršava nijedna. S obzirom da su transakcije niz naredbi, možemo reći da je svaka naredba implicitna transakcija koja se izvršava jedna po jedna i među izvršavanjima nema ovisnosti. U slučaju da se više naredbi izvršava kao cjelina, govorimo o kreiranju eksplicitnih transakcija.

Kreiranje eksplicitne transakcije započinje naredbom BEGIN što označava početak transakcije, a završava ili naredbom COMMIT koja označava uspješan završetak transakcije ili naredbom ROLLBACK koja označava neuspješan završetak transakcije. Upravo između naredbi za početak i kraj transakcije nalazi se niz SQL naredbi pa se sintaksa transakcije može napisati ovako:

```
BEGIN;
SQL naredbe
COMMIT|ROLLBACK;
```

3.1.13. Funkcije

Slično funkcijama u programskim jezicima, u PostgreSQL sustavu funkcije omogućavaju izvršavanje niza naredbi. Funkcija u suštini vraća rezultat koji je istovjetan rezultatu zadnjeg upita unutar funkcije, a u slučaju da posljednji upit ne vraća ništa – rezultat funkcije je NULL (Rabuzin, 2014). Naredba za kreiranje funkcije je CREATE FUNCTION dok opcija REPLACE služi promjeni postojeće funkcije. Nakon navođenja imena funkcije specificiraju se parametri, a uz RETURNS se navodi tip rezultata kojeg funkcija vraća. Samo tijelo funkcije, koje se sastoji od SQL naredbi, piše se unutar jednostrukih navodnika ili dva znaka dolara iza AS.

Prilikom kreiranja funkcije potrebno je navesti i jezik u kojemu je funkcija implementirana, a sintaksa za kreiranje i brisanje funkcije prikazana je u nastavku.

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = }
default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | WINDOW
    | { IMMUTABLE | STABLE | VOLATILE }
    | [ NOT ] LEAKPROOF
    | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
    | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
}
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT
}
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
DROP FUNCTION [ IF EXISTS ] name [ ( [ [ argmode ] [ argname ]
argtype [, ...] ] ) ] [, ...]
    [ CASCADE | RESTRICT ]
```

3.1.14. Okidači

Aktivne baze podataka u suštini su klasične baze podataka koje su proširene dodatnim komponentama, konkretno izvršavanjem akcija temeljem određenih jednostavnih ili složenih događaja u bazi podataka poput unosa ili brisanja podataka. Aktivna pravila, odnosno ECA (engl. *Event Condition Action*), prikazuju sposobnost automatskog reagiranja baze podataka temeljem definiranih događaja pri čemu se akcija izvršava automatski bez intervencije korisnika ili neke aplikacije. Za implementaciju aktivnih pravila potrebno je definirati događaje koji pokreću reakciju, uvjete reagiranja te akcije koje se izvršavaju, a sama implementacija se najčešće vrši putem okidača (Rabuzin, 2014).

Okidači u bazi podataka su objekti koje je pokrenuo neki događaj, najčešće DML naredbe, a specificiraju automatsko izvršavanje određene funkcije kad god se izvršava određena operacija u bazi. Prilikom kreiranja okidača, najprije je potrebno kreirati funkciju koja se izvršava kao reakcija na događaj, a zatim specificirati okidač funkcije na čijem se kraju navodi sama funkcija i njeni argumenti. Sintaksa za kreiranje funkcije napisana je u prethodnom poglavlju treće razine, a sintaksa za kreiranje okidača dana je u nastavku:

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER |
INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE |
INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ]
transition_relation_name } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

3.2. ERA model

Naziv ERA modela dolazi iz engleskog jezika u kojemu je puno ime modela *Entity Relationship Attribute model*, a sam model moćan je alat u fazi dizajniranja baza podataka. ERA model predstavlja konceptualni model podataka koji se koristi za predstavljanje strukture baze podataka na visokoj razini apstrakcije. Korištenjem ERA modela prilikom dizajna baza podataka moguće je fokusirati se na logičke odnose unutar strukture baze i izraditi pravilan model podataka s obzirom na poslovna pravila; čime ERA model pospješuje razumijevanje i mogućnosti prilagodbe modela shodno potrebama.

Upravo iza samog cjelovitog naziva modela stoje osnovni elementi ERA modela, a to su entiteti, veze te atributi; temeljem prikaza elemenata ERA modela implementira se baza podataka.

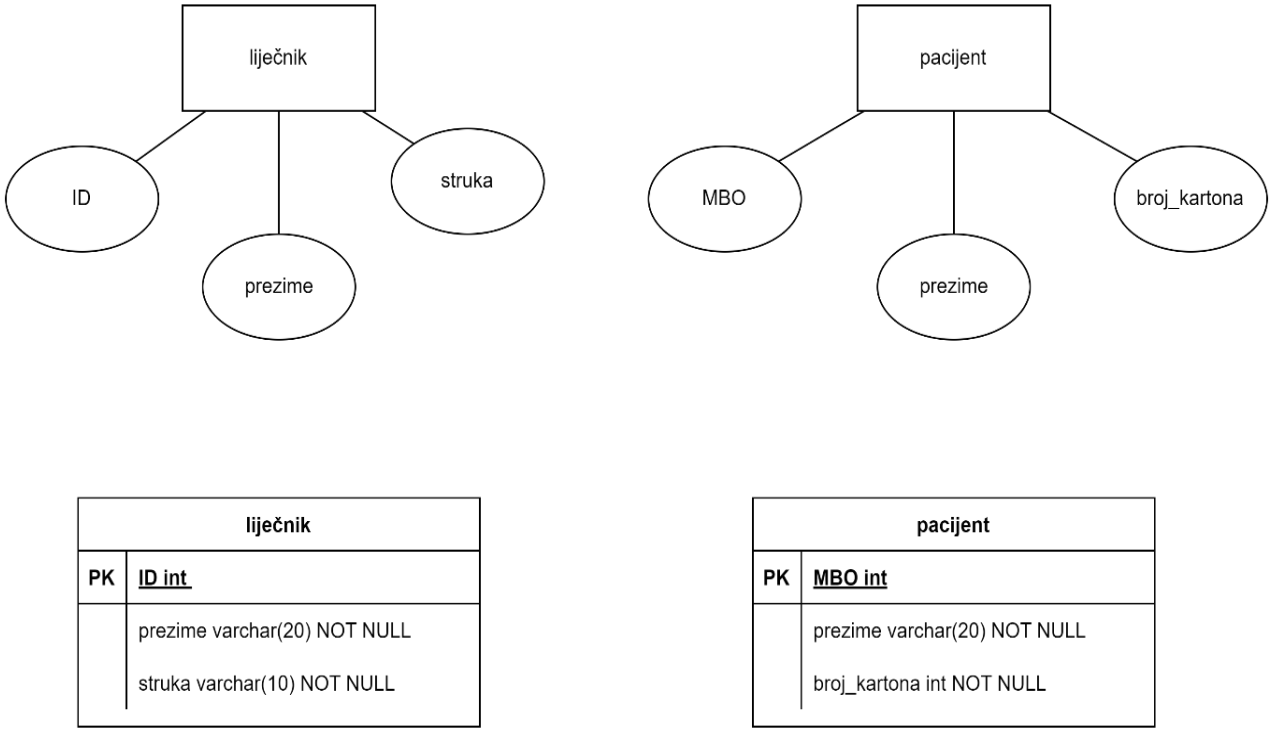
3.2.1. Entiteti i atributi

Uzmemo li ponovno u obzir da je model podataka koji prikazujemo ERA modelom zapravo model aplikacijske domene, jasno je da je potrebno analizom aplikacijske domene uočiti i razumjeti strukturu podataka sustava i fokusirati se na važne elemente u daljnjem dizajnu baze podataka.

Za započinjanje dizajna baze podataka i izradu ERA modela, obično je prvo potrebno uočiti objekte u domeni o kojima se prikupljaju podaci. U ERA modelu upravo entiteti predstavljaju objekte identificirane u domeni za koje se žele pohraniti podaci. Ovisno o aplikacijskoj domeni, ERA modeli sastoje se od različitih entiteta na temelju kojih u implementaciji baze podataka nastaju tablice odnosno relacije. U analizi domene često se uočavaju objekti međusobno slični s obzirom na neka svojstva koji se svrstavaju u tipove entiteta; dakle tip entiteta sastoji se od više objekata koji predstavljaju instancu tipa entiteta. Entiteti se u analizi domene i poslovnih pravila često pojavljuju kao imenice u jednini, no entiteti mogu biti fizički objekti poput osoba, proizvoda, knjiga ili apstraktni pojmovi poput događaja ili transakcija.

Svaki entitet definiraju specifična svojstva koja ga opisuju i karakteriziraju, a u vidu ERA modela, svojstva su poznata kao atributi. Atributi predstavljaju svojstva entiteta koja su važna i potrebna za karakteriziranje svakog tipa entiteta te se upravo vrijednosti atributa pohranjuju u bazu podataka kao informacije o entitetima. Svaki entitet može imati više atributa, a kada govorimo o instanci tipa entiteta – svaki entitet određenog tipa sadrži iste atribute koji se obično razlikuju po vrijednostima koje entitet poprima iz domene. Sukladno tome, različite tipove entiteta karakteriziraju različiti atributi iako je moguće da se određeni atributi među tipovima entiteta poklapaju. Vratimo li se na relacijske baze podataka, kako je entitet u implementaciji predstavljao tablicu odnosno relaciju, atributi predstavljaju stupce odnosno atribute relacija pri implementaciji baze (Maleković i Rabuzin, 2016; Rabuzin, 2014).

Slika 2 prikazuje dva načina crtanja entiteta i atributa. Gornja dva entiteta prikazana su u pravokutnicima, a njihovi atributi nacrtani su u elipsama povezanima s pravokutnicima. S obzirom da je postajalo nepregledno prikazivati entitete s atributima oko njih, danas je uobičajeno da se odgovarajući atributi entiteta prikazuju ispod entiteta u istom pravokutniku, kao što je prikazano u donja dva pravokutnika.



Slika 3. Primjer crtanja entiteta i atributa (autorski rad)

3.2.2. Veze među entitetima

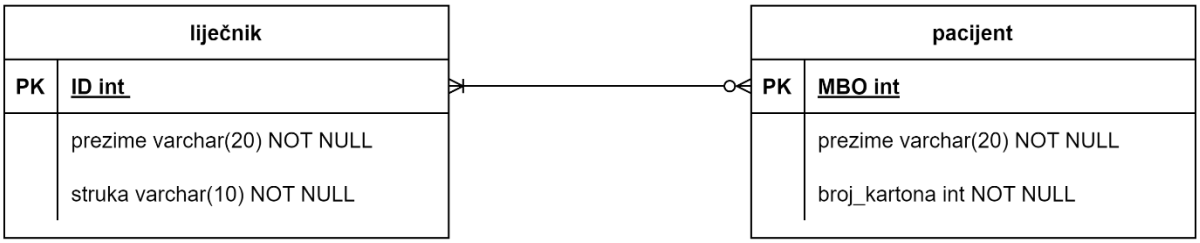
Kao što je navedeno, treći osnovni element ERA modela su tipovi veza, ili kraće, veze među entitetima. Veze predstavljaju logičke odnose među entitetima koje proizlaze iz domene i poslovnih pravila. Tip veze u dijagramu povezuje entitet određenog tipa entiteta s drugim entitetom određenog tipa, odnosno tipovi veze kao imenovane asocijacije povezuju instance tipova entiteta. U razmatranju karakteristike tipa veze moguće je analizirati broj tipova entiteta uključenih u vezu, kardinalnost veze te opcionalnost veze.

Kardinalnost i opcionalnost vezani su uz sve tipove veza neovisno o vrsti što se tiče broja participanata, a naznačuju se na krajevima veze u ERA modelu.

Kardinalnost u suštini označuje maksimalan broj sudionika u vezi; odnosno uz sam tip entiteta je naznačeno sudjeluje li u vezi nula, jedan ili više entiteta tj. instanca entiteta.

S druge strane, opcionalnost predstavlja minimalan broj sudionika u vezi. Kada je u pitanju opcionalnost, govorimo o tome mora li instanca entiteta sudjelovati u vezi ili njezino sudjelovanje nije potrebno, odnosno opcionalno je. Prema tome, pored kardinalnosti, uz svaki entitet se na vezi naznačuju oznake nula ili jedan, to jest opcionalno ili obavezno sudjelovanje instance tipa entiteta u vezi (Rabuzin, 2014; Maleković i Schatten, 2017).

Slika 3 prikazuje vezu prethodno korištenih entiteta s ciljem pojašnjenja kardinalnosti i opcionalnosti veze, u ovom slučaju binarne. Prikazom ovakvog tipa veze možemo zaključiti da jednog pacijenta (instancu entiteta tipa pacijent) liječi jedan ili više liječnika dok jedan liječnik (instanca entiteta tipa liječnik) može liječiti više pacijenata. S aspekta opcionalnosti, pacijenta mora liječiti barem jedan liječnik ili mora biti upisan kod njega, kako bi on uopće bio pacijent; dok je moguće da liječnik ne liječi nijednog pacijenta, recimo u slučaju da je zaposlen u laboratoriju. Prema tome, u ovom slučaju je entitet tipa pacijent obavezan sudionik veze dok je sudjelovanje entiteta tipa liječnik opcionalno.

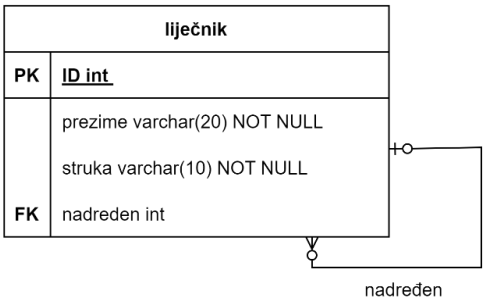


Slika 4. Primjer kardinalnosti i opcionalnosti (autorski rad)

Osim kardinalnosti i opcionalnosti, tip veze određuje i broj tipova entiteta uključenih u vezu. Sagledavamo li veze u smislu broja participanata, razlikujemo unarne veze, binarne veze, ternarne veze i veze višeg reda.

3.2.2.1. Unarna veza

Unarne veze su veze koje povezuju entitete istog tipa, odnosno u ERA modelu unarna veza povezuje entitet sa samim sobom zbog čega se ponekad za unarne veze koristi i naziv rekurzivna veza. Unarne veze konkretno prikazuju odnose među instancama entiteta istog tipa pri čemu se često uključuju odnosi nadređenosti i podređenosti s ciljem prikaza hijerarhije određenog tipa entiteta. Jedan primjer odnosa nadređenosti i podređenosti za prikaz hijerarhije kada je potrebno koristiti unarni tip veze prikazan je na slici 4.



Slika 5. Primjer unarne veze 1:M (autorski rad)

Poslovno pravilo na temelju kojeg je modelirana unarna veza jedan naprema više definira da jedan liječnik može imati više sebi podređenih liječnika ili ne mora imati podređene liječnike. S druge strane, jedan liječnik ili nema nadređenog liječnika ili mu je nadređen točno jedan liječnik. Na primjer, ako uzmemo u obzir domenu jedne bolnice, liječnik ravnatelj je kao voditelj bolnice nadređen više liječnika dok on osobno nema nadređenog liječnika. S druge strane, liječniku članu odjela radiologije nadređen je točno jedan liječnik - voditelj odjela radiologije koji može imati više podređenih liječnika, a nadređen mu je ponovno liječnik ravnatelj.

Što se tiče implementacije unarne veze jedan naprema više iz ovog primjera, prilikom kreiranja relacije u strukturu se, osim atributa koji opisuju tip entiteta liječnik, dodaje vanjski ključ koji se referencira na istu relaciju. Implementacija unarne veze tipa entiteta liječnik i popunjavanje tablice s podacima u PostgreSQL nalazi se u nastavku.

Kreiranje tablice *lijecnik*:

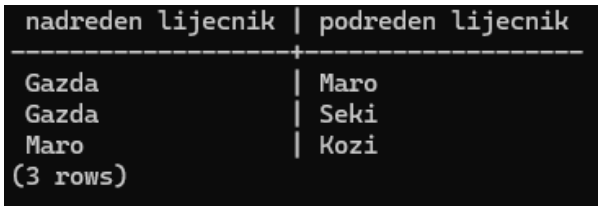
```
CREATE TABLE lijecnik (
ID INTEGER PRIMARY KEY,
prezime VARCHAR(20) NOT NULL,
struka VARCHAR(10) NOT NULL,
nadreden INTEGER REFERENCES lijecnik
ON UPDATE CASCADE ON DELETE SET NULL);
```

Unos podataka u tablicu *lijecnik*:

```
INSERT INTO lijecnik VALUES (1, 'Gazda', 'S1');
INSERT INTO lijecnik VALUES (2, 'Maro', 'S1', 1);
INSERT INTO lijecnik VALUES (3, 'Seki', 'S2', 1);
INSERT INTO lijecnik VALUES (4, 'Kozi', 'S2', 2);
```

U svrhu pojašnjenja i razumijevanja unesenih podataka, kreiran je upit koji prikazuje hijerarhiju liječnika u trenutnom primjeru.

```
SELECT l1.prezime AS "nadreden lijecnik",
l2.prezime AS "podreden lijecnik"
FROM lijecnik l1 INNER JOIN lijecnik l2
ON l1.id=l2.nadreden;
```



Slika 6. Prikaz hijerarhije liječnika (autorski rad)

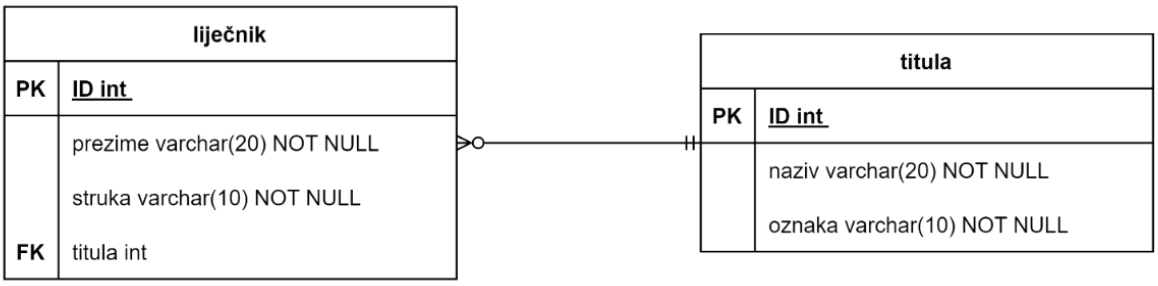
Rezultati upita prikazani su na slici 5, a moguće ih je tumačiti tako da kažemo da liječnik Gazda ima dva podređena liječnika – Maro i Seki, dok je liječnik Maro nadređen liječniku Kozi.

U ovom slučaju, liječnik Gazda nema nadređenog liječnika dok liječnici Seki i Kozi nemaju sebi podređene liječnike.

3.2.2.2. Binarna veza

Binarne veze su veze koje povezuju dva različita tipa entiteta i najčešće se pojavljuju u dizajniranju i implementaciji baza podataka. U kontekstu kardinalnosti, možemo govoriti o binarnim vezama jedan naprema jedan (1:1) u kojoj instanca jednog tipa entiteta može biti povezana s jednom instancom drugog tipa entiteta i obrnuto te ona predstavlja poseban oblik veze jedan naprema više. Binarna veza jedan naprema više (1:M) predstavlja vezu u kojoj jedna instanca jednog tipa entiteta može biti povezana s više instanci drugog tipa entiteta dok instanca drugog tipa entiteta može biti povezana samo s jednom instancom prvog tipa entiteta. Najsloženija vrsta binarne veze je veza više naprema više (M:N) u kojoj, prema prethodnoj logici, jedna instanca prvog tipa entiteta može biti povezana s više instanci drugog tipa entiteta te jedna instanca drugog tipa entiteta može biti povezana s više instanci prvog tipa entiteta. Sa strane opcionalnosti, instance entiteta mogu u vezi sudjelovati obavezno (mandatorno) ili neobavezno (opcionalno).

Primjer binarne veze jedan naprema više (1:M) prikazan je u nastavku.



Slika 7. Primjer binarne veze 1:M (autorski rad)

Poslovno pravilo na temelju kojeg je modelirana binarna veza jedan naprema više definira da liječnik može u jednom trenutku imati jednu titulu, na primjer primarius ili pripravnik, i mora mu biti dodijeljena titula. S druge strane, jedna titula može biti dodijeljena više liječnika, ali je moguće da jedna titula, u nekom trenutku u domeni jedne bolnice, nije dodijeljena nijednom liječniku; na primjer bolnica trenutno nema pripravnika.

Što se tiče implementacije binarne veze jedan naprema više iz ovog primjera, prilikom kreiranja relacije u strukturu se, osim atributa koji opisuju tip entiteta liječnik, dodaje vanjski ključ koji se referencira na relaciju titule kako bi se uz atribute liječnika navela i njegova titula. Implementacija binarne veze tipa entiteta titula i tipa entiteta liječnik te popunjavanje tablica s podacima u PostgreSQL nalaze se u nastavku.

Kreiranje tablice *titula*:

```
CREATE TABLE titula (
ID INTEGER PRIMARY KEY,
naziv VARCHAR(20) NOT NULL,
oznaka VARCHAR(10) NOT NULL);
```

Kreiranje tablice *lijecnik*:

```
CREATE TABLE lijecnik (
ID INTEGER PRIMARY KEY,
prezime VARCHAR(20) NOT NULL,
struka VARCHAR(10),
titula INTEGER REFERENCES titula
ON UPDATE CASCADE ON DELETE SET NULL);
```

Unos podataka u tablicu *titula*:

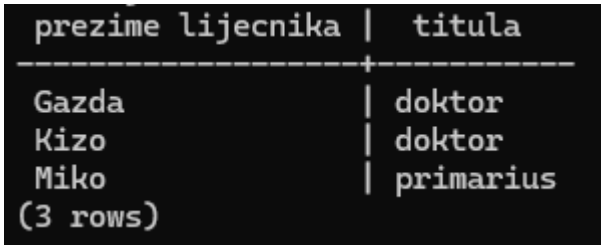
```
INSERT INTO titula VALUES
(1, 'primarius', 'prim'),(2, 'doktor', 'dr'), (3, 'specijalizant',
'spec');
```

Unos podataka u tablicu *lijecnik*:

```
INSERT INTO lijecnik VALUES
(1, 'Gazda', 'S1', 2), (2, 'Miko', 'S2', 1), (3, 'Kizo', 'S1', 2);
```

Kreiranje upita koji prikazuje prezimena liječnika i njihove titule, poredane prema nazivima titula.

