Analiza denormalizirane relacijske baze podataka

Cvrlje, Franko

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

ANALIZA DENORMALIZIRANE RELACIJSKE BAZE PODATAKA

Franko Cvrlje

Split, rujan 2024.

„Zahvaljujem se profesorici Antoneli Prnjak na ukazanom povjerenju i prihvaćanju mentorstva, kao i za veliku pomoć i podršku tijekom studiranja i izrade završnog rada.“

„Zahvaljujem se obitelji, prijateljima i kolegama na neizmjernoj podršci tijekom studiranja.“

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

ANALIZA DENORMALIZIRANE RELACIJSKE BAZE PODATAKA

Franko Cvrlje

SAŽETAK

Cilj ovog završnog rada je istražiti denormalizaciju baza podataka i usporediti njezine prednosti i nedostatke u odnosu na normalizaciju. Fokus je na usporedbi performansi normaliziranog i denormaliziranog modela, koristeći vrijeme izvršavanja upita kao ključnu metriku. Prikazati će se normalizirani model za učinkovito upravljanje procesima te denormalizirani model za brže dohvaćanje podataka. Usporedbom će se evaluirati korisnost denormalizacije kod optimizacije performansi baze podataka uz zadržavanje funkcionalnosti.

Ključne riječi: tablica, ključ, relacija, normalizacija, PostgreSQL, JOIN, performanse, vrijeme

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 36 stranica, 18 grafičkih prikaza, 16 kôdova, 1 tablicu i 18 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Dr. sc. Monika Mladenović,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Neposredni voditelj:

Antonela Prnjak, mag. educ. inf., *asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Dr. sc. Monika Mladenović,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Divna Krpan, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Antonela Prnjak, mag. educ. inf., *asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **rujan 2024.**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

DENORMALIZED RELATIONAL DATABASE ANALYSIS

Franko Cvrlje

ABSTRACT

The aim of this thesis is to explore database denormalization and compare its advantages and disadvantages in relation to normalization. The focus is on comparing the performance of normalized and denormalized models, using query execution time as a key metric. The thesis will present the normalized model for efficient process management and the denormalized model for faster data retrieval. The comparison will evaluate the usefulness of denormalization in optimizing database performance while maintaining functionality.

Key words: table, key, relation, normalization, PostgreSQL, JOIN, performances, time

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 36 pages, 18 figures,16 codes, 1 table and 18 references

Original language: Croatian

Mentor: **Monika Mladenović, Ph.D.,** *Assistant Professor of Faculty of Science, University of Split*

Supervisor: **Antonela Prnjak, mag. educ. inf.,** *assistant at the Faculty of Science, University of Split*

Reviewers: **Monika Mladenović, Ph.D.,** *Assistant Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D., *Assistant Professor of Faculty of Science, University of Split*

Antonela Prnjak, mag. educ. inf., *assistant at the Faculty of Science, University of Split*

Thesis accepted: **September, 2024.**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni rad s naslovom ANALIZA DENORMALIZIRANE RELACIJSKE BAZE PODATAKA izradio samostalno pod voditeljstvom Antonele Prnjak, mag. educ. inf. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju završnog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u završnom radu na uobičajen, standardan način citirao sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student

Franko Cvrlje

Sadržaj:

Uvod

U današnjem modernom dobu, gdje količina podataka i njihova složenost iz dana u dan sve više raste, jedan od najvećih izazova predstavlja organizacija te učinkovito upravljanje tim podacima. Kako neke poslovne organizacije mogu iz dana u dan rasti, tako paralelno s njima raste količina te složenost podataka koje je potrebno obraditi. No, kako se podaci znaju s vremenom nagomilavati, kao posljedica toga dolazilo bi do stvaranja redundantnih podataka, odnosno duplikata, te bi ažuriranja istih bila znatno složenija.

Kao rješenje tog problema javlja nam se pojam „normalizacije“. Uvođenjem ovog procesa ne samo da se smanjuje pojavljivanje duplikata, nego nam omogućava olakšano održavanje i upravljanje podacima. Normalizacija podataka tako predstavlja ključan korak kod dizajna kvalitetnih baza podataka, čime se problemi koji mogu nastati u složenim sustavima za upravljanje podataka mogu znatno brže riješiti, te također na više načina. No, kako normalizacija raspodjeljuje podatke u više tablica, često je potrebno izvršiti više spajanja (JOIN), što može povećati složenost upita i smanjiti performanse, te je bilo potrebno smisliti alternativno rješenje za ovaj problem. Tada se denormalizacija javlja kao ključ za rješenje ovog problema.

Dok proces normalizacije nastoji smanjiti pojavu redundantnih podataka, denormalizacija ih uvodi s ciljem poboljšanja performansi te smanjenja broja potrebnih spajanja. Ovaj proces ubrzava manipulaciju podacima, ali zahtijeva pažljivo balansiranje, jer povećana redundancija može uzrokovati neslaganja i složenije održavanje, što može dovesti do povratka na izvorne probleme.

Cilj ovog završnog rada je upoznati se s pojmom denormalizacije podataka, njezinim karakteristikama, prednostima i nedostacima, te na praktičnom primjeru usporediti normalizirani i denormalizirani model baze podataka. Fokusirat ćemo se na usporedbi performansi obaju modela baze podataka u sustavu za upravljanje relacijskim bazama podataka kroz izvršavanje SQL upita, gdje ćemo kao metriku za mjerenje performansi koristiti vrijeme izvršavanja. Ovim radom ćemo prikazati kako proces denormalizacije može biti koristan kod upravljanja i održavanja podataka u velikim poslovnim organizacijama.

1. Baze podataka

Baza podataka je organizirana, strojno čitljiva zbirka simbola, koja se tumači kao istinit prikaz nekog poslovnog pothvata. Baza podataka je također strojno ažurirajuća, te stoga mora biti i zbirka varijabli. Baza podataka je obično dostupna zajednici korisnika, s različitim zahtjevima [1]. Ukratko, baze podataka bi bile nekakve zbirke podataka koji su kvalitetno strukturirani te strogo organizirani, omogućavajući lako pristupanje potrebnim informacijama te bržu i učinkovitiju manipulaciju podacima. One predstavljaju temeljne komponente informacijskih sustava, te su ključna stavka u raznim poslovnim procesima, gdje je upravljanje velikim količinama složenih podataka nužna za ispravno funkcioniranje.

1.1. Povijest baza podataka

U ranijim danima razvoja tehnologije, podaci su se pohranjivali na diskovima i magnetnim trakama. Kao početak razvoja baza podataka možemo uzeti početak 1960-ih, kada je IBM (engl. *International Business Machines Corporation*) predstavio jedan od prvih hijerarhijskih sustava za upravljanje bazama podataka pod nazivom IMS (engl. *Information Management System*) [2]. Revolucija u području baza podataka dogodila se 1969. godine, kada je E. F. Codd predložio razvoj relacijskog modela. Njegov seminarski rad na temu „*A Relational Model of Data for Large Shared Data Banks*“ postavio je kamen temeljac za razvoj relacijskih baza podataka [2]. Kada je E. F. Codd razvio svoju teoriju relacijskih baza podataka, tražio je pristup koji bi zadovoljio što širi spektar korisnika i primjena [1].

U kasnijem razdoblju, tijekom 1980-ih i 1990-ih, uvidio se značajan porast