```
SELECT l.prezime AS "prezime lijecnika",
t.naziv AS "titula"
FROM titula t INNER JOIN lijecnik l
ON t.id=l.titula ORDER BY 2;
```



Slika 8. Prikaz liječnika i titula (autorski rad)

Rezultati upita prikazani su na slici 7, a moguće ih je tumačiti tako da kažemo da liječnici Gazda i Kizo imaju titulu doktora, dok liječnik Miko ima titulu primarijusa.

Na ovom primjeru vidimo da je kreiranom binarnom vezom zadovoljeno poslovno pravilo da jednu titulu može imati više liječnika, ali da ne moraju sve titule u domeni jedne bolnice biti dodijeljene – u ovom slučaju niti jedan liječnik nema titulu specijalizanta.

3.2.2.3. Ternarna veza

Ternarne veze su veze koje povezuju tri različita tipa entiteta i koriste se kada binarne veze nisu dovoljne za opis odnosa između tri tipa entiteta. Kada binarne veze nisu dovoljne za implementaciju točnog odnosa tri entiteta prema poslovnim pravilima, u model se uvodi slabi entitet koji je povezan sa ostala tri entiteta. U kontekstu kardinalnosti, možemo govoriti o ternarnim vezama više naprema više naprema više (M:N:P), ternarnim vezama više naprema više naprema jedan (M:N:1), ternarnim vezama više naprema jedan naprema jedan (M:1:1) te ternarnim vezama jedan naprema jedan naprema jedan (1:1:1).

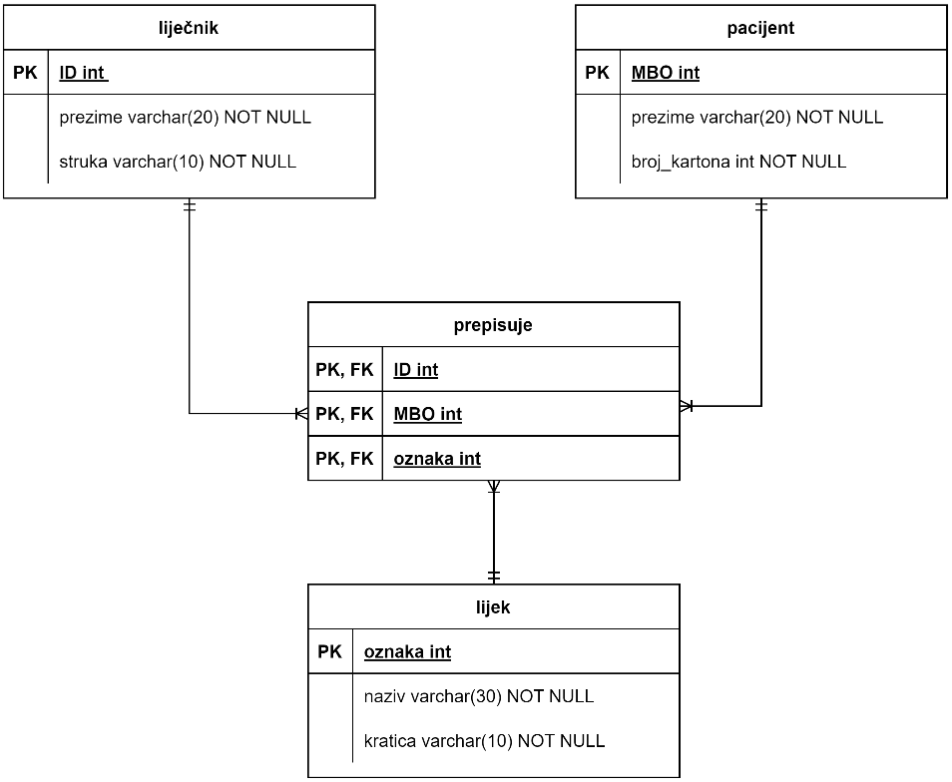
Kada se radi o ternarnim vezama, ključno je pravilno odrediti primarne i vanjske ključeve. Slabi entitet koji je uključen kako bi uspješno implementirali ternarnu vezu sadrži vanjske ključeve na preostale tri tablice koje povezuje. Međutim, ovisno o tipu ternarne veze glede kardinalnosti, primarni ključ slabog entiteta definira se na određene načine.

Kada se radi o ternarnoj vezi više naprema više naprema više (M:N:P), primarni ključ slabog entiteta je složeni primarni ključ koji se sastoji od svih vanjskih ključeva na sva tri tipa entiteta.

Radi li se o ternarnoj vezi više naprema više naprema jedan (M:N:1), primarni ključ slabog entiteta je složeni primarni ključ koji se sastoji od dva vanjska ključa tipova entiteta koji su povezani vezom više.

U slučaju ternarne veze više naprema jedan naprema jedan (M:1:1), primarni ključ slabog entiteta je složeni primarni ključ koji se sastoji od vanjskog ključa tipa entiteta povezanog vezom više te jednog vanjskog ključa tipa entiteta povezanog vezom jedan po izboru.

Posljednja situacija je ternarna veza jedan naprema jedan naprema jedan (1:1:1) u kojoj se primarni ključ slabog entiteta sastoji od vanjskih ključeva dva tipa entiteta povezanih vezom jedan po izboru. Primjer ternarne veze tipa više naprema više naprema više (M:N:P) slijedi u nastavku.



Slika 9. Primjer ternarne veze M:N:P (autorski rad)

Prema poslovnom pravilu na temelju kojeg je modelirana ternarna veza više naprema više naprema više (M:N:P) ne možemo definirati nikakva ograničenja nad kombinacijama atributa liječnik, pacijent i lijek. Konkretno, jedan liječnik može prepisati jednom pacijentu više lijekova; jednom pacijentu može biti prepisan jedan lijek od strane više liječnika; jedan liječnik može prepisati jedan lijek više pacijenata. Odnosno, u tipu ternarne veze (M:N:P) za svaku kombinaciju parova instanci entiteta postoji više instanca trećeg tipa entiteta (Rabuzin, 2014).

Što se tiče implementacije ternarne veze više naprema više naprema više iz ovog primjera, prilikom kreiranja relacija *liječnik, pacijent i lijek* potrebno je kreirati i relaciju temeljenu na slabom entitetu koja reprezentira ternarnu vezu – *prepisuje*. S obzirom da se radi o ternarnoj vezi (M:N:P), tablica *prepisuje* sadrži vanjske ključeve na tablice *liječnik, pacijent* i *lijek* te koristimo pravilo koje upućuje da se složeni primarni ključ slabog entiteta sastoji od svih vanjskih ključeva sva tri tipa entiteta kao što je prikazano na slici 8.

Implementacija ternarne veze tipa entiteta liječnik, tipa entiteta pacijent i tipa entiteta lijek te popunjavanje tablica s podacima u PostgreSQL nalaze se u nastavku.

Kreiranje tablice *lijecnik*:

```
CREATE TABLE lijecnik (
ID INTEGER PRIMARY KEY,
```

```
prezime VARCHAR(20) NOT NULL,
struka VARCHAR(10));
```

Kreiranje tablice *pacijent*:

```
CREATE TABLE pacijent (
MBO INTEGER PRIMARY KEY,
prezime VARCHAR(20) NOT NULL,
broj_kartona INTEGER NOT NULL);
```

Kreiranje tablice *lijek*:

```
CREATE TABLE lijek (
oznaka INTEGER PRIMARY KEY,
naziv VARCHAR(30) NOT NULL,
kratica VARCHAR(10) NOT NULL);
```

Kreiranje tablice *prepisuje*:

```
CREATE TABLE prepisuje (
lijecnik_id INTEGER REFERENCES lijecnik
ON UPDATE CASCADE ON DELETE CASCADE,
pacijent_MBO INTEGER REFERENCES pacijent
ON UPDATE CASCADE ON DELETE CASCADE,
oznaka_lijek INTEGER REFERENCES lijek
ON UPDATE CASCADE ON DELETE CASCADE,
PRIMARY KEY(lijecnik_id, pacijent_MBO, oznaka_lijek));
```

Unos podataka u tablicu *lijecnik*:

```
INSERT INTO lijecnik VALUES
(1, 'Gazda', 'S1'), (2, 'Miko', 'S2'), (3, 'Kizo', 'S2');
```

Unos podataka u tablicu *pacijent*:

```
INSERT INTO pacijent VALUES
(321, 'Bazic', 123), (654, 'Rubi', 456), (987, 'Jovic', 789);
```

Unos podataka u tablicu *lijek*:

```
INSERT INTO lijek VALUES
(19, 'Linex', 'Lx'), (28, 'Andol', 'And'), (37, 'Brufen', 'Bruf');
```

Unos podataka u tablicu *prepisuje*:

```
INSERT INTO prepisuje VALUES
(1, 321, 37), (2, 987, 19), (2, 987, 28), (3, 321, 37), (3, 654, 19);
```

Kreiranje upita koji prikazuje prezimena liječnika, pacijente i lijekove koji su im prepisani, poredane prema prezimenima liječnika.

```
SELECT l.prezime AS "lijecnik",
p.prezime AS "pacijent",
k.naziv AS "lijek"
FROM lijecnik l
JOIN prepisuje pr ON l.ID = pr.lijecnik_id
JOIN pacijent p ON pr.pacijent_MBO = p.MBO
```

```
JOIN lijek k ON pr.oznaka_lijek = k.oznaka
ORDER BY 1;
```

```
 lijecnik | pacijent | lijek
----------+----------+--------
 Gazda    | Bazic    | Brufen
 Kizo     | Bazic    | Brufen
 Kizo     | Rubi     | Linex
 Miko     | Jovic    | Linex
 Miko     | Jovic    | Andol
(5 rows)
```

Slika 10. Prikaz liječnika, pacijenata i prepisanih lijekova (autorski rad)

Rezultati upita koji je kreiran prikazani su na slici 8, a kroz njih možemo potvrditi poslovno pravilo navedeno prilikom modeliranja ternarne veze. Na primjer, liječnik Kizo liječi više pacijenata i prepisuje im više lijekova, dok liječnik Gazda liječi jednog pacijenta, pri čemu je pacijent Bazić pacijent više liječnika – Kize i Gazde. Nadalje, Pacijent Jović je pacijent samo liječnika Mike, ali mu je prepisano više lijekova – Linex i Andol. Ovako popunjenim tablicama i prikazanim upitom potvrđuje se da u ovom slučaju za svaku kombinaciju parova instanci tipa entiteta postoji više instanca trećeg tipa entiteta te je bilo potrebno kreirati ternaru vezu tipa (M:N:P) implementirajući i slabi entitet (Rabuzin, 2014).

4. Nerelacijske (NoSQL) baze podataka

Nerelacijske baze podataka predstavljaju alternativu tradicionalnim relacijskim bazama podataka te su u današnje vrijeme velikih podataka sve popularnije. Sadalage i Fowler (2015) navode kako relacijske baze podataka predstavljaju uspješnu tehnologiju koja godinama pruža točnost i dosljednost radu s podacima te su bile prvi izbor u području upravljanja ozbiljnim pohranama podataka visoke sigurnost. Međutim, najznačajniji utjecaj na novonastale promjene u području baza podataka predstavlja pojava velikih podataka (engl. *Big Data*) i potreba za podržavanjem velike količine podataka radom na klasterima gdje nastupaju upravo nerelacijske baze podataka koje pružaju znatno fleksibilnije modele podataka.

4.1. NoSQL baze podataka

NoSQL baze podataka, dolazeći od naziva *Not Only SQL*, su nerelacijske baze podataka koje koriste različite modele za skladištenje i upravljanje podacima i tako pružaju fleksibilnost u području baza podataka prijeko potrebnu za sve veću količinu velikih podataka koja nastaje svakodnevno. Iako naziv NoSQL upućuje na nedostatak mogućnosti korištenja SQL jezika to nije u potpunosti točno. Prateći razvoj područja istraživanja i rada s bazama podataka, pojedini sustavi za upravljanje bazama podataka (SUBP) prvotno namijenjeni radu s NoSQL bazama podataka uključuju SQL sučelja u svoje sustave kako bi pružili mogućnost povezivanja s relacijskim bazama podataka koje još uvijek imaju značajnu ulogu u području baza podataka. Posljedica navedenog je da nije moguće jednostavno raspodijeliti sustave za upravljanje bazama podataka na one koji pružaju upravljanje relacijskim (SQL) bazama podataka i one specijalizirane za nerelacijske (NoSQL) baze podataka, kako navodi Hills (2016). Približavanje SQL i NoSQL baza podataka može se pojasniti i činjenicom da se akronimi ACID i BASE u današnje vrijeme ne odnose isključivo na SQL (ACID) ili NoSQL (BASE). Akronim ACID predstavlja četiri karakteristike koje implementiraju SQL baze podataka; atomnost (engl. *atomicity*), konzistentnost (engl. *consistency*), izoliranost (engl. *isolation*) i trajnost (engl. *duraility*) koje su, prema Hillsu (2016), pojašnjene u nastavku.

Karakteristika atomnosti govori da će se SQL transakcije u SUBP ponašati nedjeljivo, odnosno jedini načini na koji se transakcije može odviti je da se u potpunosti izvrši ili bude u potpunosti neuspješna.

Konzistentnost je značajna karakteristika SQL-a kojom je definirano da će se transakcije u SUPB odviti isključivo ukoliko su sva primijenjena ograničenja zadovoljena. Konkretno, ukoliko u bilo kojem trenutku transakcija ne zadovolji određeno ograničenje baze

podataka, ista se prekida i označava neuspješnom te se sve dotadašnje promjene koje je uzrokovala vraćaju na početno stanje prije početka transakcije.

Izolacija je garancija koju pružaju SUBP onemogućavajući transakcijama uvide u dijelove rezultata ostalih transakcije. Konkretno, u slučaju da SUBP obrađuje više transakcija istovremeno, jedini podaci i rezultati koje ostale transakcije mogu uvidjeti su rezultat uspješno obrađene transakcije.

Karakteristika trajnosti također je vezana uz obradu transakcija, a njome je garantirano da će podaci rezultirani uspješnom transakcijom trajno biti pohranjeni i vidljivi u bazi podataka neovisno o daljnjem utjecaju rušenja sustava ili ponovnog pokretanja.

Nastavno na analizu akronima ACID, akronim BASE (engl. *basic availability, soft state, eventual consistency*) karakterističan je za NoSQL baze podataka i znatno pojašnjava zašto nerelacijske baze podataka pružaju mogućnost upravljanja velikim količinama podataka, (engl. *Big Data).*

Basic availability, odnosno osnovna dostupnost, definira da je većina podataka u nerelacijskim bazama podataka dostupna većinu vremena. Iako to na prvu zvuči čudno, NoSQL radi s toliko velikim količinama podataka da se mora oslanjati na činjenicu da ne može osigurati dostupnost apsolutno svih podataka u baš svakom trenutku.

Soft state karakteristika NoSQL baza podataka „dozvoljava“ da se pojedini *commit* odnosno izvršavanje transakcije izgubi. Konkretno, to znači da iako je neka transakcija uspješno izvršena, može se dogoditi da se rezultati iste ne pohrane.

Eventual consistency je karakteristika koju se može tumačiti kao da će se dosljednost ispuniti „kad-tad“*,* a govori o tome da će se kopije podataka, koje se u trenutcima pogrešaka unutar sustava ne sinkroniziraju, najzad sinkronizirati i sustići ispravne kopije podataka.

Kako navodi Hills (2016), akronim BASE može zvučati pomalo nesigurno i nepouzdano, ali važno je u obzir uzeti ogromnu količinu velikih podataka kojima NoSQL upravlja. U takvom kontekstu, jednostavnije je razumjeti da su odstupanja od čvrstoće ACID karakteristika prema postizanju BASE akronima ipak dovoljno efikasna za uspješnost rada s velikim bazama podataka.

Povezanost ova dva akronima leži upravo u tome da više nisu direktno vezani za jedan ili drugi tip baza podataka. Današnje nerelacijske baze podataka sve se više i više mogu okarakterizirati akronimom ACID dok se relacijske baze u situacijama velikih količina podataka počinju djelomično priklanjati odlikama akronima BASE.

Glavne odlike NoSQL baza podataka koje vrijedi istaknuti su nepostojanje fiksne sheme, dobar rad na klasterima, *open source* te tradicionalnog klasičnog relacijskog modela podataka. Karakteristike nerelacijskih baza podataka, prema Malekoviću i Rabuzinu (2016.), mogu se pojasniti kraticom 3V koja dolazi od riječi engl. *Volume, Velocity, Variety*.

Kao što je nekoliko puta spomenuto, nerelacijske baze podataka ponajviše se koriste za obrade velikih podataka kada relacijski modeli podataka gube na uspješnosti, prema čemu i dolazimo do pojašnjenja riječi *Volume*.

Velocity označava brzinu kojom se mijenjaju podaci kada govorimo o *Big Data*. Osim samih vrijednosti podataka, izuzetno brzo mijenja se i njihova struktura što predstavlja popriličan problem ukoliko se upravljanje podacima obavlja kroz relacijski model podataka. U takvim situacijama, na važnosti dobivaju fleksibilni modeli podataka koje pruža NoSQL.

Variety, raznolikost označava situacije u kojima podaci nisu uvijek potpuni, što na primjer predstavlja problem kod relacijskih baza podataka u postizanju entitetskog integriteta, i značajno se razlikuju u trenutku vremena. Jedan od glavnih razloga za navedeno su različiti izvori velikih podataka, što može biti skoro cijeli Internet, što onemogućava modeliranje strogo definirane, fiksne sheme podataka.

Zaključno, nerelacijske baze podataka moguće je kategorizirati u četiri grupe; ključ/vrijednost baze podataka (engl. *Key-Value*), grafovske baze podataka, dokumentne baze podataka (engl. *Document-oriented*) te stupčane baze podataka (engl. *Column-oriented*); a svi predstavnici će biti pojašnjeni u nastavku rada.

4.1.1. Ključ - vrijednost baze podataka

Hills (2016) navodi kako je organiziranje podataka u obliku ključ – vrijednost nerelacijskih baza podataka vjerojatno najjednostavniji način rada s podacima. Razlog tomu leži u činjenici da se u radu s ključ – vrijednost bazama podataka koristi model koji pohranjuje upravo samo parove oblika ključ – vrijednost. Pokušamo li približiti ključ – vrijednost baze podataka relacijskim bazama podataka, možemo zamisliti relaciju sa dva atributa, na primjer ID i naziv, pri čemu je ID primarni ključ ekvivalentan ključu dok naziv predstavlja vrijednost. Ključ – vrijednost baze podataka pristupačne su radi svoje visoke skalabilnosti i jednostavnosti te baš zbog nje vrlo je jednostavno postići visoku efikasnost i performanse. Ipak, problem sa terminologijom ključ – vrijednost, prema Hillsu (2016), nastaje uzmemo li u obzir da su ključevi podaci tekstualnog tipa i sami po sebi već imaju vrijednost. Zbog toga dolazi do situacije da je vrijednost definirana kao dio para u ovoj vrsti NoSQL baza podataka zapravo objekt, to jest instanca, neke klase koja nije poznata sustavu za upravljanje bazama podataka. Također,

SUBP koji su temeljeni na ključ – vrijednost bazama podataka ne primjenjuju ograničenja poput vanjskih ključeva te je od velike važnosti kontrolirati ispravnost unesenih podataka.

Unatoč jednostavnosti, sama vrijednost unutar para ključ – vrijednost može biti jednostavan tip podatka, ali moguće je upravljati i složenim vrijednostima poput lista, setova, slika, JSON, XML, tekst i slično. Također, jednostavnost značajno utječe i na mogućnosti kreiranja upita prilikom upravljanja ključ – vrijednost bazama podataka pa tako upitni jezik izostaje. Za upravljanje ključ – vrijednost NoSQL bazama podataka osnovne naredbe koje se koriste su PUT, GET i DELETE. Naredba PUT koristi se za postavljanje vrijednosti para ključ – vrijednost pri čemu je potrebno obratiti pozornost da je uneseni ključ jedinstven kako nova vrijednost ne bi prepisala staru vrijednost pridruženu ključu. Ukoliko se navedeno izvrši namjerno, moguće je ovu situaciju protumačiti i kao postojanje naredbe za ažuriranje podataka pridruživanjem nove vrijednosti postojećem ključu. Naredba GET koristi se za pronalaženje vrijednosti povezane sa ključem koji je zadan u naredbi te se naredba DELETE koristi za brisanje para čiji je član ključ zadan u naredbi.

Osim jednostavnih tipova podataka, ključevi mogu na mjestu vrijednosti sadržavati i složenije strukture podataka poput lista pri čemu su podržane drugačije operacije (Maleković i Rabuzin, 2016). U Redis-u, ključ – vrijednost sustavu, moguće je upravljati listama s nekoliko jednostavnih naredbi. Naredba RPUSH dodaje element na kraj liste pri čemu se na mjesto ključa navodi sama lista, a na mjesto vrijednosti element liste dok LPUSH dodaje element na početak liste.

```
>RPUSH lijecnici "Gazda"
(integer)1
>LPUSH lijecnici "Kizo"
(integer)2
>RPUSH lijecnici "Miko"
(integer)3
```

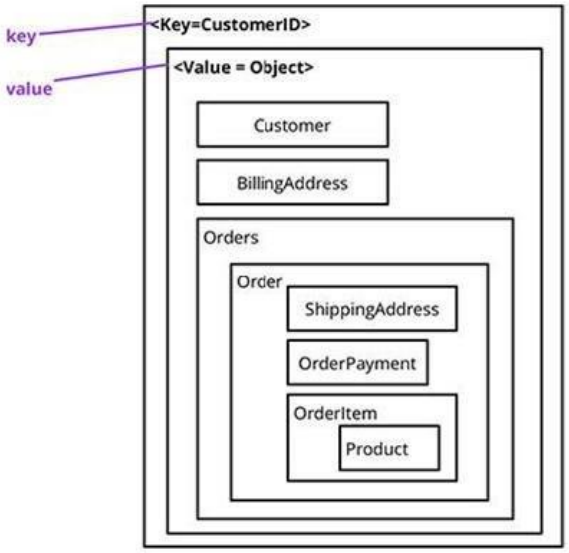
Nakon dodavanja elemenata, moguće je naredbom LLEN saznati dužinu liste, a naredbom LRANGE ispisati elemente liste. Uz naredbu LRANGE potrebno je navesti indeks prvog i zadnjeg elementa kojeg sustav treba ispisati, a u slučaju korištenja -1 kao zadnjeg elementa, ispisuju se svi elementi liste.

```
>LLEN lijecnici
(integer)3
>LRANGE lijecnici 0 1
1) "Kizo"
2) "Gazda"
```

Brisanje elemenata liste u Redis-u moguće je naredbama RPOP, za brisanje zadnjeg elementa, i LPOP, za brisanje prvog elementa.

```
>RPOP lijecnici
"Miko"
>LRANGE lijecnici 0-1
1) "Kizo"
2) "Gazda"
```

Slika 10 prikazuje pohranu podataka o klijentu korištenjem ključ – vrijednost baza podataka. U prikazanoj situaciji, pod jedinstvenim ključem u vrijednost su pohranjeni željeni podaci o klijentu tako da su u jednom objektu pohranjeni svi podaci i postavljeni u jednu „košaru“. Na ovaj način je moguće sve željene podatke o klijentu pohraniti korištenjem ključ – vrijednost baza podataka što je pogodno za pohranu podataka koji se često ažuriraju, poput sesija ili podataka o košarici web trgovina kako navode Sadalage i Fowler (2013). Međutim, jednostavnost ključ – vrijednost baza podataka nije uvijek dovoljno dominantna naprema situacijama u kojima nije pogodno koristiti ovu vrstu NoSQL baza podataka. U slučajevima kada je potrebno kreirati veze među setovima podataka, pohranjivati više ključ – vrijednosti od jednom, koristiti upite nad podacima unutar vrijednosti ili izvršavati operacije nad više ključeva u jednom trenutku, ključ – vrijednost nisu najbolja vrsta nerelacijskih baza podataka za upravljanje podacima.



Slika 11. Ključ - vrijednost pohrana (ResearchGate, 10. lipnja 2024.)

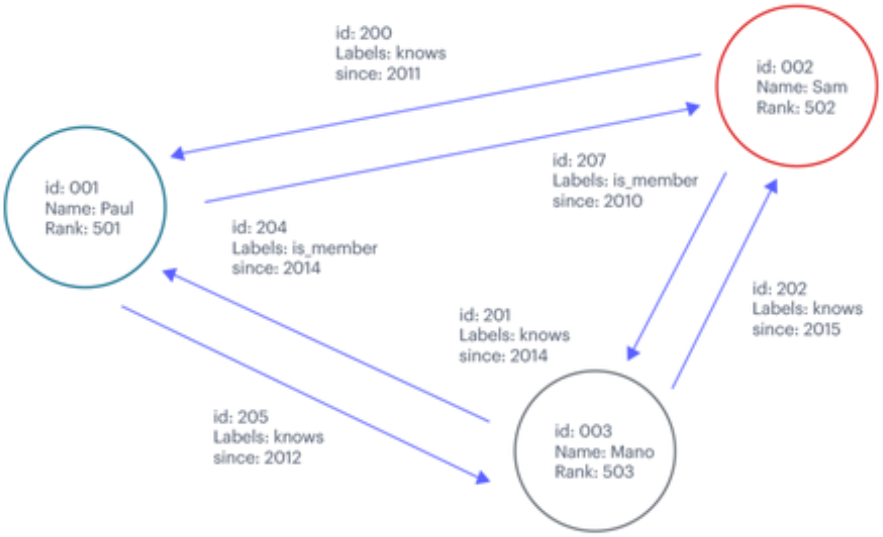
4.1.2. Grafovske baze podataka

U prethodnom poglavlju spomenuto je da ključ – vrijednost baze podataka nisu pogodne za upravljanje podacima koji su međusobno dosta povezani, a grafovske baze podataka korisne su upravo za takve podatke. Grafovske baze podataka jako su slične grafovima i temelje se upravo na pohrani entiteta, koji se mogu poistovjetiti sa čvorovima, i vezama među entitetima odnosno lukovima među čvorovima. Čvorovi (engl. *nodes*) su

instance objekta u aplikaciji koji imaju određena svojstva te su veze među njima ostvarene lukovima (engl. *edges*). S obzirom da lukovi povezuju čvorove, svaki od njih određen je početnim i završnim čvorom, svojim smjerom te svojstvima koja ga određuju.

Kao što je spomenuto, grafovske baze podataka izuzetno su korisne kada su od velike važnosti odnosi među tipovima entiteta i općenito podacima. Odnose među podacima ključno je pravilno definirati i, prilikom modeliranja relacijske baze podataka, poštivati poslovna pravila. Pri tome je često potrebno modelirati veze više naprema više, ternarne ili veze višeg reda što u pojedinim situacijama može biti zahtjevno. Grafovske baze podataka jednostavnije definiraju veze među entitetima, čvorovima, i nisu znatno ograničene na sličnost tipova entiteta.

Slika 11 prikazuje primjer grafovske baze podataka određene društvene mreže. Kao što je prikazano, grafovska baza podataka sastoji se od čvorova (engl. *nodes*), prikazani kao krugovi, za koje su definirana svojstva (engl. *properties*) id, ime i rang, a predstavljaju jednog korisnika društvenih mreža. Čvorovi su međusobno povezani lukovima (engl. *edges*) koji imaju strelicom određen smjer te dodatne oznake (engl. *lables*) koje pobliže definiraju vezu među čvorovima. Konkretno, analiziramo li prikazanu grafovsku bazu podataka možemo zaključiti da korisnik Sam, ranga 502, poznaje korisnika Paul, ranga 501, od 2011. godine. Na isti je način moguće analizirati preostale veze među čvorovima obraćajući pozornost na smjer te je moguće postavljati upite nad grafovskom bazom podataka koji se nazivaju prelaženje (engl. *traversing*).



Slika 12. Grafovska baza podataka (Redis, 10. lipnja 2024.)

SUBP Neo4j je sustav za upravljanje grafovskim bazama podataka koji još nije stekao izuzetnu popularnost među SUBP što se pripisuje relativno novonastaloj primjeni teorije grafova u upravljanju podacima (Maleković i Rabuzin, 2016). Iako Neo4j još nije dostigao veliku

popularnost, jedna od velikih prednosti ovog SUBP je podržavanje transakcija i usklađenosti sa ACID kriterijima. S obzirom da se grafovske baze podataka temelje na čvorovima i vezama među njima, kreiranje grafovske baze u Neo4j temelji se na stvaranju vrhova grafa i bridova među njima.

Kreiranje čvorova u sustavu Neo4j:

```
Node Gazda = graphDb.createNode();
Gazda.setProperty("ime", "Sara Gazda");
Gazda.setProperty("struka", "pedijatrija");
Gazda.setProperty("bolnica", "Opca bolnica Bjelovar");

Node Bazic = graphDb.createNode();
Bazic.setProperty("ime", "Boro Bazic");
Bazic.setProperty("MBO", "987");
Bazic.setProperty("karton", "564-21");
```

Kada su kreirana najmanje dva čvora, moguće je kreirati veze među njima. Prilikom kreiranja veza važno je obratiti pozornost na smjer veze i prema tome, po potrebi, među čvorovima kreirati dvije veze. Konkretno, u primjeru pacijenta i liječnika, Gazda je liječnik od Bazic, ali ne možemo reći da je Bazic liječnik od Gazda pa je kreirana i veza s nazivom pacijent.

Kreiranje veza među bridovima u sustavu Neo4j:

```
Gazda.createRelationshipTo (Bazic, LIJECNIK);
Bazic.createRelationshipTo (Gazda, PACIJENT);
```

U sustavu Neo4j moguće je korištenje indeksa s ciljem bržeg pretraživanja čvorova i veza među njima. S obzirom da veze imaju smjer, za pojedini čvor razlikuju se ulazne i izlazne veze samog čvora (Sadalage i Fowler, 2013). U nastavku je prikazan kod za kreiranje indeksa koji je napisan unutar transakcije.

```
Transaction transakcija = graphDb.beginTx();
try {
      Index<Node> indeksi = graphDb.index().forNodes("nodes");
      indeksi.add(Gazda, "ime", Gazda.getProperty("ime"));
      indeksi.add(Bazic, "ime", Bazic.getProperty("ime"));
      transakcija.success();
} finally {
      transakcija.finish();
}
```

Dohvaćanje ulaznih veza čvora Gazda:

```
ulazneGazda = Gazda.getRelationships(Direction.INCOMING);
```

Dohvaćanje izlaznih veza čvora Gazda:

```
izlazneGazda = Gazda.getRelationships(Direction.OUTGOING);
```

Dohvaćanje svih veza čvora Bazic:

```
sveBazic = Bazic.getRelationships();
```

Grafovske baze podataka korisne su u svim domenama gdje je potrebno implementirati mnoge veze među entitetima, poput društvenih mreža; često se koriste u upravljanju podacima

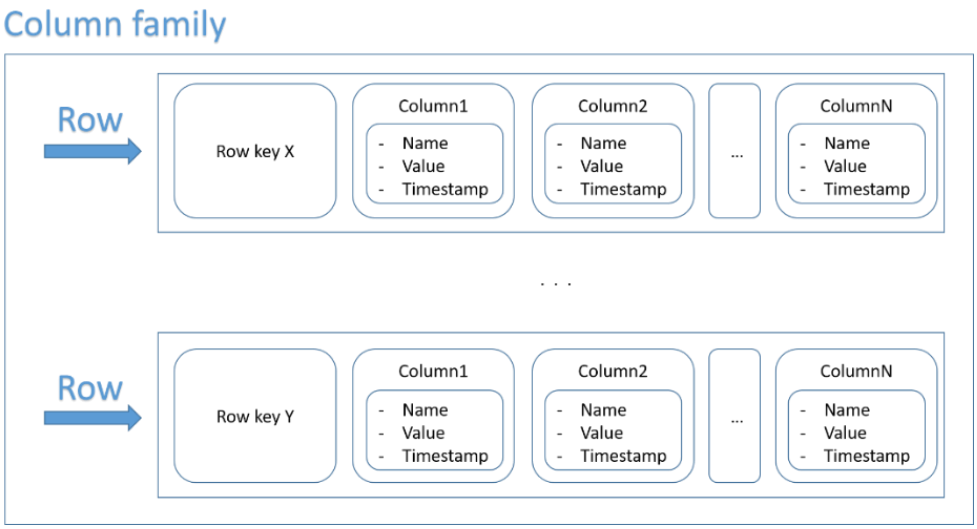
o otpremi ili prijevoznim rutama gdje je moguće prikazati udaljenosti lokacija; te u svrhe pružanja preporuka vezanih uz kupnje ili putovanja. Prema autorima Sadalage i Fowler (2013), posljedica korištenja grafičkih baza podataka u svrhu preporuka je rudarenje podataka i pronalaženje uzoraka, koji se mogu koristiti istim podacima kao i preporuke.

4.1.3. Stupčane baze podataka

Stupčane baze podataka su vrsta NoSQL baza podataka koje, iako nerelacijske, koriste svojevrstan tablični oblik zapisa podataka. Stupčane baze podataka pohranjuju podatke u familije stupaca na način da su zajedno pohranjene vrijednosti iz nekog stupca pri čemu je glavna prednost stupčanih baza podataka mogućnost pristupanja i dodavanja stupaca bez utjecaja na ostale. Stupčane baze podataka izuzetno su pogodne za upravljanje jako velikim količinama podataka jer prilikom pristupanja podacima i dohvaćanja istih nije potrebno pristupati cijeloj memoriji već se pristupa samo traženim stupcima. Za usporedbu, kada se koriste relacijske baze podataka, za svaki pristup podacima potrebno je pristupiti cijeloj zauzetoj memoriji. Stupčane baze podataka znatno optimiziraju čitanje podataka iz baze dok je moguće da će upis podataka trajati duže nego upis u relacijske baze, ali zbog toga su stupčane baze izuzetno pogodne za analitiku podataka.

Familija stupaca, koja sliči tablici u relacijskim bazama podataka, može sadržavati više redaka, kao što je prikazano na slici 12, a svaki od redaka sadrži jedinstveni identifikator sličan primarnim ključevima. Svaki redak sadrži određen broj stupaca koji se ne mora podudarati, a same je stupce vrlo jednostavno dodati u već definirane retke i familiju stupaca. Svaki pojedini stupac koji je dio određenog reda sadrži naziv kolone, vrijednost koja je upisana te vremensku oznaku koji pohranjuju vrijeme unosa podatka koja se ažurira prilikom izmjene podataka unutar stupca.

SUBP Cassandra jedan je od popularnih sustava za upravljanje stupčanim bazama podataka, a odlikuju ga brzina i visoka skalabilnost. Podaci u sustavu Cassandra strukturirani su kako je prikazano na slici 12, a osnovna jedinica pohrane podataka su stupci (Sadalage i Fowler, 2013).



Slika 13. Stupčana baza podataka (DataCamp, 10. lipnja 2024.)

U nastavku su prikazani primjeri upravljanja podacima u sustavu Cassandra.

Stupac u sustavu Cassandra:

```
{
name: "imeprezime",
value: "Sara Gazda",
timestamp: 0987654321
}
```

Stupac se sastoji od ključa *imeprezime* te vrijednosti *Sara Gazda*, a dodana mu je i vremenska oznaka (engl. *timestamp*) koji se koristi za situacije isteka podataka ili rješavanje zastarjelih podataka. Nadalje, kako je vidljivo na slici 12, red se sastoji od stupaca povezanih ključem reda (engl. *row key*), a kolekcija sličnih redova predstavlja familiju stupaca (engl. *column family*) koja je prikazana u nastavku.

```
{
"SaraGazda": {
      imeprezime: "Sara Gazda",
      bolnica: "Opca bolnica Bjelovar",
      struka: "pedijatrija"
      }
"MiroMiko": {
      imeprezime: "Miro Miko",
      bolnica: "KBC Zagreb"
      }
}
```

U prethodnom su primjeru stupci jednostavni pa se takva familija naziva standardna familija stupaca (engl. *standard column family*). Za složenije, super stupce (engl. *super colums*) može se reći da su spremnik stupaca, a sastoje se od imena i vrijednosti što je mapa stupaca (engl. *map of columns*). Super stupci mogu biti povezani u kolekciju super familija stupaca (engl. *super column family*) koja je prikazana u nastavku.

```
{
name: "lijecnik:SaraGazda",
value: {
      osobno: {
            name: "osobni-podaci",
            value:{
                  imeprezime: "Sara Gazda",
                  struka: "pedijatrija",
                  adresa: "Zagrebacka 13, Bjelovar"
                  }
              },
      bolnica: {
            name: "podaci-bolnica"
            value:{
                  naziv: "Opca bolnica Bjelovar‟,
                  odjel: "Odjel pedijatrije",
                  ambulanta: "amb.15-2"
                  }
                }
              }
name: "lijecnik:MiroMiko",
value: {
      osobno: {
            name: "osobni-podaci",
            value:{
                  imeprezime: "Miro Miko",
                  struka: "kirurgija",
                  adresa: "Vukovarska 2, Zagreb"
                  }
              },
      bolnica: {
            name: "podaci-bolnica"
            value:{
                  naziv: "KBC Zagreb‟,
                  odjel: "Traumatologija",
                  ambulanta: "amb.23-8"
                  }
                }
       }
}
```

Svojevrsna baza podataka u koju se u Cassandra sustavu pohranjuju familije stupaca, standardne i super, naziva se engl. *keyspace.* Kreiranje engl. keyspace s ciljem dodjeljivanja familija stupaca u „bazu podataka“ radi se naredbom *create keyspace*, npr. `create keyspace hzzo`. Jednostavnim operacijama poput GET, SET i DELETE moguće je upravljati podacima u familijama stupaca koje se nalaze u *keyspace hzzo*; prije čega je potrebno „postaviti“ se u *keyspace* u kojemu su pohranjeni podaci naredbom `use keyspace hzzo` (Sadalage i Fowler, 2013).

Kreiranje familije stupaca *lijecnik:*

```
CREATE COLUMN FAMILY lijecnik
WITH comparator = UTF8Type
AND key_validation_class=UTF8Type
AND column_metadata = [
```

```
{column_name: ime, validation_class: UTF8Type}
{column_name: struka, validation_class: UTF8Type}
{column_name: bolnica, validation_class: UTF8Type}
];
```

Kreirana je familija stupaca naziva *lijecnik* koja sadrži stupce ime, struka i bolnica, a korištenjem naredbe SET moguće je unijeti podatke o liječniku:

```
SET lijecnik ['Gazda'] ['ime'] = 'Sara Gazda';
SET lijecnik ['Gazda'] ['struka'] = 'pedijatrija';
SET lijecnik ['Miko'] ['bolnica'] = 'KBC Zagreb';
```

Brisanje stupca iz familije stupaca:

```
DEL lijecnik ['Gazda'] ['struka'];
```

Brisanje familije stupaca:

```
DEL lijecnik ['Miko'];
```

Naredbom GET moguće je prikazati atribute nekog reda, a s ciljem poboljšanja efikasnosti bolje je dohvaćati samo one stupce koji su potrebni umjesto cijele familije stupaca.

Dohvaćanje svih podataka o lijecniku:

```
GET lijecnik ['Gazda'];
```

Dohvaćanje jednog stupca iz familije stupaca:

```
GET lijecnik ['Gazda'] ['ime'];
```

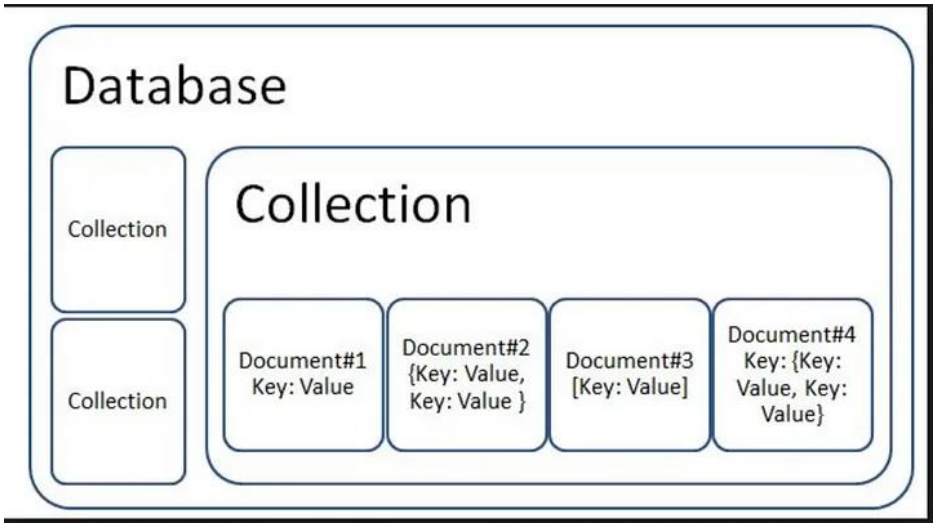
Spomenuto je da su stupčane baze podataka pogodne za analitiku podataka s obzirom da optimiziraju pretraživanje i ispis podataka, ali, prema autorima Sadalage i Fowler (2013), njihovo se korištenje ne preporuča kada je potrebno ispuniti ACID karakteristike transakcija.

4.1.4. Dokumentne baze podataka

Stojanović (2016) u svom osvrtu na NoSQL baze podataka navodi kako su dokumentne baze podataka najpopularnija vrsta nerelacijskih baza podataka što objašnjava intuitivnošću modela podataka koju pružaju dokumente baze podataka. U suštini, dokumentna baza podataka pohranjuje i dohvaća podatke u obliku tekstualnih dokumenata poput XML, JSON, BSON i slično. U paru ime – vrijednost, dokument je pohranjen unutar vrijednosti (engl. *value*) pa, prema autorima Sadalage i Fowler (2016), same dokumentne baze podataka možemo sagledati kao ključ – vrijednost pohranu podataka. Dokument je stoga, kao osnovni element dokumentnih baza podataka, uređen skup ključeva s pridruženim vrijednostima (Stojanović, 2016).

Unutar kolekcija u koje se dokumenti smještaju, svaki pojedini dokument može imati različitu strukturu iz čega proizlazi dinamička shema baze i predstavlja veliku prednost

dokumentnih baza u usporedbi s relacijskim bazama. Unatoč dinamičkoj shemi, za dokumente baze podataka može se reći da imaju sličnu strukturu relacijama u tradicionalnim bazama podataka pri čemu je kolekcija slična tablici, a redak tablice je upravo jedan dokument (Stojanović, 2016).



Slika 14. Dokumentna baza podataka (Medium, 11. lipnja 2024.)

Jedna od najvažnijih karakteristika dokumentnih baza podataka koje pruža dinamička shema jest fleksibilnost. Unutar jedne kolekcije može se nalaziti više dokumenata koji, kao što je prikazano na slici 13, ne moraju sadržavati iste atribute odnosno imati istu strukturu što je slučaj u relacijskim bazama gdje je unutar atributa za koji nije poznata vrijednost potrebno spremati NULL. Osim toga, dokumentne baze podataka mogu sadržavati dokumente koji su dio drugih dokumenata i tako podržavaju hijerarhijsku strukturu koja je, prema Stojanoviću (2016) od velike koristi u objektno – orijentiranom razvoju.

Dokumente baze podataka za cilj imaju visoku skalabilnost prilikom upravljanja s vrlo velikim količinama podataka, a prilikom upravljanja nerelacijskim bazama podataka u SUBP moguće je unositi podatke, pretraživati ih i ažurirati te brisati podatke što pruža dovoljnu fleksibilnost za specificiranje svih potrebnih upita (Stojanović, 2016). Među najpopularnijim SUBP za dokumente baze podatka je sustav MongoDB koji je korišten za primjer u nastavku rada.

5. SQL

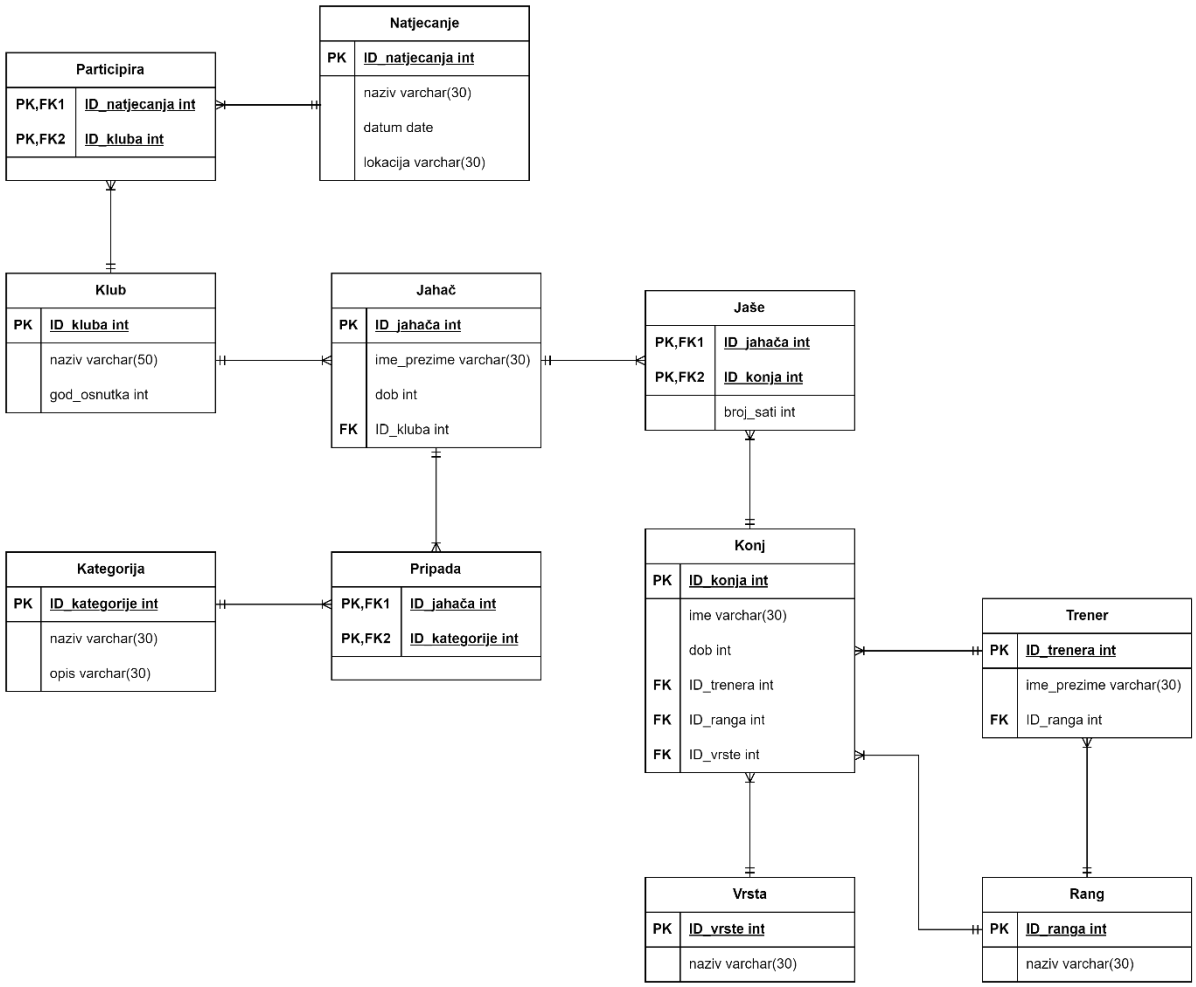
PostgreSQL je objektno – relacijski sustav za upravljanje bazama podataka koji koristi SQL jezik za upravljanje relacijskim bazama podataka. Početak razvoja PostgreSQL sustava datira u 1986. godini kada je na Sveučilištu Kalifornije u Berkeleyu razvijen kao dio POSTGRES projekta. Prema podacima sa Statista, gdje je prikazano rangiranje najpopularnijih sustava za upravljanje bazama podataka 2023. godine, PostgreSQL nalazi se na visokom četvrtom mjestu. Neki od glavnih razloga za visoku zastupljenost PostgreSQL-a u upravljaju bazama podataka su pouzdanost, integritet podataka, posvećenost *open source* zajednici te inovativna rješenja koja odlikuju ovaj SUBP. Na službenim stranicama PostgreSQL sustava navedeno je kako je osnovni cilj PostgreSQL-a biti usklađen sa SQL standardima te su stoga mnoge značajke zahtijevane SQL standardima podržane, a s vremenom se mogu očekivati novi koraci prema još značajnijoj usklađenosti. PostgreSQL navodi kako niti jedna relacijska baza podataka ne zadovoljava punu usklađenosti sa standardom SQL:2023 Core dok je sam PostgreSQL usklađen s najmanje 170 od 179 značajki ovog standarda.

5.1. Izrada primjera baze podataka u PostgreSQL

Za potrebe izrade primjera baze podataka u PosgtreSQL-u instaliran je sustav za upravljanje bazama podataka PostgreSQL te je za prikaz modela podataka napravljen ERA model u programu draw.io.

Za izradu primjera osmišljena je baza podataka Hrvatskog konjičkog saveza (HKS) za potrebe vođenja natjecanja Croatia Cup-a (CC) i Prvenstva Hrvatska (PH) koja su u stvari hrvatska endurance jahačka liga. U nastavku je prikazan ERA model koji je nastao analizom sljedećih poslovnih pravila: Na natjecanju participira više klubova, a jedan klub u određenom trenutku može sudjelovati na više natjecanja. Pojedini klub čini više jahača, dok jedan jahač može biti registriran u jednom klubu pri čemu pojedini jahač može pripadati više kategorija ovisno o nacionalnim i internacionalnim natjecanjima. Jedan jahač u sezoni može jahati više konja te jednog konja može jahati više jahača, a konju je određena jedna vrsta kojoj može pripadati više konja. Uz konje je vezan i trener koji može trenirati više konja dok jedan konj ima jednog trenera. Sukladno pravilniku Međunarodne konjičke federacije [FEI], konj i trener pripadaju jednom rangu, pri čemu ne moraju biti u istom rangu, a jednom rangu može pripadati više trenera i konja.

Analizom poslovnih pravila modeliran je ERA model koji se sastoji od jedanaest relacija, pri čemu su *Participira*, *Pripada* i *Jaše* slabi entiteti koji su potrebni za pravilnu implementaciju veza više naprema više (M:N) iz poslovnih pravila.



Slika 15. ERA model za HKS (autorski rad)

5.1.1. Implementacija ERA modela

U SUBP PostgreSQL prvo je kreirana baza podataka Hrvatskog konjičkog saveza (HKS) u kojoj će se potom kreirati sve tablice i upravljati unesenim podacima.

```
CREATE DATABASE hks;
\c hks;
```

U bazi podataka *hks* potom su kreirane relacije prikazane ERA modelom. Prilikom kreiranja relacija, odnosno tablica unutar baze, prvotno su kreirane tablice koje ne sadrže vanjske ključeve poput *Natjecanje* i *Klub*; zatim su kreirane tablice koje sadrže vanjske ključeve poput *Trener* i *Jahač*; te su na kraju kreirane tablice sa složenim primarnim ključevima za

implementiranje veze više naprema više (M:N) poput *Jaše*. U nastavku su prikazane pripadajuće naredbe za kreiranje pojedine tablice.

Kreiranje tablice *natjecanje*:

```
CREATE TABLE natjecanje(
id_natjecanja INTEGER PRIMARY KEY,
naziv VARCHAR(30),
datum DATE,
lokacija VARCHAR(30));
```

Kreiranje tablice *klub*:

```
CREATE TABLE klub(
id_kluba INTEGER PRIMARY KEY,
naziv VARCHAR(50),
god_osnutka INTEGER);
```

Kreiranje tablice *kategorija*:

```
CREATE TABLE kategorija(
id_kategorije INTEGER PRIMARY KEY,
naziv VARCHAR(30),
opis VARCHAR(30));
```

Kreiranje tablice *vrsta*:

```
CREATE TABLE vrsta(
id_vrste INTEGER PRIMARY KEY,
naziv VARCHAR(30));
```

Kreiranje tablice *rang*:

```
CREATE TABLE rang(
id_ranga INTEGER PRIMARY KEY,
naziv VARCHAR(30));
```

Nakon kreiranja tablica bez vanjskih ključeva, moguće je kreirati tablice koje sadrže vanjske ključeve s obzirom da je moguće definirati na što se vanjski ključ odnosi.

Kreiranje tablice *jahac*:

```
CREATE TABLE jahac(
id_jahaca INTEGER PRIMARY KEY,
ime_prezime VARCHAR(30),
dob INTEGER,
id_kluba INTEGER REFERENCES klub(id_kluba)
ON UPDATE CASCADE ON DELETE SET NULL);
```

Kreiranje tablice *trener*:

```
CREATE TABLE trener(
id_trenera INTEGER PRIMARY KEY,
ime_prezime VARCHAR(30),
id_ranga INTEGER REFERENCES rang(id_ranga)
ON UPDATE CASCADE ON DELETE SET NULL);
```

Kreiranje tablice *konj*:

```
CREATE TABLE konj(
id_konja INTEGER PRIMARY KEY,
ime VARCHAR(30),
dob INTEGER,
```

```
id_tenera INTEGER REFERENCES trener(id_trenera)
ON UPDATE CASCADE ON DELETE SET NULL,
id_ranga INTEGER REFERENCES rang(id_ranga)
ON UPDATE CASCADE ON DELETE SET NULL,
id_vrste INTEGER REFERENCES vrsta(id_vrste)
ON UPDATE CASCADE ON DELETE SET NULL);
```

Pošto su kreirane sve „jednostavnije“ relacije, moguće je kreirati slabe entitete i tako implementirati veze (M:N).

Kreiranje tablice *participira*:

```
CREATE TABLE participira(
id_natjecanja INTEGER REFERENCES natjecanje(id_natjecanja),
id_kluba INTEGER REFERENCES klub(id_kluba),
PRIMARY KEY(id_natjecanja, id_kluba));
```

Kreiranje tablice *pripada*:

```
CREATE TABLE pripada(
id_jahaca INTEGER REFERENCES jahac(id_jahaca),
id_kategorije INTEGER REFERENCES kategorija(id_kategorije),
PRIMARY KEY(id_jahaca, id_kategorije));
```

Kreiranje tablice *jase*:

```
CREATE TABLE jase(
id_jahaca INTEGER REFERENCES jahac(id_jahaca),
id_konja INTEGER REFERENCES konj(id_konja),
broj_sati INTEGER,
PRIMARY KEY(id_jahaca, id_konja));
```

5.1.2. Unos podataka u bazu podataka Hrvatskog konjičkog saveza

Kada su kreirane sve tablice unutar baze podataka Hrvatskog konjičkog saveza [HKS], moguće je kreirane tablice popuniti pripadajućim podacima za vođenje natjecanja Croatia Cup-a i Prvenstva Hrvatske.

Za primjer unosa podataka prikazan je unos podataka u četiri tablice i struktura istih, a unutar baze u PostgreSQL sustavu podacima su popunjene i sve preostale tablice.

Struktura tablice *natjecanje*:

```
SELECT * FROM natjecanje;

id_natjecanja | naziv | datum | lokacija
--------------+-------+-------+----------
(0 rows)
```

Unos podataka u tablicu *natjecanje*:

```
INSERT INTO natjecanje VALUES
(1, '1. kolo CC', '12-03-2024', 'SCR Kukavica'),
(2, '2. kolo CC', '24-04-2024', 'Barban'),
(3, 'Prvenstvo Hrvatske', '15-05-2024', 'Gradec'),
(4, '3. kolo CC', '17-06-2024', 'Vukosavljevica');
```

```
id_natjecanja |       naziv         |   datum    |    lokacija
--------------+---------------------+------------+----------------
            1 | 1. kolo CC          | 2024-03-12 | SCR Kukavica
            2 | 2. kolo CC          | 2024-04-24 | Barban
            3 | Prvenstvo Hrvatske | 2024-05-15 | Gradec
            4 | 3. kolo CC          | 2024-06-17 | Vukosavljevica
(4 rows)
```

Unos podataka i struktura tablice *klub*:

```
INSERT INTO klub VALUES
(1, 'KK Arion', 2023), (2, 'KK Istra', 2006),
(3, 'KK Vinia', 2015), (4, 'KK Moslavina', 2010);

id_kluba  |    naziv     | god_osnutka
----------+--------------+-------------
        1 | KK Arion     |        2023
        2 | KK Istra     |        2006
        3 | KK Vinia     |        2015
        4 | KK Moslavina |        2010
(4 rows)
```

Unos podataka i struktura tablice *jahac*:

```
INSERT INTO jahac VALUES
(1, 'Sara Gazda', 22, 1), (2, 'Ivana Ivic', 18, 1),
(3, 'Petra Peric', 14, 2), (4, 'Marko Markic', 28, 3),
(5, 'Lea Leic', 24, 3), (6,'Zora Zoric', 12, 3),
(7, 'Tara Taric', 20, 4), (8, 'Jan Janic', 20, 4),
(9, 'Sven Svenic', 31, 4),(10, 'Ivan Ivanic', 26, 1);

id_jahaca  | ime_prezime  | dob | id_kluba
-----------+--------------+-----+----------
         1 | Sara Gazda   |  22 |        1
         2 | Ivana Ivic   |  18 |        1
         3 | Petra Peric  |  14 |        2
         4 | Marko Markic |  28 |        3
         5 | Lea Leic     |  24 |        3
         6 | Zora Zoric   |  12 |        3
         7 | Tara Taric   |  20 |        4
         8 | Jan Janic    |  20 |        4
         9 | Sven Svenic  |  31 |        4
        10 | Ivan Ivanic  |  26 |        1
(10 rows)
```

S obzirom da je tablica *participira* slabi entitet kojim se implementira veza više naprema više među relacijama *natjecanje* i *klub*, sam prikaz strukture tablice *participira* neće nam puno toga „reći“.

Unos podataka u tablicu *participira*:

```
INSERT INTO participira VALUES
(1, 1), (1, 2), (1, 3), (2, 3), (2,4), (4,1), (4,3);
```

Iz unesenih podataka možemo zaključiti, na primjer, da na natjecanju s ID-em jedan sudjeluju tri kluba, a intuitivniji ispis podataka za analizu moguće je prikazati kreiranjem upita.

5.1.3. Jednostavni upiti u bazi podataka Hrvatskog konjičkog saveza

Nakon što su uneseni pripadajući podaci u sve tablice baze podataka *hks*, nad njima je moguće kreirati upite koji služe dohvaćanju određenih redova, zapisa, iz relacija najčešće uz smislene uvjete. Kada se govori o upitima, razlikujemo jednostavne upite vezane uz jednu tablicu te složene upite koji se kreiraju za prikaz podataka iz dvije ili više tablica.

Na primjer, u tablici *rang* uneseni su CEI i FEI rangovi koje treneri i konji mogu postizati, a recimo da postoji potreba prikaza samo europskih, CEI, rangova koji postoje – navedeno je moguće realizirati sljedećim upitom:

```
SELECT id_ranga AS "ID ranga", naziv AS "Europski rangovi"
FROM rang WHERE naziv LIKE 'CEI%';

ID ranga  | Europski rangovi
----------+-----------------
        1 | CEI1*
        2 | CEI2*
        3 | CEI3*
(3 rows)
```

Nadalje, pretpostavimo da je za organizaciju odlaska na međunarodne utrke potrebno znati raspored utrka u ožujku i travnju 2024. godine što se može napraviti sljedećim upitom:

```
SELECT id_natjecanja AS "ID natjecanja",
naziv AS "Naziv natjecanja",
datum AS "Datum natjecanja",
lokacija AS "Lokacija natjecanja"
FROM natjecanje
WHERE EXTRACT(MONTH FROM datum) IN (3, 4);
```

```
ID natjecanja  | Naziv natjecanja | Datum natjecanja | Lokacija natjecanja
---------------+------------------+------------------+---------------------
             1 | 1. kolo CC       | 2024-03-12       | SCR Kukavica
             2 | 2. kolo CC       | 2024-04-24       | Barban
(2 rows)
```

Do sada su prikazana dva jednostavna upita temeljem postavljanja uvjeta nad jednom tablicom, a u nastavku su primjeri složenih upita nad dvije ili više tablica.

5.1.4. Složeni upiti u bazi podataka Hrvatskog konjičkog saveza

Prilikom kreiranja složenih upita do izražaja dolaze veze među relacijama prikazane ERA modelom koje su implementirane primarnim i vanjskim ključevima – upravo se oni koriste za povezivanje tablica prilikom ispisa podataka složenim upitima.

Tablica *konj* sadrži tri vanjska ključa, konkretno se referencira na tablice *trener*, *rang* i *vrsta*. Osim toga, s obzirom da jedan jahač može jahati više konja i jednog konja jaše više jahača, tablica *konj* prisutna je i u vezi (M:N) s tablicom *jahac*. Prema navedenom može se

zaključiti da sam ispis tablice *konj* ne daje potpuni prikaz podataka, odnosno prikazuje puno primarnih ključeva (ID-eva), te je za kvalitetnu analizu podataka o konjima potrebno kreirati upit koji povezuje tablicu *konj* i ostale tablice s kojima je u vezi. Sljedeći upit omogućio je pregledniji i iscrpniji ispis podataka o konjima:

```
SELECT k.id_konja AS "ID konja",
k.ime AS "Ime konja", k.dob AS "dob konja",
vr.naziv AS "Vrsta konja", r.naziv AS "Rang konja",
j.ime_prezime AS "Ime i prezime jahaca",
jas.broj_sati AS "Broj sati tjedno",
t.ime_prezime AS "Ime i prezime trenera"
FROM konj k
JOIN jase jas ON k.id_konja = jas.id_konja
JOIN jahac j ON jas.id_jahaca = j.id_jahaca
JOIN trener t ON k.id_tenera = t.id_trenera
JOIN rang r ON k.id_ranga = r.id_ranga
JOIN vrsta vr ON k.id_vrste = vr.id_vrste
ORDER BY k.dob;
```

```
ID konja  | Ime konja | dob konja | Vrsta konja | Rang konja | Ime i prezime jahaca | Broj sati tjedno | Ime i prezime trenera
----------+-----------+-----------+-------------+------------+----------------------+------------------+----------------------
        3 | Jantar    |         7 | Lipicanac   | CEI3*      | Lea Leic             |                2 | Robi Robic
        2 | Hungaro   |        10 | Haflinger   | FEI2*      | Marko Markic         |                3 | Jakov Jakic
        2 | Hungaro   |        10 | Haflinger   | FEI2*      | Ivan Ivanic          |                1 | Jakov Jakic
        4 | Quitto    |        11 | Arap        | CEI2*      | Sara Gazda           |                2 | Jakov Jakic
        4 | Quitto    |        11 | Arap        | CEI2*      | Jan Janic            |                1 | Jakov Jakic
        1 | Alka      |        14 | Gidran      | CEI2*      | Sara Gazda           |                5 | Luka Lukic
        1 | Alka      |        14 | Gidran      | CEI2*      | Sven Svenic          |                3 | Luka Lukic
(7 rows)
```

Tablica *kategorija* sadrži kategorije kojima jahači mogu pripadati, a s obzirom da se kategorije razlikuju na europskoj i svjetskoj razini, pojedini jahač može pripadati više kategorija. Pretpostavimo da je za formiranje hrvatske endurance reprezentacije na nadolazećim Olimpijskim igrama [OI] potreban pregled svih jahača koji pripadaju i europskim i svjetskim kategorijama, što je uvjet sudjelovanja na OI. Takav pregled pružen je sljedećim upitom:

```
SELECT j.id_jahaca AS "ID jahaca", j.ime_prezime AS "Ime i prezime",
MAX(CASE WHEN k.naziv LIKE 'CEI%' THEN k.naziv
|| ', ' || k.opis ELSE NULL END) AS "Kategorija Europa",
MAX(CASE WHEN k.naziv LIKE 'FEI%' THEN k.naziv
|| ', ' || k.opis ELSE NULL END) AS "Kategorija svijet"
FROM jahac j
JOIN pripada p
ON j.id_jahaca = p.id_jahaca
JOIN kategorija k
ON p.id_kategorije = k.id_kategorije
WHERE j.id_jahaca IN (
        SELECT id_jahaca FROM pripada
        WHERE id_kategorije IN (
            SELECT id_kategorije FROM kategorija
            WHERE naziv LIKE 'CEI%')
        INTERSECT
        SELECT id_jahaca FROM pripada
        WHERE id_kategorije IN (
            SELECT id_kategorije FROM kategorija
            WHERE naziv LIKE 'FEI%'))
```

```
      GROUP BY j.id_jahaca, j.ime_prezime;

ID jahaca | Ime i prezime |    Kategorija Europa    |    Kategorija svijet
----------+---------------+-------------------------+---------------------
        2 | Ivana Ivic    | CEI-SR, europski senior | FEI-JR, svjetski junior
        3 | Petra Peric   | CEI-JR, europski junior | FEI-KD, svjetski kadet
        4 | Marko Markic  | CEI-SR, europski senior | FEI-SR, svjetski senior
        6 | Zora Zoric    | CEI-JR, europski junior | FEI-KD, svjetski kadet
        9 | Sven Svenic   | CEI-SR, europski senior | FEI-SR, svjetski senior
       10 | Ivan Ivanic   | CEI-SR, europski senior | FEI-SR, svjetski senior
(6 rows)
```

5.1.5. Funkcija u bazi podataka Hrvatskog konjičkog saveza

Za svrhu kreiranja primjera funkcije, recimo da je za analizu opterećenosti konja potrebno jednostavno saznati broj jahača koji jašu konja te koliko sati tjedno treniraju. Kako bi omogućili takav ispis podataka za svakog konja po potrebi, kreirana je sljedeća funkcija:

```
CREATE OR REPLACE FUNCTION treninzi(VARCHAR)
RETURNS TABLE ("Ime jahaca" VARCHAR, "Broj sati tjedno" INTEGER)
AS
    'SELECT j.ime_prezime, ja.broj_sati
    FROM konj k
    JOIN jase ja ON k.id_konja = ja.id_konja
    JOIN jahac j ON ja.id_jahaca = j.id_jahaca
    WHERE k.ime = $1;'
LANGUAGE SQL;
```

Kada je funkcija uspješno kreirana, sustav ispisuje poruku CREATE FUNCTION kao potvrdu, a za samo korištenje funkcije istu je potrebno pozvati. S obzirom da je funkcija kreirana na način da vraća tablicu kako bi se osigurao povrat skupova redova iz tablice, moguće je funkciju pozvati na dva načina. Prvi način je klasičan poziv funkcije pri čemu rezultat nije strukturiran u obliku tablice koju funkcija vraća već samo kao rezultat funkcije *treninzi:*

```
SELECT treninzi ('Alka');

     treninzi
-------------------
 ("Sara Gazda",5)
 ("Sven Svenic",3)
(2 rows)
```

Drugi način poziva iste funkcije je jednak ispisu tablica u sustavu, a to je moguće jer sama funkcija vraća tablicu. Prilikom ovakvog poziva, rezultati funkcije strukturirani su u obliku tablice s nazivima stupaca kako je definirano u funkciji:

```
SELECT * FROM treninzi ('Hungaro');

  Ime jahaca  | Broj sati tjedno
--------------+-----------------
 Marko Markic |               3
 Ivan Ivanic  |               1
```

5.1.6. Okidač u bazi podataka Hrvatskog konjičkog saveza

Implementacijom i upotrebom okidača u SUBP povećavaju se performanse baza podataka; smanjuje se mogućnost pogreške i raste efikasnost. Jedan od dobrih primjera mogućnosti smanjenja pogrešaka korištenjem okidača je implementacija referencijalnog integriteta o kojemu je već bilo govora (Rabuzin, 2014). Primjer provjere odnosno osiguranja referencijalnog integriteta kreiranjem okidača dan je u nastavku.

Kako je vidljivo u ERA modelu, entiteti *klub* i *jahac* povezani su vezom jedan naprema više (1:M) s obzirom da jednom klubu može pripadati više jahača, a jahač pripada jednom klubu. Navedeno je u bazi podataka *hks* implementirano dodavanjem vanjskog ključa *id_kluba* u relaciju *jahac*. Pravilnik Hrvatskog konjičkog saveza nalaže da jahač mora pripadati jednom od klubova u članstvu HKS-a kako bi se mogao natjecati u CC-u ili na PH, a upravo se u tome ogleda prilika za implementiranje okidača koji osigurava referencijalni integritet. Prilikom upisivanja novog jahača u bazu podataka ili ažuriranja podataka o jahaču, potrebno je provjeriti odgovara li unesena vrijednost vanjskog ključa, *id_kluba*, nekoj od vrijednosti stupca primarnog ključa tablice *klub*. Kako bi kreirali okidač, prvo je potrebno kreirati funkciju koja će se izvršiti kao reakcija na događaj unosa novog jahača ili ažuriranja podataka o postojećem jahaču, a zatim odgovarajući okidač koji je povratni tip funkcije. Kreirana funkcija u jeziku PL/pgSQL i okidač nalaze se u nastavku (Rabuzin, 2014).

```
CREATE OR REPLACE FUNCTION pripadnost_klubu() RETURNS TRIGGER
AS $$
DECLARE red klub%ROWTYPE;
BEGIN
      SELECT * INTO red FROM klub WHERE klub.id_kluba=new.id_kluba;
IF NOT FOUND THEN
      RAISE EXCEPTION 'Ne postoji klub cija je vrijednost primarnog
      kljuca id_kluba=%! Potrebno je unesti identifikator postojeceg
      kluba!', new.id_kluba;
END IF;
RETURN NEW; END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER unos_azuriranje_jahaca
BEFORE INSERT OR UPDATE ON jahac
FOR EACH ROW WHEN (new.id_kluba IS NOT NULL)
EXECUTE PROCEDURE pripadnost_klubu();
```

Kako bi testirali ispravnost kreiranog okidača za osiguranje ispunjenja referencijalnog integriteta, namjerno su uneseni i ažurirani podaci na način da krše referencijalni integritet. Prikaz klubova u bazi podataka *hks*:

```
id_kluba  |    naziv     | god_osnutka
----------+--------------+-------------
        1 | KK Arion     |        2023
        2 | KK Istra     |        2006
        3 | KK Vinia     |        2015
```

```
        4 | KK Moslavina |        2010
(4 rows)
```

Ažuriranje jahača Sara Gazda s postavljanjem vanjskog ključa na nepostojeći klub:

```
UPDATE jahac SET id_kluba=7 WHERE ime_prezime='Sara Gazda';

ERROR:  Ne postoji klub cija je vrijednost primarnog kljuca id_kluba=7!
Potrebno je unesti identifikator postojeceg kluba!
```

Unos novog jahača s pravilnim vanjskim ključem:

```
INSERT INTO jahac VALUES (12, 'Erik Eric', 24, 4);

INSERT 0 1

id_jahaca | ime_prezime  | dob | id_kluba
-----------+--------------+-----+----------
         1 | Sara Gazda   |  22 |        1
         2 | Ivana Ivic   |  18 |        1
         3 | Petra Peric  |  14 |        2
         4 | Marko Markic |  28 |        3
         5 | Lea Leic     |  24 |        3
         6 | Zora Zoric   |  12 |        3
         7 | Tara Taric   |  20 |        4
         8 | Jan Janic    |  20 |        4
         9 | Sven Svenic  |  31 |        4
        10 | Ivan Ivanic  |  26 |        1
        11 | Ivo Ivic     |  18 |        4
        12 | Erik Eric    |  24 |        4
(12 rows)
```

Unos novog jahača s nepravilnim vanjskim ključem:

```
INSERT INTO jahac VALUES (13, 'Borna Boric', 14, 8);

ERROR:  Ne postoji klub cija je vrijednost primarnog kljuca id_kluba=8!
Potrebno je unesti identifikator postojeceg kluba!
```

Kreiranjem ERA modela za Hrvatski konjički savez s ciljem vođenja baze podataka za Croatia Cup i Prvenstvo Hrvatske – hrvatsku endurance ligu, prikazan je praktični primjer korištenja sustava za upravljanje bazama podataka PostgreSQL. Kroz modeliranje baze podataka, implementiranje tablica i unos podataka stvorena je baza podataka *hks* temeljem koje su prikazani primjeri jednostavnih i složenih upita, funkcije i okidača koji su usko vezani uz stvarne primjere i zahtjeve Hrvatskog konjičkog saveza.

6. MongoDB

MongoDB je sustav za upravljanje nerelacijskim bazama podataka, konkretno dokument bazama podataka. Osnivanje tvrtke MongoDB datira u 2007. godinu kada su Dwight Merriman, Eliot Horowitz i Kevin Ryan primijetili ograničenja relacijskih baza podataka u pogledu skalabilnosti. Naziv MongoDB dolazi od riječi *humongous,* značenja ogroman ili iznimno velik, a upravljanje iznimno velikim količinama podataka u današnje vrijeme velikih podataka (engl. *Big Data*) upravo je glavna karakteristika sustava MongoDB. Prema podacima sa Statisa, gdje je prikazano rangiranje najpopularnijih sustava za upravljanje bazama podataka 2023. godine, MongoDB nalazi se na petom mjestu pri čemu je prvi u rangu SUBP za nerelacijske baze podataka. Ovako visok rang i popularnost MongoDB-a zasniva se na visokim performansama, dostupnosti, fleksibilnosti i skalabilnosti koje pruža sustav MongoDB u radu s NoSQL bazama podataka.

6.1. Izrada primjera baze podataka u MongoDB

Za potrebe izrade primjera baze podataka u MongoDB sustavu za upravljanje bazama podataka izvršena je instalacija sustava. Instaliran je MongoDB Shell te MongoDB Compass, grafičko korisničko sučelje u kojem je također moguće upravljati bazama podataka. Kako je i navedeno na MongoDB službenim stranicama, MongoDB Shell je proizvod otvorenog koda (engl. *open source*) te predstavlja najbrži i najjednostavniji način za upravljanje podacima u MongoDB SUBP. Kako bi se MongoDB Shell mogao koristiti, putanja izvršnog MongoDB Shell-a, mongosh.exe, postavljena je u varijablama okruženja (engl. *Environment Variables*) te je sučelje MongoDB dostupno za korištenje. Poradi grafički intuitivnijeg korištenja MongoDB Shell-a, isti je putem ekstenzija (engl. *Extensions*) dodan u uređivač koda (engl. *code editor*) VSCode. Unutar terminala (engl. *terminal*) u VSCode upravljat će se bazama podataka u MongoDB Shell-u za kreiranje primjera, a dodatno će biti prikazane i mogućnosti upravljanja podacima u grafičkom sučelju – MongoDB Compass.

Modeliranje baze podataka za PostgreSQL prikazano je ERA modelom, ali modeliranje baze podataka modelom entiteta i veza (engl. *Entity - Relationship model*) nije moguće za dizajn NoSQL baza podataka. Sam zapis ER modela sugerira da će se baza podataka implementirati u SUBP koji pohranjuje podatke u tablice zbog čega ne postoji pravilan način izražavanja pohrane nizova i ugniježđenih dokumenata koje NoSQL baze omogućuju. Osim toga, kao što će kasnije biti prikazano, MongoDB omogućuje izravnu agregaciju nizova i ugniježđenih struktura podataka bez korištenja primarnih i vanjskih ključeva. S obzirom da korištenjem E – R modela za modeliranje baze nije moguće ispravno prikazati strukture

podataka koje su međusobno povezane bez korištenja ključeva, E – R model ne koristi se za modeliranje NoSQL baza podataka (Hills, 2016).

Modeliranje baze za MongoDB, kako navode i na službenim stranicama sustava, nema striktno određena pravila s obzirom na to da se radi o fleksibilnoj shemi i prema tome shemu treba kreirati prema zahtjevima aplikacije za koju se modelira baza. Iako se radi o fleksibilnoj shemi, tipovi veza među podacima još uvijek postoje te ih je moguće modelirati za MongoDB.

Veza jedan naprema jedan (1:1) najjednostavnija je veza za prikaz u shemi i u sustavu MongoDB implementira se pridruživanjem jedne vrijednosti jednom imenu u paru *ime:vrijednost* od kojih se sastoje dokumenti unutar kolekcija. Primjer dokumenta s vezama (1:1) u MongoDB prikazan je u nastavku.

```
{
_id: 1
naziv: "KK Arion"
osnutak: 2023
}
```

Veza jedan naprema više (1:M) u sustavu MongoDB jednostavno se implementira upisom više vrijednosti u par *ime:vrijednost,* u obliku niza ili niza objekta, koje mogu ili ne moraju odgovarati vrijednostima u drugoj kolekciji kao u sljedećem primjeru.

```
konj {
      _id: 1
      ime: "Gidran-121 Alka"
      ozdrijebljen: 2009-01-21T00:00:00.000+00:00
      ocjena: 4.8
      potkovan: true

      stala: Object
            naziv: "Ergela Piber"
            adresa: "Piber 1, Koflach, Austrija"
            boks: 12

      jahaci: Array (3)
            0: "Sara Gazda"
            1: "Ivo Ivic"
            2: "Pero Peric"
      Vrsta: "Gidran"
      }

jahac {
      _id: ObjectId('667882af357761ca9c90defe')
      ime: "Sara Gazda"
      dob: 22
      kategorija: "senior"
      klub: "KK Arion"
      }
```

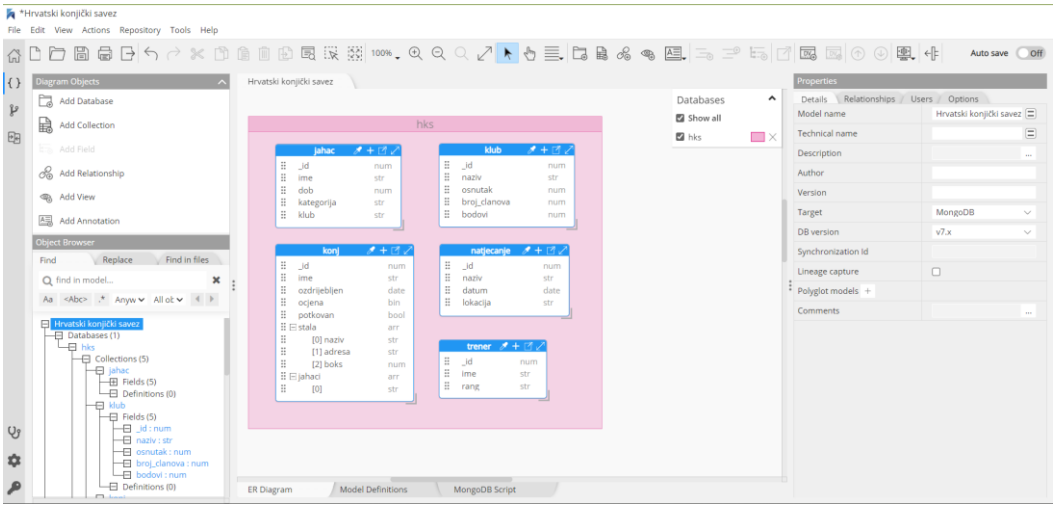
U ovom primjeru veze (1:M) imena jahača u kolekciji *konj* odgovaraju vrijednostima dokumenata u kolekciji *jahac* dok su vrijednosti uz atribut stala unesene samo u dokumentu o konju Alka u kolekciji *konj*.

Veza više naprema više (M:N) u MongoDB može se implementirati tako da u vrijednost para *ime:vrijednost* jednog dokumenta upišu vrijednosti koje odgovaraju vrijednostima dokumenta iz druge kolekcije te se isto napravi u dokumentu druge kolekcije. Primjer implementacije takve veze prikazan je u nastavku.

```
natjecanje {
      _id: 1
      naziv: "1. kolo CC"
      datum: 2024-03-12T00:00:00.000+00:00
      lokacija: "SRC Kukavica"
      prijavljeni: Array (3)
            0:  "KK Arion"
            1: "KK Istra"
            2: "KK Moslavina"
      }
klub {
      _id: 1
      naziv: "KK Arion"
      osnutak: 2023
      natjecanja: Array(2)
            0: "1. kolo CC"
            1: "Prvenstvo Hrvatske"
       }
```

S obzirom da na jednom natjecanju može sudjelovati više timova, u dokumentu natjecanja 1. kola Croatia Cupa uneseni su svi prijavljeni klubovi. Nadalje, kako se svaki klub može prijaviti na više natjecanja, u dokumentu kluba KK Arion upisana su dva natjecanja na koja je klub prijavljen.

S ciljem optimiziranja modeliranja podataka i dizajna sheme za MongoDB, na službenim stranicama sustava predložen je program Hackolade koji se može besplatno koristiti online. Program Hackolade nudi kreiranje sheme podataka za različite formate pohrane, REST API-jeve, JSON u RDBMS-u i NoSQL SUBP među kojima je i MongoDB. Izuzetno korisna funkcija programa je mogućnost uvoza (engl. *import*) datoteke u kojoj je implementiran model podataka u SQL jeziku iz sustava poput PostgreSQL ili Oracle. Nakon uvoza modela podataka i prikaza tradicionalnog ERA modela moguće je detaljnim procesom denormalizacije, kojim se više tablica kombinira u jednu, kreirati model podataka za MongoDB. Za primjer korištenja programa Hackolade ručno je kreiran model podataka za bazu Hrvatskog konjičkog saveza u sustavu MongoDB koji je prikazan na slici 16.



Slika 16. Modeliranje baze hks u programu Hackolade (autorski rad)

6.1.1. Kreiranje korisnika

Sustav MongoDB omogućuje autentikaciju i autorizaciju korisnika koje služe potvrđivanju identiteta korisnika i određivanju korisnikovog pristupa resursima i operacijama te tako pridonose sigurnosti baze podataka. Prema službenim stranicama MongoDB-a, autentikacija se provodi omogućavanjem kontrole pristupa za samostalnu MongoDB instancu što je moguće konfiguriranjem MongoDB-a s različitim autentikacijama, korištenjem SCRAM-a ili certifikata koji su ponuđeni na službenim stranicama. Sljedeći korak nakon omogućavanja kontrole pristupa je stvaranje dodatnih korisnika, upravljanje korisnicima i dodjeljivanje uloga.

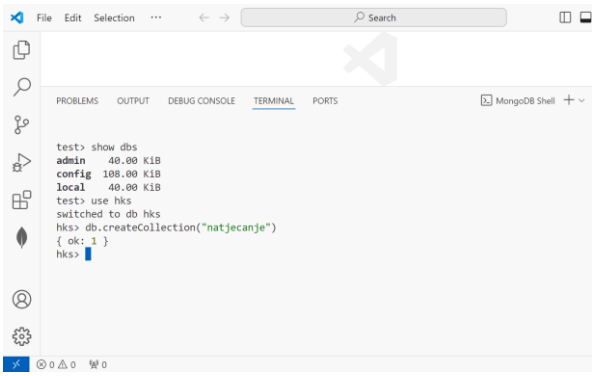
Kada je uključena kontrola pristupa, sustav MongoDB od korisnika zahtjeva identificiranje te je pojedinim korisnicima dodijeljena jedna ili više uloga koje daju privilegije za izvođenje određene radnji u MongoDB sustavu. Naredbom *db.createUser()* moguće je kreirati korisnika, a cijela sintaksa naredbe dana je u nastavku.

```
db.createUser(
      {
        user: "<name>",
        pwd: passwordPrompt(),      // Or  "<cleartext password>"
        customData: { <any information> },
        roles: [
          { role: "<role>", db: "<database>" } | "<role>",
          ...
        ],
        authenticationRestrictions: [
            {
              clientSource: ["<IP>" | "<CIDR range>", ...],
              serverAddress: ["<IP>" | "<CIDR range>", ...]
            },
            ...
        ]
      })
```

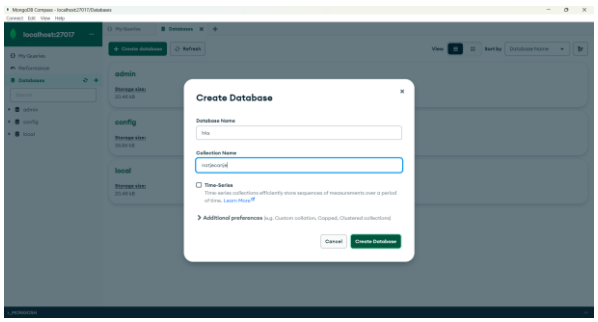
Prilikom kreiranja korisnika potrebno je novom korisniku dodijeliti ime i lozinku, a uz *customData* moguće je upisati bilo koje dodatne informacije važne za korisnika. Korisniku je potrebno dodijeliti uloge koje definiraju privilegije za izvođenje određenih radnji. Ukoliko se korisniku dodjeljuje uloga za istu bazu u kojoj se korisnik kreira, dovoljno je navesti samo naziv uloge; u protivnom je potrebno uz *db* navesti za koju bazu se dodjeljuje određena uloga. Svaka baza podataka u sustavu MongoDB uključuje uloge poput korisničkih uloga *read* i *readWrite* te admin uloga *dbAdmin*, *dbOwner* i *userAdmin*, a na službenim je stranicama uz svaku ulogu definirano koje su akcije omogućene dodjeljivanjem određene uloge.

6.1.2. Kreiranje baze podataka za Hrvatski konjički savez

Za početak rada u sustavu MongoDB, prikazan je popis baza podataka naredbom *show db* te je naredbom *use hks* kreirana baza podataka za Hrvatski konjički savez u ovom SUBP. Ukoliko se nakon kreiranja baze *hks* ponovno ispiše popis baza podataka, sama baza *hks* neće se prikazati iz razloga što je prazna. Baza hks prikazati će se kada se u nju doda kolekcija s pripadajućim dokumentima što se može ostvariti eksplicitnom naredbom *db.createCollection()* ili na implicitan način (*db.naziv_kolekcije.insert ({dokument})*) gdje se navodi naziv kolekcije i unose se dokumenti pri čemu se, u slučaju nepostojanja kolekcije, ista kreira. U nastavku je prikazan ispis baza podataka, kreiranje nove baze *hks* te dodavanje kolekcije *natjecanje* u bazu MongoDB Shell-u i MongoDB Compass-u.



Slika 17. Kreiranje baze MongoDB Shell (autorski rad)



Slika 18. Kreiranje baze MongoDB Compass (autorski rad)

6.1.3. Unos podataka u bazu podataka Hrvatskog konjičkog saveza

MongoDB je sustav za upravljanje dokumentnim bazama podataka te se u njemu baze podataka sastoje od kolekcija, slično relacijama, a kolekcije se sastoje od dokumenata – sličnih redovima u relacijama. Za unos podataka u bazu *hks*, odnosno dokumenata u kolekciju *natjecanje*, koriste se naredbe *insertOne ili insertMany*; unos dokumenata prikazan je u nastavku.

```
hks> db.natjecanje.insertOne({_id:1, naziv: "1. kolo CC", datum: new
Date("2024-03-12"), lokacija: "SRC Kukavica"})
```

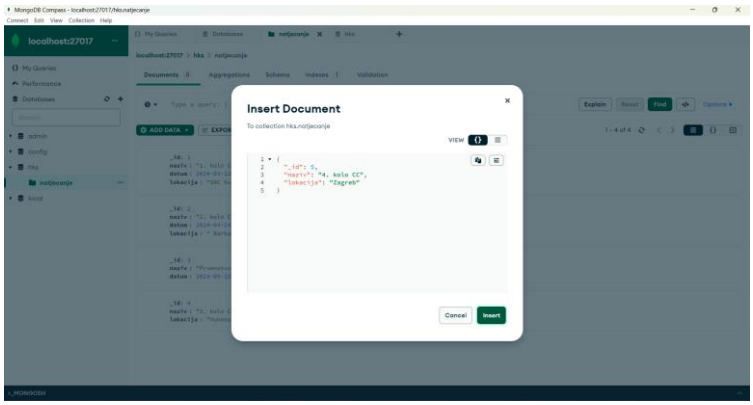
Prethodnom naredbom unesen je jedan dokument u kolekciju *natjecanja*, a radi se o JSON formatu podataka koji je omeđen vitičastim zagradama. Osnovu JSON formata čine objekti koji su u stvari skup parova oblika *ime:vrijednost* poput *naziv: "1. kolo CC".* Korištenjem naredbe insertMany moguće je unijeti više dokumenata u kolekciju odjednom:

```
hks> db.natjecanje.insertMany([
{_id:2, naziv: "2. kolo CC", datum: new Date("2024-04-24"), lokacija:"
Barban"},
{_id:3, naziv: "Prvenstvo Hrvatske", datum: new Date("2024-05-15")},
{_id:4, naziv: "3. kolo CC", lokacija: "Vukosavljevica" }])
```

Prilikom unosa dokumenata, odnosno parova oblika *ime:vrijednost*, dostupni su različiti tipovi podataka (numerički, znakovni, polje, objekt) te je stoga moguće ugnježđivanje (Rabuzin, 2014). U idućem primjeru unosa dokumenta kreirana je nova kolekcija *konj* te su uneseni različiti tipovi podataka u MongoDB Shell-u.

```
hks> db.konj.insertOne({_id:1, ime:"Gidran-121 Alka",
ozdrijebljen: new Date("2009-01-21"),
vrsta:"Gidran",
ocjena: 4.8,
potkovan: true,
stala: {naziv: "Ergela Piber", adresa: "Piber 1, Koflach, Austrija",
boks: 12},
jahaci: ["Sara Gazda", "Ivo Ivic", "Pero Peric"]})
```

Korištenjem grafičkog sučelja MongoDB Compass također je moguće unositi dokumente, a primjer je prikazan na slici 17. Osim toga, na slici 17 je, s obzirom na prikazane dokumente u pozadini, jasno vidljivo da su MongoDB Shell i MongoDB Compass povezani te su sve promjene automatski vidljive i u jednom i u drugom sučelju.



Slika 19. Unos dokumenta u MongoDB Compass-u (autorski rad)

6.1.4. Ispis podataka baze Hrvatskog konjičkog saveza

Ispis dokumenata kolekcije u sustavu za upravljanje bazama podataka MongoDB moguć je naredbom *find* uz definiranje kolekcije čiji se dokumenti ispisuju. U nastavku je prikazan ispis kolekcije *natjecanje*, a dokumenti su upravo sva natjecanja u kalendaru Hrvatskog konjičkog saveza.

```
hks> db.natjecanje.find()
[
  {
    _id: 1,
    naziv: '1. kolo CC',
    datum: ISODate('2024-03-12T00:00:00.000Z'),
    lokacija: 'SRC Kukavica'
  },
  {
    _id: 2,
    naziv: '2. kolo CC',
    datum: ISODate('2024-04-24T00:00:00.000Z'),
    lokacija: ' Barban'
  },
  {
    _id: 3,
    naziv: 'Prvenstvo Hrvatske',
    datum: ISODate('2024-05-15T00:00:00.000Z')
  },
  { _id: 4, naziv: '3. kolo CC', lokacija: 'Vukosavljevica' },
  { _id: 5, naziv: '4. kolo CC', lokacija: 'Zagreb' }
]
```

Iz ispisa dokumenata unutar kolekcije *natjecanje* moguće je primijetiti da sva natjecanja nemaju jednaku strukturu dokumenta odnosno jednake parove. Na primjer, za Prvenstvo Hrvatske još nije određena točna lokacija natjecanja pa ista nije niti unesena. Iako je moguće koristiti tip podatka NULL i napisati npr. *lokacija: null* kreirajući rezervirano mjesto (engl. *placeholder*), nerelacijske baze podataka nemaju strogo definiranu strukturu te nije potrebno da svi dokumenti sadrže jednake parove oblika *ime:vrijednost*. Upravo je fleksibilna struktura baze podataka jedna od osnovnih razlika u radu s NoSQL bazama podataka gdje, za razliku od relacijskih baza, svi redovi (dokumenti) ne moraju imati jednaku strukturu.

6.1.5. Sortiranje i limitiranje ispisa podataka Hrvatskog konjičkog saveza

Osim klasičnog ispisa podataka koji je prikazan u prethodnom primjeru, naredbu *find* moguće je i svojevrsno nadograđivati s ciljem sortiranja i limitiranja rezultata ispisa podataka.

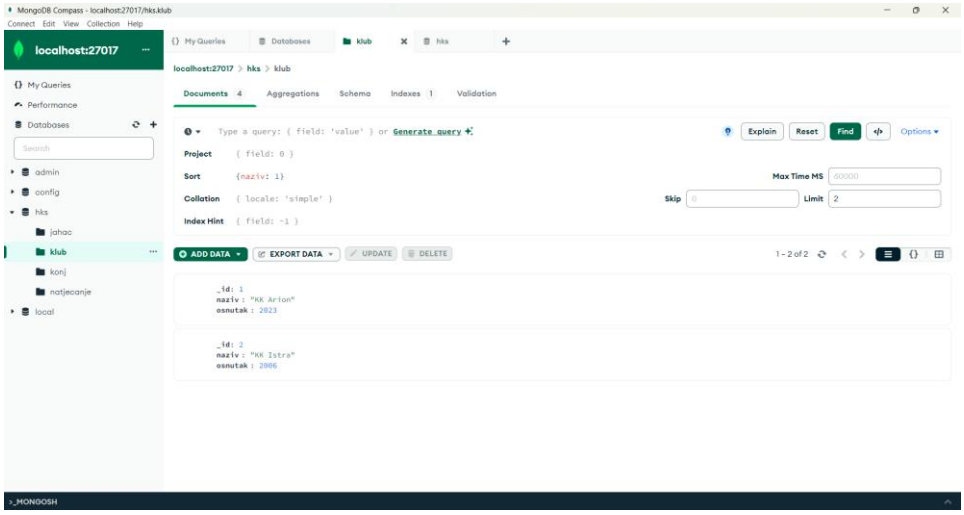
Sortiranje rezultata ispisa moguće je uzlazno, što se označava s 1, i silazno, označeno s -1. Oznaka uzlaznog ili silaznog sortiranja naznačuje se uz ime para prema kojem se želi sortirati rezultate ispisa. U nastavku je prikazan primjer sortiranja klubova silazno prema godinama osnutka.

```
hks> db.klub.find().sort({osnutak: -1})
[
  { _id: 1, naziv: 'KK Arion', osnutak: 2023 },
  { _id: 3, naziv: 'KK Vinia', osnutak: 2015 },
  { _id: 4, naziv: 'KK Moslavina', osnutak: 2010 },
  { _id: 2, naziv: 'KK Istra', osnutak: 2006 }
]
```

Limitiranje rezultata ispisa služi, kao i u relacijskim bazama podataka, određivanju koliko redova rezultata sustav treba prikazati s mogućnošću preskakanja nekih dokumenata. Isječak koda u nastavku prikazuje limitiranje rezultata ispisa jahača na četiri s preskakanjem jednog jahača. Osim toga, u idućem primjeru prilikom upisa dokumenta nije unesen identificirajući objekt *_id* te je zbog toga SUBP sam dodijelio *ObjectId* svakom dokumentu.

```
hks> db.jahac.insertMany([{ime: "Sara Gazda", dob: 22},{ime: "Ivana
Ivic", dob: 17},{ime: "Petra Peric", dob: 11},{ime: "Marko Markic",
dob: 24},{ime: "Lea Leic", dob: 31},{ime: "Zora Zoric", dob: 15},{ime:
"Tara Taric", dob: 25}])

hks> db.jahac.find().limit(4).skip(1)
[
  {
    _id: ObjectId('667882af357761ca9c90deff'),
    ime: 'Ivana Ivic',
    dob: 17
  },
  {
    _id: ObjectId('667882af357761ca9c90df00'),
    ime: 'Petra Peric',
    dob: 11
  },
  {
    _id: ObjectId('667882af357761ca9c90df01'),
    ime: 'Marko Markic',
    dob: 24
  },
  {
    _id: ObjectId('667882af357761ca9c90df02'),
    ime: 'Lea Leic',
    dob: 31
  }
]
```

Sortiranje i limitiranje rezultata ispisa u grafičkom sučelju MongoDB Compass omogućeno je odabirom Opcija (engl. *Options*) u kartici Dokumenti (engl. *Documents*) pojedine kolekcije. Slika 18 prikazuje abecedno sortiranje klubova s limitiranjem rezultata na dva kluba u sučelju MongoDB Compass.



Slika 20. Sortiranje i limitiranje u MongoDB Compass-u (autorski rad)

6.1.6. Filtriranje podataka Hrvatskog konjičkog saveza

Osim mogućnosti sortiranja i limitiranja rezultata, naredba *find* sadrži dva parametra kojima je moguće filtrirati ispisane rezultate. Prvi parametar je uvjet prema kojemu se filtriraju podaci, a paralela se može povući sa klauzulom WHERE u PostgreSQL sustavu. Drugi parametar je popis atributa odnosno parova dokumenata koje sustav treba prikazati što odgovara klauzuli SELECT u PostgreSQL sustavu (Rabuzin, 2014).

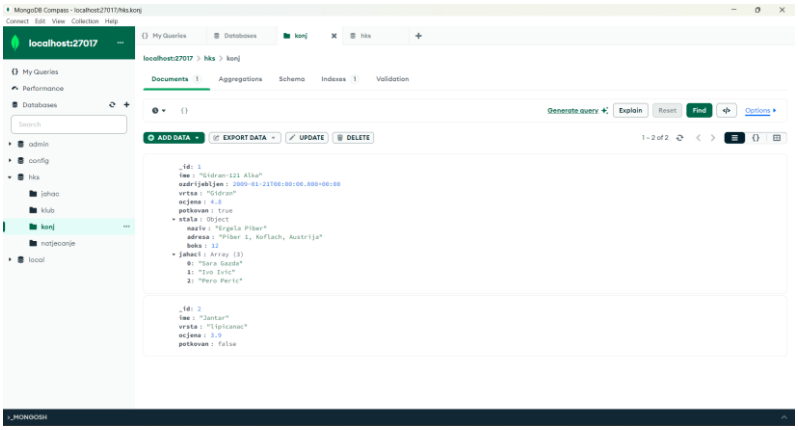
U parametru uvjeta naredbe *find* moguće je postavljati različite uvjete za filtriranje podataka poput točno određene vrijednosti, različite vrijednosti ($ne), manje ili veće vrijednosti ($gt i $lt), uspoređivanja vrijednosti ($in i $nin) te definirati dio vrijednosti navodeći ga unutar kosih crta.

Parametar popisa atributa služi uključivanju ili isključivanju atributa iz ispisa rezultata pri čemu se uključivanje radi navođenjem oznake 1 (*true*) uz ime para, a isključivanje navođenjem oznake 0 (*false*) uz ime para.

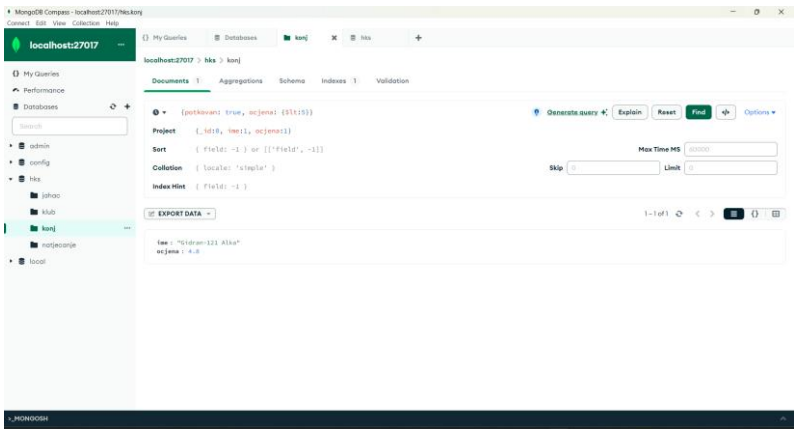
```
hks> db.jahac.find({dob: {$gte:22}}, {_id:0, ime:1})
[
  { ime: 'Sara Gazda' },
  { ime: 'Marko Markic' },
  { ime: 'Lea Leic' },
  { ime: 'Tara Taric' }
]
```

Prethodni isječak koda prikazuje filtriranje rezultata ispisa jahača s uvjetom da pojedini jahač ima više od ili točno 22 godine s time da sustav ispisuje samo ime jahača.

MongoDB Compass također nudi mogućnost filtriranja rezultata postavljanjem uvjeta i određivanjem popisa atributa koje sustav ispisuje. Slika 19 prikazuje ispis dokumenata unutar kolekcije *konj* u MongoDB Compass-u, a na slici 20 prikazano je postavljanje uvjeta i željenih atributa te rezultat ispisa u MongoDB Compass-u.



Slika 21. Ispis kolekcije konj u MongoDB Compass-u (autorski rad)



Slika 22. Filtriranje rezultata ispisa u MongoDB Compass-u (autorski rad)

6.1.7. Logički operatori u bazi Hrvatskog konjičkog saveza

Logički operatori u MongoDB sustavu slični su logičkim operatorima u PostgreSQL sustavu te služe povratu podataka temeljem izraza koji su istiniti ili lažni. Logički operatori se, kao i u relacijskim bazama podataka, često koriste u postavljanju uvjeta za filtriranje podataka, ažuriranje ili brisanje dokumenata. U MongoDB sustavu moguće je koristiti sljedeće logičke operatore:

$and koji vraća dokumente koji zadovoljavaju oba uvjeta,

$not koji vraća dokumente koji ne zadovoljavaju uvjet,

$or koji vraća dokumente koji zadovoljavaju barem jedan uvjet te
$nor koji vraća dokumente koji ne zadovoljavaju nijedan uvjet.
Za prikaz primjera, u nastavku su ispisani dokumenti unutar kolekcije *trener*.

```
hks> db.trener.find()
[
  {
    _id: ObjectId('6678ab74357761ca9c90df05'),
    ime: 'Luka Lukic',
    rang: 'CEI1*'
  },
  {
    _id: ObjectId('6678ab74357761ca9c90df06'),
    ime: 'Jakov Jakic',
    rang: 'CEI2*'
  },
  {
    _id: ObjectId('6678ab74357761ca9c90df07'),
    ime: 'Robi Robic',
    rang: 'FEI3*'
  },
  {
    _id: ObjectId('6678ab74357761ca9c90df08'),
    ime: 'Tin Tinic',
    rang: 'FEI1*'
  }
]
```

Kada se u HKS-u određuju izbornici reprezentacije, minimalni uvjeti za izbornika su treća razina ranga (europskog ili svjetskog) ili bilo koja razina svjetskog, FEI, ranga. Kako bi pronašli kandidate za izbornika, kreiran je sljedeći upit s odgovarajućim uvjetima:

```
hks> db.trener.find({$or:[{rang:/FEI/}, {rang:/3/}]})
[
  {
    _id: ObjectId('6678ab74357761ca9c90df07'),
    ime: 'Robi Robic',
    rang: 'FEI3*'
  },
  {
    _id: ObjectId('6678ab74357761ca9c90df08'),
    ime: 'Tin Tinic',
    rang: 'FEI1*'
  }
]
```

6.1.8. Ažuriranje podataka Hrvatskog konjičkog saveza

Kako bi se dokumenti unutar kolekcija mogli naknadno izmijeniti, koristi se naredba *updateOne* ili *updateMany*. Parametri funkcije su filter prema kojem se određuje koji dokumenti se ažuriraju te samo ažuriranje koje se obavlja nad dokumentima.

Definiranje uvjeta u naredbi *update* jednako je definiranju uvjeta kod korištenja naredbe *find*. U parametru promjene koristi se *$set*, istovjetan klauzuli SET u PostgreSQL sustavu, a

koristi se i $unset kojim je moguće ukloniti određeni par *ime:vrijednost* iz dokumenta. Idući isječak koda prikazuje ažuriranje dokumenata dodavanjem kategorije senior svim jahačima koji su stariji od 20 godina.

```
hks> db.jahac.updateMany({dob:{$gt: 20}}, {$set:{kategorija:
"senior"}})

hks> db.jahac.find({}, {_id:false})
[
  { ime: 'Sara Gazda', dob: 22, kategorija: 'senior' },
  { ime: 'Ivana Ivic', dob: 17 },
  { ime: 'Petra Peric', dob: 11 },
  { ime: 'Marko Markic', dob: 24, kategorija: 'senior' },
  { ime: 'Lea Leic', dob: 31, kategorija: 'senior' },
  { ime: 'Zora Zoric', dob: 15 },
  { ime: 'Tara Taric', dob: 25, kategorija: 'senior' }
]
```

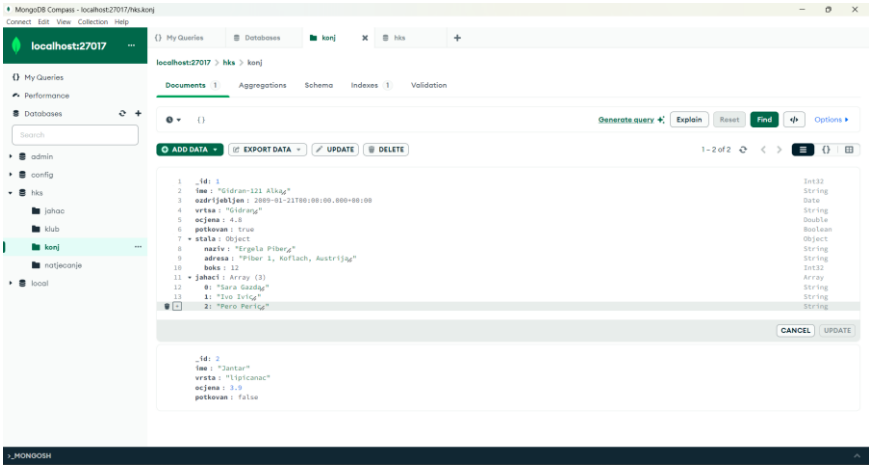
Uklanjanje para korištenjem *$unset* može se primijeniti u slučaju kada konj više ne sudjeluje na natjecanjima i nema definiranih jahača pa mu se stoga ne dodjeljuje ocjena. Primjer za uklanjanje atributa ocjena u tom slučaju prikazan je u nastavku.

```
hks>
db.konj.updateMany({jahaci:{$exists:false}},{$unset:{ocjena:""}})

hks> db.konj.find()
[
  {
    _id: 1,
    ime: 'Gidran-121 Alka',
    ozdrijebljen: ISODate('2009-01-21T00:00:00.000Z'),
    vrtsa: 'Gidran',
    ocjena: 4.8,
    potkovan: true,
    stala: {
      naziv: 'Ergela Piber',
      adresa: 'Piber 1, Koflach, Austrija',
      boks: 12
    },
    jahaci: [ 'Sara Gazda', 'Ivo Ivic', 'Pero Peric' ]
  },
  { _id: 2, ime: 'Jantar', vrsta: 'lipicanac', potkovan: false }
]
```

S obzirom da za konja Jantar nisu bili definirani jahači, što znači da ne može sudjelovati na natjecanjima, za njega je uklonjen atribut ocjena.

Ažuriranje dokumenata, dodavanje i brisanje atributa u grafičkom sučelju MongoDB Compass izuzetno je intuitivne prirode. Pritiskom na ikonu olovke s nazivom Uredi dokument (engl. *Edit document*) dokument se prikazuje kao na slici 21. Odabirom ikone Ukloni polje (engl. *Remove field*) uz broj reda uklanja se par iz dokumenta, odabirom ikone Dodaj polje (engl. *Add field*) uz broj reda dodaje se par u dokument, a ažuriranje para moguće je

jednostavnim upisom novog željenog imena ili vrijednosti. Sve je potrebno potvrditi odabirom gumba Ažuriraj (engl. *Update*) ili odbaciti odabirom Otkaži (engl. *Cancel*).



Slika 23. Ažuriranje dokumenata u MongoDB Compass-u (autorski rad)

6.1.9. Brisanje podataka Hrvatskog konjičkog saveza

U MongoDB Shell SUBP brisanje podataka, odnosno dokumenata, unutar kolekcija moguće je naredbom *deleteOne* i *deleteMany*. Sama naredba *delete* poprima parametar uvjeta kojim se definira koji dokumenti se brišu. Prilikom korištenja naredbe *deleteMany* potreban je oprez pri postavljanju parametra uvjeta kako nehotice ne bi obrisali potrebne podatke. Pretpostavimo da je Prvenstvo Hrvatske otkazano za 2024. godinu te je isto natjecanje potrebno obrisati iz kalendara utrka; isječak koda te svrhe i ispis kolekcije nakon brisanja prikazan je u nastavku.

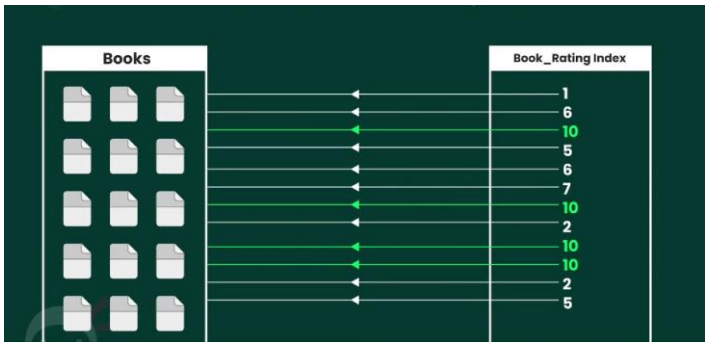
```
hks> db.natjecanje.deleteOne({naziv:"Prvenstvo Hrvatske"})

hks> db.natjecanje.find()
[
  {
    _id: 1,
    naziv: '1. kolo CC',
    datum: ISODate('2024-03-12T00:00:00.000Z'),
    lokacija: 'SRC Kukavica'
  },
  {
    _id: 2,
    naziv: '2. kolo CC',
    datum: ISODate('2024-04-24T00:00:00.000Z'),
    lokacija: ' Barban'
  },
  {
   _id: 4, naziv: '3. kolo CC', lokacija: 'Vukosavljevica'
  },
  {
  _id: 5, naziv: '4. kolo CC', lokacija: 'Zagreb'
  }
]
```

6.1.10. Indeksi u bazi Hrvatskog konjičkog saveza

Indeksi u MongoDb sustavu slični su indeksima u SQL bazama podataka te ovi objekti služe poboljšavanju performansi baza podataka. Indeks se koristi za brže pretraživanje baze podataka, ali važno je naglasiti da zauzima memoriju te usporava umetanje, ažuriranje i brisanje dokumenata pa se treba koristiti za prikladne situacije.

Prema MongoDB Manual-u, kada se izvršava upit potrebno je da sustav skenira svaki dokument unutar kolekcije kako bi vratio ispravne rezultate. U radu s velikim podacima (engl. *Big Data*) pojedina kolekcija može sadržavati tisuće i tisuće dokumenata te skeniranje svih dokumenata nikako nije optimalno. Korištenjem indeksa, MongoDB pregledava indeks te pronalazeći odgovarajuće vrijednosti, putem pokazivača na dokument pristupa dokumentima koji zadovoljavaju upit i pregledava samo njih. Slika 22 prikazuje primjer indeksa kreiranog za pretraživanje knjiga po ocjenama.

Iako korištenje indeksa poboljšava performanse pretraživanja kolekcija, svaka izmjena, unos ili brisanje podataka kolekcije znači da se i sam indeks mora ažurirati kako bi odgovarao kolekciji te stoga usporava navedene operacije.



Slika 24. Indeks za kolekciju knjiga u MongoDB (https://netninja.dev/)

U bazi Hrvatskog konjičkog saveza kreiran je indeks za pretraživanje jahača koji su mlađi od 20 godina. Korištenjem naredbe *explain(„executionStats“)* uz rezultate upita prikazane su detaljne statistike o izvršavanju upita, a u smislu indeksa konkretno nam koriste informacije o broju rezultata i skeniranih dokumenata.

Kod u nastavku prikazuje upit koji prikazuje jahače mlađe od 20 godina te je u *executionStats* ispisu moguće vidjeti da je pregledano sedam dokumenata, koliko ima jahača u kolekciji, te su prikazana tri rezultata koja zadovoljavaju upit.

```
hks> db.jahac.find({dob: {$lt:20}}).explain("executionStats")

[
  {
    _id: ObjectId('667882af357761ca9c90deff'),
```

```
    ime: 'Ivana Ivic',
    dob: 17
  },
  {
    _id: ObjectId('667882af357761ca9c90df00'),
    ime: 'Petra Peric',
    dob: 11
  },
  {
    _id: ObjectId('667882af357761ca9c90df03'),
    ime: 'Zora Zoric',
    dob: 15
  }
]
{
  executionStats: {
    executionSuccess: true,
    nReturned: 3,
    executionTimeMillis: 0,
    totalKeysExamined: 0,
    totalDocsExamined: 7,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { dob: { '$lt': 20 } },
      nReturned: 3,
      executionTimeMillisEstimate: 0,
      works: 8,
      advanced: 3,
      needTime: 4,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 7
    }
}, ok: 1
}
```

S obzirom da baza Hrvatskog konjičkog saveza sadrži kolekciju *jahac* sa samo sedam dokumenata, poboljšanje performansi pretraživanja neće biti značajno, ali je prikladan primjer za prikaz korisnosti indeksa u radu s velikim podacima (engl. *Big Data*). Sljedeći isječak koda prikazuje kreiranje indeksa na temelju atributa dob gdje su podaci u indeksu sortirani uzlazno.

```
hks> db.jahac.createIndex({dob: 1})
dob_1
```

Prikazom indeksa za kolekciju *jahac* prikazan je indeks primarnih ključeva koje sustav MongoDB samostalno kreira za svaku kolekciju te prethodno kreiran indeks *dob_1*.

```
hks> db.jahac.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { dob: 1 }, name: 'dob_1' }
]
```

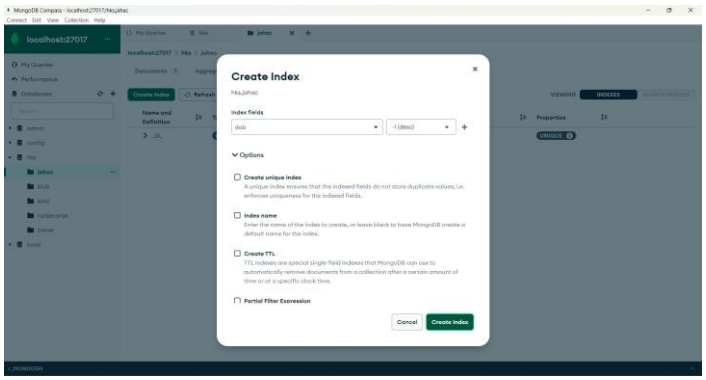
Kada je indeks *dob_1* kreiran, moguće je prikazom statistike izvršenja istog upita za jahače mlađe od 20 godina primijetiti utjecaj pretraživanja uz postojanje indeksa.

```
hks> db.jahac.find({dob: {$lt:20}}).explain("executionStats")
[
  {
    _id: ObjectId('667882af357761ca9c90df00'),
    ime: 'Petra Peric',
    dob: 11
  },
  {
    _id: ObjectId('667882af357761ca9c90df03'),
    ime: 'Zora Zoric',
    dob: 15
  },
  {
    _id: ObjectId('667882af357761ca9c90deff'),
    ime: 'Ivana Ivic',
    dob: 17
  }
]
{
executionStats: {
    executionSuccess: true,
    nReturned: 3,
    executionTimeMillis: 12,
    totalKeysExamined: 3,
    totalDocsExamined: 3,
    executionStages: {
      stage: 'FETCH',
      nReturned: 3,
      executionTimeMillisEstimate: 10,
      works: 4,
      advanced: 3,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 3,
      alreadyHasObj: 0,
    }
  }, ok: 1
}
```

Analizom statusa izvršenja nakon kreiranja indeksa *dob_1* za kolekciju *jahac* moguće je uvidjeti svrhu indeksa i njegovo djelovanje. Rezultat ponovljenog upita za pronalazak jahača mlađih od 20 godina ponovno je jednak, prikazana su tri jahača, no u ovom su slučaju pregledana samo tri dokumenta, a ne svih sedam dokumenata unutar kolekcije *jahac*. Ovime je konkretno pokazano da će naredba *find* u slučaju postojanja indeksa prvo pregledati kreirani indeks, unutar njega pronaći vrijednosti koje zadovoljavaju uvjet upita te će iz indeksa, putem pokazivača, pristupiti dokumentima kolekcije koji zadovoljavaju upit. Tako je korištenjem indeksa izbjegnuto pregledavanje svih dokumenata unutar kolekcije, kojih može biti jako puno, i performanse baze podataka su bolje. S obzirom da indeksi ipak zauzimaju memoriju i usporavaju određene operacije nad bazom podataka, kada se indeksi ne koriste ili su kreirani s pogrešnom svrhom, preporuka ih je obrisati. Brisanje indeksa i ispis svih indeksa kolekcije *jahac* nakon uklanjanja prikazano je kodom u nastavku.

```
hks> db.jahac.dropIndex({dob: 1})
{ nIndexesWas: 2, ok: 1 }
hks> db.jahac.getIndexes()
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

Kreiranje indeksa u grafičkom sučelju MongoDB za određenu kolekciju moguće je u kartici Indeksi (engl. *Indexes*) odabirom gumba Kreiraj indeks (engl. *Create index*). Kreiranje indeksa za kolekciju jahac koji je sortiran silazno pod dobi prikazano je na slici 23.



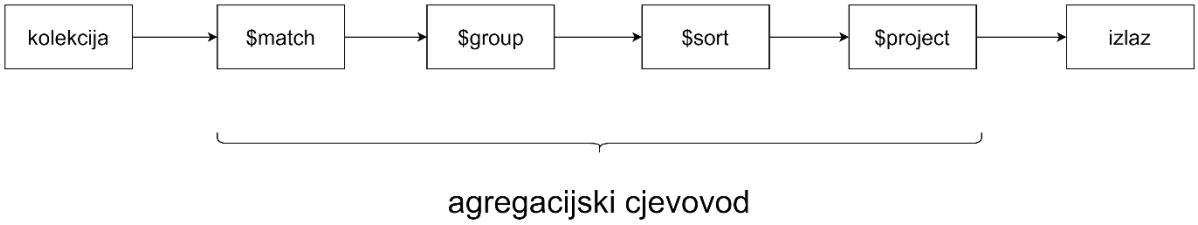
Slika 25. Kreiranje indeksa u MongoDB Compass-u (autorski rad)

6.1.11. Agregirajuće funkcije u bazi Hrvatskog konjičkog saveza

U dosadašnjim primjerima korištena je naredba *find()* za kreiranje jednostavnih upita postavljanjem parametara uvjeta i parametra popisa atributa. Međutim, kada upiti postaju složeniji potrebno je koristi agregirajuće funkcije koje se koriste za različite kompleksne operacije. Agregacija u sustavu MongoDB je proces grupiranja više dokumenata iz kolekcije te korištenje agregirajuće funkcije s povratom rezultata korisniku.

Korištenje agregirajućih funkcija omogućava grupiranje vrijednosti iz različitih dokumenata, povezivanje podataka za izvršavanje kompleksnih naredbi te filtriranje i sortiranje dokumenata i analiziranje čestih promjena podataka tijekom vremena. Kako je navedeno na službenim stranicama sustava, za agregirajuće funkcije mogu se koristiti agregacijski cjevovod (engl. *aggregation pipelines*) koji su poželjna metoda ili jednonamjenske metode agregacije koje su jednostavnije, ali nemaju mogućnosti kao cjevovodi agregacija.

Slika 25 prikazuje jednostavniji agregacijski cjevovod u MongoDB sustavu koji se sastoji od četiri faze (engl. *stage*) agregacije s ciljem prikaza funkcioniranja faza. Prikazani agregacijski cjevovod sastoji se od metoda *$match, $group, $sort* i *$project* pri čemu svaka od faza agregacijskog cjevovoda izvodi određenu operaciju na ulaznim dokumentima te izlazne dokumente prosljeđuje idućoj fazi.



Slika 26. Agregacijski cjevovod u MongoDB (autorski rad)

Faza *$match* agregacijskog cjevovoda za parametar ima uvjet po kojemu se povezuju dokumenti kolekcije i tako određuje nad kojim dokumentima se provode agregirajuće funkcije. Izlazni dokumenti faze *$match* su ulazni dokumenti faze *$group* koja služi grupiranju dokumenata u grupe određene prema nekoj zajedničkoj vrijednosti. Faza *$group* potom prosljeđuje izlazne dokumente fazi *$sort* koja sortira dokumente određenim redoslijedom te je na kraju u fazi *$project* moguće ukloniti nepotrebne atribute ili dodati nove.

Agregirajuće funkcije koje su na raspolaganju unutar agregacijskog cjevovoda u sustavu MongoDB su mnogobrojne, a neke najkorištenije su:
$sum – sumira vrijednosti atributa dokumenata u kolekciji,
$avg – vraća prosječnu vrijednost atributa dokumenata kolekcije,
$min – vraća minimalnu vrijednost atributa,
$max – vraća maksimalnu vrijednost atributa,
$round – zaokružuje vrijednost na cijeli broj ili određeni broj decimala te
$push – vraća niz svih vrijednosti koje proizlaze iz dokumenta.

Za korištenje agregirajućih cjevovoda potrebna je naredba *db.collection.aggregate()* u koju se potom upisuju sve faze agregacije s pripadajućim parametrima. Za potrebe prikaza primjera u kolekciju *konj* dodani su dokumenti o tri konja, a kolekcija je prikazana u nastavku.

```
hks> db.konj.find()
[{
    _id: 1,
    ime: 'Gidran-121 Alka',
    ozdrijebljen: ISODate('2009-01-21T00:00:00.000Z'),
    ocjena: 4.8,
    potkovan: true,
    stala: {
      naziv: 'Ergela Piber',
      adresa: 'Piber 1, Koflach, Austrija',
      boks: 12
    },
    jahaci: [ 'Sara Gazda', 'Ivo Ivic', 'Pero Peric' ],
    vrsta: 'Gidran'
  },
  {
    _id: 2,
```

```
    ime: 'Jantar',
    vrsta: 'lipicanac',
    potkovan: false,
    ozdrijebljen: ISODate('2010-01-12T23:00:00.000Z'),
    ocjena: 2.5
  },
  {
    _id: 3,
    ime: 'Hungaro',
    vrsta: 'Gidran',
    ocjena: 3.9,
    potkovan: false,
    jahaci: [ 'Pero Peric', 'Ana Anic' ],
    ozdrijebljen: ISODate('2011-11-20T00:00:00.000Z')
  },
  {
    _id: 4,
    ime: 'Cleopatra',
    potkovan: false,
    stala: { naziv: 'Ergela Spar', boks: 22 },
    ozdrijebljen: ISODate('2005-03-12T00:00:00.000Z')
  },
  {
    _id: 5,
    ime: 'Lahari',
    vrsta: 'HSK',
    ocjena: 5,
    potkovan: true,
    jahaci: [ 'Sara Gazda', 'Marko Markic' ]
  }]
```

Hrvatski konjički savez ima potrebu prikazati konje mlađe od 18 godina kako bi mogli odlučiti o prijavi konja na europsku izložbu vrsta uzgojnih konja. Rezultat ovakvog upita moguće je prikazati korištenjem agregirajućih funkcija na sljedeći način:

```
hks> db.konj.aggregate([
      {
      $match: {"ozdrijebljen":{$gt: new ISODate("2005-12-31")}}
      },
      {
      $group:{_id:"$vrsta",konji:{$push: "$ime"},
            ocjene:{$push: "$ocjena"} , prosjecna: {$avg: "$ocjena"}}
      },
      {
      $sort: {prosjecna: -1}
      }
      ])
```

U *$match* fazi agregiranja definiran je uvjet prema kojemu će se u obzir doći samo dokumenti koji imaju definiran datum oždrebljenja i pri tome je najraniji datum 01.01.2006. kako bi se osiguralo da su konji mlađi od 18 godina. Nakon izvršavanja faza *$match* proslijedila je dokument fazi *$group* u kojoj je definirano grupiranje konja prema pripadnosti vrsti. Osim samog grupiranja, unutar grupe prikazana su imena konja, njihove ocjene te prosječna ocjena za pojedinu vrstu konja. Faza *$group* je naposljetku proslijedila dokument fazi *$sort* u kojoj je

definirano da se rezultati sortiraju tako da su grupe prikazane od najviše prosječne ocjene prema najnižoj.

```
[
  {
    _id: 'Gidran',
    konji: [ 'Gidran-121 Alka', 'Hungaro' ],
    ocjene: [ 4.8, 3.9 ],
    prosjecna: 4.35
  },
  {
    _id: 'lipicanac',
    konji: [ 'Jantar' ],
    ocjene: [ 2.5 ],
    prosjecna: 2.5
  }
]
```

6.1.12. Spajanje kolekcija u bazi Hrvatskog konjičkog saveza

U sustavu MongoDB moguće je uz pomoć agregacijske faze *$lookup* povezati dvije kolekcije. Kako je navedeno na službenim stranicama sustava, faza agregacijskog cjevovoda *$lookup* izvodi lijevo vanjsko spajanje (engl*. left outer join*) pri čemu pruža mogućnost spajanja kolekcija unutar iste baze podataka s ciljem filtriranja dokumenata iz povezane kolekcije. Faza *$lookup* funkcionira na način da dodaje novo polje u dokument koje sadrži odgovarajuće dokumente iz povezane kolekcije te zatim prosljeđuje preoblikovane dokumente u iduću fazu agregacijskog cjevovoda. Sintaksa povezivanja kolekcija fazom *$lookup* temeljem jednakih vrijednosti ulaznih dokumenata i dokumenata povezane kolekcije prikazana je u nastavku.

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from"
                      collection>,
       as: <output array field>
     }
}
```

U bazi podataka Hrvatskog konjičkog saveza jahačima su dodani podaci o pripadnosti klubu kako je prikazano u nastavku:

```
hks> db.jahac.find()
[
  {
    _id: ObjectId('667882af357761ca9c90defe'),
    ime: 'Sara Gazda',
    dob: 22,
    kategorija: 'senior',
    klub: 'KK Arion'
  },
  {
```

```
    _id: ObjectId('667882af357761ca9c90deff'),
    ime: 'Ivana Ivic',
    dob: 17,
    klub: 'KK Arion'
  },
  {
    _id: ObjectId('667882af357761ca9c90df00'),
    ime: 'Petra Peric',
    dob: 11,
    klub: 'KK Vinia'
  },
  {
    _id: ObjectId('667882af357761ca9c90df01'),
    ime: 'Marko Markic',
    dob: 24,
    kategorija: 'senior',
    klub: 'KK Moslavina'
  },
  {
    _id: ObjectId('667882af357761ca9c90df02'),
    ime: 'Lea Leic',
    dob: 31,
    kategorija: 'senior',
    klub: 'KK Istra'
  },
  {
    _id: ObjectId('667882af357761ca9c90df03'),
    ime: 'Zora Zoric',
    dob: 15,
    klub: 'KK Moslavina'
  },
  {
    _id: ObjectId('667882af357761ca9c90df04'),
    ime: 'Tara Taric',
    dob: 25,
    kategorija: 'senior',
    klub: 'KK Istra'
  }
]
```

Korištenjem faze *$lookup* moguće je prikazati klubove i njihove članove u bazi *hks*:

```
hks> db.klub.aggregate([
      {
      $lookup:{
            from:"jahac",
            localField:"naziv",
            foreignField: "klub",
            as: "clanoviKluba"}},
      {

      $project:{
            _id:0,
            naziv:1,
            bodovi:1,
            clanoviKluba:{
                  $map:{
                        input: "$clanoviKluba",
                        as: "clan",
                        in:"$$clan.ime"}}}}
      ])
```

Rezultat povezivanja kolekcija *klub* i *jahac* iz baze *hks* je ispis klubova s pripadajućim članovima, a sam izgled ispisa definiran je u fazi *$project* odabirom atributa i podataka koji se prikazuju.

```
[
  {
    naziv: 'KK Arion',
    bodovi: 4352,
    clanoviKluba: [ 'Sara Gazda', 'Ivana Ivic' ]
  },
  {
    naziv: 'KK Istra',
    bodovi: 4263,
    clanoviKluba: [ 'Lea Leic', 'Tara Taric' ]
  },
  {
    naziv: 'KK Vinia',
    bodovi: 2356,
    clanoviKluba: [ 'Petra Peric' ]
  },
  {
    naziv: 'KK Moslavina',
    bodovi: 1209,
    clanoviKluba: [ 'Marko Markic', 'Zora Zoric' ]
  }
]
```

U prethodnom primjeru prikazano je povezivanje kolekcija u fazi *$lookup* temeljem jednog podudaranja vrijednosti, konkretno naziva kluba, a u fazi *$lookup* moguće je izvoditi i upite s dvije kolekcije uz izvođenje drugih uvjeta spajanja pored podudaranja korištenjem sljedeće sintakse (MongoDB, bez dat):

```
{
   $lookup:
      {
         from: <joined collection>,
         let: {<var_1>: <expression>,…,<var_n>: <expression>},
         pipeline: [ <pipeline to run on joined collection> ],
         as: <output array field>
      }
}
```

Pri tome se u polju *let* specificiraju varijable koje se nakon toga koriste u cjevovodu kako bi cjevovod imao pristup spojenim poljima dokumenata. Upravo uz korištenje agregacijskog cjevovoda moguće je izvoditi faze agregacije i agregirajuće funkcije nad dokumentima koji su rezultat povezivanja kolekcija.

Primjer složenijeg upita s povezivanjem kolekcija čiji je cilj prikazati ukupan broj klubova s prosječnim brojem bodova te klub s najviše i najmanje bodova prikazan je u nastavku:

```
hks> db.klub.aggregate([

{ $group: {
      _id: null,
```

```
      ukupnoKlubova: { $sum: 1 },
      prosjecniBodovi: { $avg: "$bodovi" },
      najveciBodovi: { $max: "$bodovi" },
      najmanjiBodovi: { $min: "$bodovi" } } },

{ $lookup: {
      from: "klub",
      let: { maxBodovi: "$najveciBodovi" },
      pipeline: [
            { $match: { $expr: { $eq: ["$bodovi", "$$maxBodovi"]}}},
            { $lookup: {
                  from: "jahac",
                  localField: "naziv",
                  foreignField: "klub",
                  as: "clanovi" } },
      { $project: {
            _id: 0,
            naziv: 1,
            bodovi: 1,
            clanovi: {
                  $map:{
                        input:"$clanovi",
                        as:"clan",
                        in: "$$clan.ime"}}}}
      ],
      as: "klubNajveciBodovi" } },

{ $lookup: {
      from: "klub",
      let: { minBodovi: "$najmanjiBodovi" },
      pipeline: [
            { $match: { $expr: { $eq: ["$bodovi", "$$minBodovi"]}}},
            { $lookup: {
                  from: "jahac",
                  localField: "naziv",
                  foreignField: "klub",
                  as: "clanovi" } },
      { $project: {
            _id: 0,
            naziv: 1,
            bodovi: 1,
            clanovi: {
                  $map:{
                        input:"$clanovi",
                        as:"clan",
                        in:"$$clan.ime" }}}}
      ],
      as: "klubNajmanjiBodovi" } },

{ $project: {
      _id: 0,
      ukupnoKlubova: 1,
      prosjecniBodovi: { $round: ["$prosjecniBodovi", 2] },
      klubNajveciBodovi: "$klubNajveciBodovi",
      klubNajmanjiBodovi: "$klubNajmanjiBodovi" } }
])
```

U prethodno prikazanom kodu prvo su nad dokumentima kolekcije *klub* unutar faze *$group* korištene agregirajuće funkcije kojima je izračunat ukupan broj klubova, prosječan broj bodova svih klubova te maksimalni i minimalni bodovi. Nakon toga je, za dohvaćanje podataka

o klubu s najvećim brojem bodova, u fazi *$lookup* u polju *let* definirano korištenje varijable *maxBodovi* u polju *pipeline*. U polju *pipeline* kreiran je cjevovod koji u fazi *$match* filtrira dokumente na način da pronađe klub s bodovima koji odgovaraju maksimalnim bodovima. Uz to je u cjevovod dodana faza *$lookup* u kojoj se kolekcija *klub* povezuje s kolekcijom *jahac* kako bi uz sam klub ispisali i članove kluba. U fazi *$project* zadano je da se od podataka o klubu ispisuju samo naziv i bodovi te imena članova, a s opcijom as definiran je naziv za prikaz kluba s najvećim bodovima. Jednak kod napisan je za prikaz kluba s najmanjim brojem bodova i njegovim članovima. Na kraju cjelokupnog cjevovoda u fazi *$project* definirano je koji atributi će se ispisati u rezultatu kreiranog upita, a prosječni bodovi zaokruženi su funkcijom *$round* na dvije decimale. U nastavku se nalazi rezultat kreiranog upita:

```
[
  {
    ukupnoKlubova: 4,
    prosjecniBodovi: 3045,
    klubNajveciBodovi: [
      {
        naziv: 'KK Arion',
        bodovi: 4352,
        clanovi: [ 'Sara Gazda', 'Ivana Ivic' ]
      }
    ],
    klubNajmanjiBodovi: [
      {
        naziv: 'KK Moslavina',
        bodovi: 1209,
        clanovi: [ 'Marko Markic', 'Zora Zoric' ]
      }
    ]
  }
]
```

6.1.13. Transakcije u bazi Hrvatskog konjičkog saveza

Autori Sadalage i Fowler (2013) navode kako transakcije u smislu poznatom u tradicionalnim sustavima za upravljanje bazama podataka nisu dostupne u sustavima za upravljanje nerelacijskim bazama podataka. Radi se o tome da promjene u bazi podataka, na primjer unos ili ažuriranje podataka, za koje je transakcijama moguće odrediti hoće li se izvršiti ili poništiti, nije dostupno u nerelacijskim sustavima jer u njima promjene poput pisanja ili uspijevaju ili ne uspijevaju.

Na službenim stranicama MongoDB sustava navedeno je da je operacija na jednom dokumentu atomna operacija, što znači da se izvršava u cijelosti ili se ne izvršava, a sama atomnost jednog dokumenta uklanja potrebu za distribuiranim transakcijama u mnogim slučajevima. Ipak, ukoliko je potrebna atomnost čitanja i pisanja u više dokumenata moguće je kreirati distribuirane transakcije u MongoDB sustavu te koristiti transakcije u više operacija,

dokumenata, kolekcija i baza podataka. Distribuirane transakcije također su atomne transakcije s obzirom da ili primjenjuju sve promjene podataka ili poništavaju promjene te izolirane s obzirom da se sve promjene podataka napravljene u transakciji vide izvan transakcije tek nakon izvršavanja (MongoDB, bez dat).

U sustavu MongoDB transakcije su usko povezane sa sesijama s obzirom da sesije omogućavaju izvršavanje ACID transakcija koje se izvode unutar samih sesija. Za jednu sesiju moguće je imati najviše jednu otvorenu transakciju koja se, ako je otvorena, u slučaju završetka sesije prekida (MongoDB, bez dat).

Za rad s transakcijama unutar sesija moguće je koristiti naredbu za započinjanje transakcije *session.startTransaction(),* izvršavanje transakcije *session.commitTransaction(),* prekidanje transakcije i poništavanje svih promjena *session.abortTransaction()* te pokretanje specificirane lambda funkcije naredbom *session.withTransaction().*

U nastavku je prikazan primjer unosa novih 80 bodova klubu KK Arion koji je pobijedio na utrci od 80 kilometara na Prvenstvu Hrvatske.

```
const session = db.getMongo().startSession()

session.startTransaction()

const klub = session.getDatabase('hks').getCollection('klub')

klub.updateOne({naziv:"KK Arion"}, {$inc:{bodovi:80}})

{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

session.commitTransaction()
```

6.1.14. Pogledi u bazi Hrvatskog konjičkog saveza

O pogledima je bilo govora u prethodnim poglavljima, a u sustavu MongoDB pogledi su upitni objekti samo za čitanje koji se ne pohranjuje već se prikazuje kada klijent kreira upit za pogled. Sadržaj pogleda definiran je agregacijskim cjevovodima kreiranim nad kolekcijama dokumenata ili drugim pogledima (MongoDB, bez dat). Sintaksa za kreiranje pogleda dana je u nastavku.

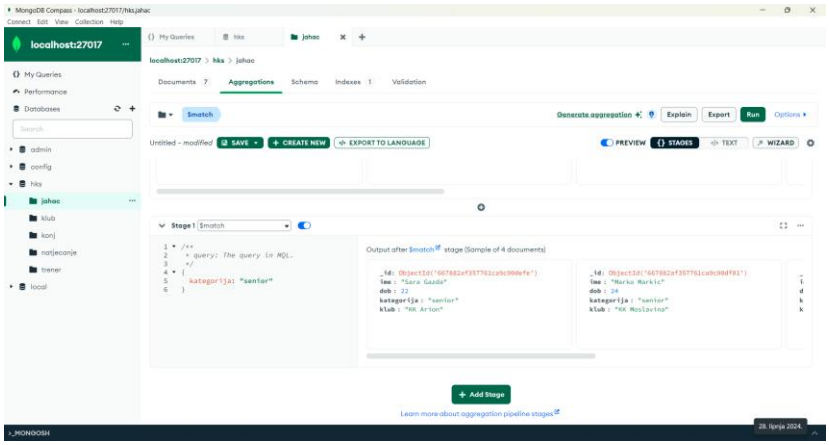
```
db.createView(
  "<viewName>",
  "<source>",
```

```
    [<pipeline>],
    {
      "collation" : { <collation> }
    }
)
```

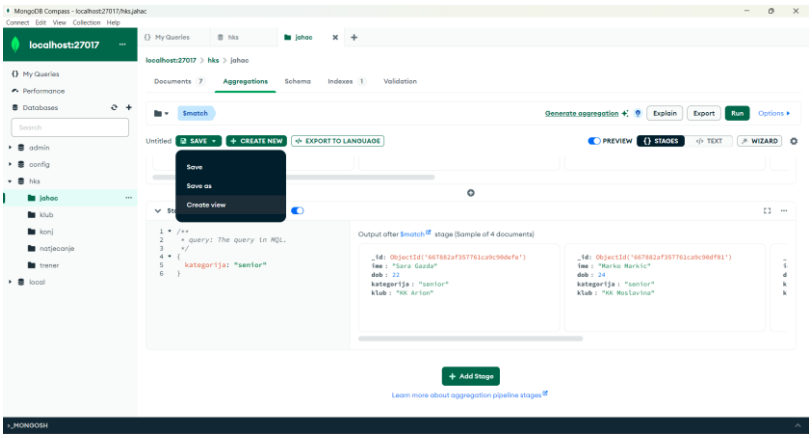
Ograničenja pogleda u sustavu MongoDB odnose se na nemogućnost preimenovanja pogleda jednom kada je kreiran te mogućnost kreiranja pogleda isključivo u bazi u kojoj se nalazi kolekcija nad kojom se kreira pogled.

U nastavku je prikazano kreiranje i brisanje pogleda koji sadrži jahače u kategoriji seniora u MongoDB Shell-u i grafičkom sučelju MongoDB Compass.

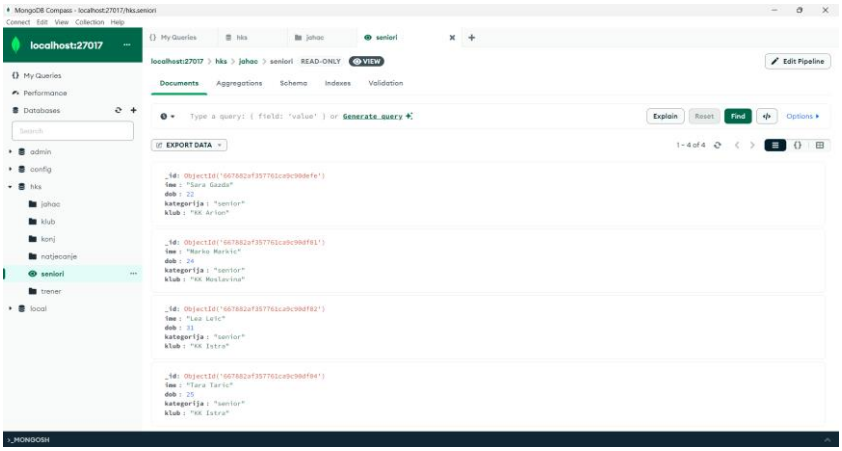
```
hks> db.createView(
   "seniori",
   "jahac",
   [{$match: {kategorija: "senior"}}])
{ ok: 1 }

hks> db.seniori.find({}, {_id:0, ime:1, dob:1, kategorija:1})
[
  { ime: 'Sara Gazda', dob: 22, kategorija: 'senior' },
  { ime: 'Marko Markic', dob: 24, kategorija: 'senior' },
  { ime: 'Lea Leic', dob: 31, kategorija: 'senior' },
  { ime: 'Tara Taric', dob: 25, kategorija: 'senior' }
]

hks> db.seniori.drop()
true
```



Slika 27. Kreiranje agegacijskog cjevovoda u MongoDB Compass-u (autorski rad)



Slika 28. Spremanje cjevovoda kao pogled u MongoDB Compass-u (autorski rad)



Slika 29. Prikaz pogleda "seniori" u MongoDB Compass-u (autorski rad)

7. Usporedba sustava PostgreSQL i Mongo DB

7.1. Prednosti i nedostatci relacijskih i nerelacijskih baza podataka

Relacijske i nerelacijske baze podataka razlikuju se po mnogo čemu i prilikom odabira SUBP važno je detaljno proučit model podataka te pokušati odabrati najprikladniju bazu podataka. Ipak, čak i najpovoljnija vrsta baza podataka ovisno o modelu može imati svoje manjkavosti, upravo kao što i relacijske i nerelacijske baze podataka imaju prednosti i nedostataka.

Govorimo li o relacijskim bazama podataka, jedna od njihovih osnovnih prednosti je strukturiranost podataka i modeliranje jasno definirane strukture podataka. Fiksna shema i definiranost strukture podataka te dugogodišnja dominacija relacijskih baza u području upravljanja s podacima doveli su do visoke razine standardizacije SQL jezika te dostupnost mnogobrojnih sustava za upravljanje bazama podataka u kojima se lako usvaja praksa rada. Osim toga, kada se u relacijskim SUBP upravlja podacima na ispravan i profesionalan način, relacijske baze podataka osiguravaju visoku razinu pouzdanosti, sigurnosti i integriteta podataka podržavajući karakteristike ACID transakcija.

Iako je fiksna shema i definirana struktura podataka s jedne strane prednost relacijskih baza podataka, u situacijama kada je potrebno mijenjati strukturu podataka sama izmjena može biti prilično složena i zahtijevati puno vremena s obzirom na strogu strukturu relacijskog modela podataka. Skalabilnost, koja označava svojstvo sustava da uspješno nastavlja s radom prilikom povećanja opsega zadataka ili dodavanja resursa, kod relacijskih baza podataka je vertikalna. Konkretno, vertikalna skalabilnost označava dodavanje resursa i povećanja opsega zadataka serveru dok horizontalna skalabilnost podrazumijeva dodavanje više servera u svrhu raspodjele opterećenja, zadataka i mogućnosti upravljanja velikim količinama podataka. Kako SQL baze podataka imaju vertikalnu skalabilnost, često se suočavaju s lošijim performansama prilikom upravljanja velikim količinama podataka, (engl. *Big Data*), naspram NoSQL baza podataka.

Horizontalna skalabilnost NoSQL baza podataka čini nerelacijske baze podataka pogodne za upravljanje iznimno velikim količinama podataka bez značajnog pada performansi. Nadalje, naspram fiksne sheme relacijskih baza, NoSQL baze podataka odlikuje fleksibilnost sheme i različitost modela podataka što omogućava odabir prigodnog modela podataka bez potrebe za migracijom ili konverzijom podataka. Već je spomenuto u radu da nerelacijske baze

podataka pružaju bolje performanse prilikom izvođenja pojedinih jednostavnih operacija, poput pretraživanja ili dohvaćanja podataka, jer nije potrebno uvijek pretraživati cijelu bazu podataka.

Raznolikost modela podataka i fleksibilna shema nerelacijskih baza te djelomična nezrelost tehnologije utjecale su na teško razvijanje unificiranog NoSQL jezika i općenit nedostatak standardizacije. Nerelacijske baze podataka nisu u mogućnosti ispunjavati karakteristike ACID transakcija, ali ih dobro opisuje postizanje BASE akronima koji sam po sebi ima određene nedostatke, ali je za potrebe upravljanja *Big Data* često upravo ono što je potrebno. Iako ima odlike visoke razine fleksibilnosti i brzine, upravljanje NoSQL bazama podataka u SUBP djelomično je ograničeno izborom operacija koje je moguće izvršiti te nedostatkom mogućnosti izvođenja kompleksnih upita.

Navedene prednosti i nedostatci relacijskih i nerelacijskih baza podataka karakteriziraju obje vrste modela podataka te je prilikom izbora određenog sustava za upravljanje bazama podataka potrebno analizirati model podataka, zahtjeve skalabilnosti i fleksibilnosti te potrebe klijenta.

7.2. Sustavi PostgreSQL i MongoDB

PostgreSQL i MongoDB sustavi za upravljanje bazama podataka među najpopularnijim su SUBP na svijetu. Osnovna razlika dva sustava je vrsta baze podataka kojom upravljanju; PostgreSQL je sustav za upravljanje relacijskim bazama podataka dok je MongoDB namijenjen za dokumentne nerelacijske baze podataka. Započeti korištenje PostgreSQL i MongoDB sustava nije zahtjevno s obzirom da su stranice oba sustava pune iscrpnih uputa o instalaciji i postavljanju okruženja za početak rada. Osim toga, obje stranice nude dokumentaciju i upute za početak rada s bazama podataka u sustavu, a online su dostupni razni materijali i tutoriali (engl. *tutorials*) koja pružaju podršku u učenju upravljanja bazama podataka određenim SUBP.

PostgreSQL sustav koristi SQL (engl. Structured Query Language) za upravljanje podacima unutar baze te pruža moćne performanse manipulacije strukturiranim podacima. Baza podataka u PostgreSQL sustavu sastoji se od tablica i njihovih međusobnih odnosa koji su implementirani primarnim i vanjskim ključevima. Pojedina tablica predstavlja jedan entitet koji sadrži atribute kao stupce važne za definiranje samog entiteta, a redovi tablica predstavljaju instance entiteta. S druge strane, MongoDB koristi upitni jezik koji se bazira na JSON formatu u čijem su središtu parovi *ime:vrijednost*. Baza podataka u MongoDB sustavu sastoji se od kolekcija, slično tablicama, pri čemu se svaka pojedina kolekcija sastoji od

dokumenata, sličnih redovima, napisanih u JSON formatu koji je intuitivan i jednostavan te razumljiv i čovjeku i računalu.

S obzirom da su oba sustava puna prednosti za upravljanje određenim tipom baza podataka, važno je pravilno izabrati sustav koji je pogodan za upravljanje određenim modelom podataka. PostgreSQL sustav prigodan je za upravljanje podacima koje zahtjeva složene upite, funkcije, transakcije i visoku razinu dosljednosti podataka te se stoga koristi u modeliranju podataka financijskih sustava i općenito ERP sustava (engl. *Enterprise resource planing*) u čijem je središtu često upravo relacijska baza podataka. PostgreSQL prigodan je za upravljanje strukturiranim, tipiziranim podacima održavajući referencijalni integritet te je stoga pouzdan za kritične aplikacije u kojima je dosljednost podataka od velike važnosti. MongoDB sustav često se koristi u web aplikacijama kada je potrebno upravljati izuzetno velikim podacima (engl. *Big Data*) gdje dolazi do nešto drugačijih zahtjeva od SUBP. Upravljanje velikim podacima ne zahtjeva jednako visoku preciznost i dosljednost podataka kao relacijski modeli te je često dovoljno zadovoljiti BASE akronim. Fleksibilnost sheme nerelacijskih baza podataka omogućava visoku razinu prilagodbe na nagle promjene u zahtjevima podataka, a horizontalno skaliranje jednostavno odgovara velikoj količini podataka i pruža visoku razinu dostupnosti i performansi što čini MongoDB pogodnim za upravljanje velikim sadržajem, analitiku podataka i IoT (engl. *Internet of Things*) aplikacije.

7.3. Usporedba izrađenih primjera

Primjeri koji su prikazani u ovom završnom radu izrađeni su upravo u sustavima PostgreSQL i MongoDB s ciljem prikaza upravljanja relacijskim i nerelacijskim bazama podataka. Prije kreiranja baze podataka i svih sastavnih dijelova u oba sustava, objašnjeno je modeliranje podataka ERA modelom za sustav PostgreSQL te odnos modeliranja podataka za nerelacijske baze podataka i E – R modela. Program Hackolade iskorišten je za primjer mogućnosti modeliranja nerelacijske baze podataka temeljem sheme podataka SQL jezika te je ručno kreiran model podataka za bazu Hrvatskog konjičkog saveza u sustavu MongoDB. Nadalje, za oba sustava prikazano je i pojašnjeno kreiranje korisnika te rad s korisnicima koje se u sustavima PostgreSQL i MongoDB koristi s istim ciljem – sprječavanje neautoriziranog pristupa podacima i posljedično predstavlja korak prema osiguravanju visoke razine sigurnosti baze podataka koje je od velikog značaja u današnje vrijeme dostupnosti podataka.

U oba su sustava kreirane baze podataka Hrvatskog konjičkog saveza i uneseni su odgovarajući podaci kojima se kasnije upravljalo. Kreiranje baza, relacija i kolekcija te unos instanci entiteta i dokumenata jednostavno je i intuitivno u oba sustava. U radu sa sustavom

MongoDB prikazane su i naredbe za ispisi, ažuriranje i brisanje podataka koje su vrlo lako razumljive, a u grafičkom su sučelju još jednostavnije.

Nadalje, u PostgreSQL sustavu prikazani su jednostavni i složeni upiti kreirani nad bazom podataka koji služe ispisu rezultata temeljem određenih uvjeta vezanih za HKS. Postavljanje jednostavnih upita u PostgreSQL sustavu temeljilo se na odabiru atributa tablice koji su potrebni u rezultatu te postavljanju uvjeta u klauzuli WHERE temeljem kojih se uključuju odnosno isključuju redovi u rezultatu. Kreirani složeni upiti nešto su kompliciraniji s obzirom da je potrebno povezivati tablice temeljem primarnih i vanjskih ključeva, što je realizirano korištenjem naredbe JOIN, te zatim ponovno uvjetima uključiti redove koji su potrebni u ispisu rezultata. U sustavu MongoDB također su kreirani upiti nad kolekcijama koji rezultiraju ispisom zadovoljavajućih dokumenata, a samo pisanje upita bilo je nešto kompliciranije od upita u PostgreSQL-u s obzirom na početak učenja upravljanja podacima u sustavu MongoDB. Jednostavniji upiti realizirani su različitim operatorima filtriranja i sortiranja unutar naredbe *find()* uz korištenje logičkih operatora za složenije uvjete. Pisanje upita u sustavu MongoDB Shell nije značajno zahtjevnije od upita u PostgreSQL sustavu kada se prilagodi sintaksi, a grafičko sučelje MongoDB Compass značajno pojednostavljuje postavljanje upita i razumijevanje same sintakse sustava. Da bi se u sustavu MongoDB postavljali složeniji upiti, prvotno je pojašnjena naredba *db.collection.aggregate(),* faze agregacije te agregirajuće funkcije. Najzahtjevniji dio agregiranja dokumenata u sustavu zapravo je shvaćanje velikog broja različitih faza i agregirajućih funkcija koje sustav nudi kako bi se postavljali najoptimalniji upiti. S obzirom da izuzetno veliku ponudu faza agregacijskog cjevovoda i agregirajućih funkcija, za rad s istima u MongoDB-u potrebno je dobro i dugo iskustvo. Složeniji upiti, kojima su povezivane kolekcije i pružena je mogućnost ispisa podataka iz različitih dokumenata u različitim kolekcijama, postavljani su u fazi *$lookup* koja je dodatno mogla sadržavati i agregacijski cjevovod.

U sustavu PostgreSQL prikazano je korištenje nešto zahtjevnijih objekata, konkretno funkcija i okidača, a u MongoDB Shell-u i MongoDB Compass-u prikazan je rad s pogledima. Transakcije u sustavu PostgreSQL objašnjene su u razradi SQL jezika s obzirom na svoju jednostavnost, a u sustavu MongoDB posvećeno im je više pažnje. Ukazano je na različite odlike transakcija u dva sustava te atomnost i kreiranje distribuiranih transakcija unutar sesija u sustavu MongoDB koje ipak pruža mogućnost kreiranja transakcija. Uz razumijevanje pojmova sesija i transakcija u kontekstu sustava MongoDB, samo kreiranje transakcija unutar sesija, izvršavanje ili prekidanje istih ne predstavlja zahtjevan problem.

8. Zaključak

Usporedba sustava PostgreSQL i MongoDB prikazuje dva vrlo različita načina upravljanja bazama podataka pri čemu svaki nudi prednosti, ali sadrži i nedostatke. Sustav PostgreSQL izuzetno je popularan sustav za upravljanje relacijskim bazama podataka kojeg odlikuje visoka kompatibilnost s SQL standardima i napredne funkcionalnosti upravljanja podacima koje se konstantno nadograđuju. S druge strane, MongoDB najpopularniji je sustav za upravljanje nerelacijskim bazama podataka, a orijentiran je na rad s dokumentnim bazama podataka. Dokumenti pisani u JSON formatu podataka čine nerelacijske podatke u sustavu MongoDB jasno razumljivima i čitljivima, a lako upravljanje promjenjivim skupovima podataka čini MongoDB pogodnim za upravljanje velikim podacima (engl. *Big Data*) koji su u današnje vrijeme napretka tehnologije sve prisutniji i značajniji.

Primjeri u oba sustava za upravljanje bazama podataka pružili su mogućnost učenja o upravljanju različitim tipovima baza podataka. Nakon izrade i usporedbe relacijske i nerelacijske baze podataka, produbilo se razumijevanje korisnosti obje vrste baza podataka, razlog porasta popularnosti nerelacijskih baza te shvaćanje potrebe odabira prikladne vrste baze podataka i SUBP ovisno o modelu i svrsi upravljanja podacima. Implementacija i upravljanje bazom podataka u sustavu MongoDB predstavljala je nešto potpuno novo, ali je uz poznavanje rada u sustavu PostgreSQL i korištenje raznih materijala i dokumentacije sustava pružila nova znanja i dragocjena iskustva rada u MongoDB, omogućavajući uspješnu primjenu i savladavanje različitih naredbi na jednostavnijim primjerima te postepeno razumijevanje kompleksnijih aspekata sustava za upravljanje dokumentnim bazama podataka.

Iako nije moguće odrediti bolji ili općenito najbolji sustav za upravljanje bazama podataka, PostgreSQL i MongoDB s razlogom drže visoka mjesta na ljestvici popularnosti SUBP. Odabir između dva sustava ovisi o modelu podataka te specifičnim potrebama pojedinog projekta i prema tome oba sustava mogu biti idealno rješenje ukoliko se koriste u pravim situacijama.

Popis literature

Maleković, M., i Rabuzin, K. (2016). *Uvod u baze podataka*. Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu.

Rabuzin, K. (2011). *Uvod u SQL*. Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu.

Rabuzin, K. (2014). *SQL – Napredne teme*. Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu.

Hills, T. (2016). *NoSQL and SQL Data Modeling*. Basking Ridge: Technics Publications.

Sadalage, P., J. i Fowler, M. (2015). *NoSQL Distilled: a brief guide to the emerging world of polyglot persistence*. Indiana, USA: Addison-Wesley.

International Business Machines [IBM] (bez dat.) *The relational database.* Preuzeto 10.06.2024. sa https://www.ibm.com/history/relational-database

Beaulieu, A. (2009). *Learning SQL*. Sebastopol, CA, USA: O'Reilly Media, Inc.

Groff, J., R. i Weinberg, P., N. (1999). *The Complete Reference SQL.* Berkeley, CA, USA: Osborne/McGraw-Hill.

Maleković, M. i Schatten, M. (2017). *Teorija i primjena baza podataka.* Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu.

Geeks for Geeks [Slika] (22.05.2024.). Preuzeto 02.06.2024. sa https://www.geeksforgeeks.org/sql-ddl-dml-tcl-dcl/

Ramakrishnan, R. i Gehrke, J. *Database Management Systems*. Preuzeto 11.06.2024. sa https://xuanhien.wordpress.com/wp-content/uploads/2011/04/database-management-systems-raghu-ramakrishnan.pdf

Maleković, M. i Rabuzin, K. (bez dat.). *BAZE PODATAKA II*. Baze podataka 2 [Moodle]. Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin

Darko, G. (2018). *Referencijalni integritet baze podataka* (Završni rad). Varaždin: Sveučilište u Zagrebu, Fakultet organizacije i informatike. Preuzeto sa https://urn.nsk.hr/urn:nbn:hr:211:885941

ResearchGate [Slika] (bez dat.) Preuzeto 10.06.2024. sa https://www.researchgate.net/figure/A-representation-of-key-value-stores_fig1_324922396

Redis [Slika] (bez dat.) Preuzeto 10.06.2024. sa https://redis.io/glossary/graph-database/

DataCamp [Slika] (bez dat.) Preuzeto 10.06.2024. sa https://campus.datacamp.com/courses/nosql-concepts/column-family-databases?ex=1

Medium [Slika] (25.09.2018.) Preuzeto 11.06.2024. sa https://medium.com/@mark.rethana/introduction-to-nosql-databases-c5b43f3ca1cc

PostgreSQL (bez dat.) *About PostgreSQL*. Preuzeto 12.06.2024. sa https://www.postgresql.org/about/

MongoDB (bez dat.) *About MongoDB*. Preuzeto 11.06.2024. sa https://www.mongodb.com/company

Stojanović, A. (2016). Osvrt na NoSQL baze podataka – četiri osnovne tehnologije, *POLYTECHNIC & DESIGN, Tehničko veleučilište u Zagrebu*, DOI: 10.19279/TVZ.PD.2016-4-1-06

Taylor, P. (14.09.2023.) *Ranking of the most popular database management systems worldwide, as of September 2023*. Statista. Preuzeto 11.06.2024. sa https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

Gazda, S. (2024). *Baza podataka za konjički klub Arion* (Projekt iz kolegija Baze podataka 2). Fakultet organizacije i informatike, Varaždin, Sveučilište u Zagrebu.

MongoDB (bez dat.) *MongoDB Manual*. Preuzeto 18.06.2024. sa https://www.mongodb.com/docs/manual/

AlphaCodingSkills [Slika] (bez dat.) Preuzeto 27.06.2024. sa https://www.alphacodingskills.com/postgresql/notes/postgresql-keyword-join.php

Popis slika

Popis tablica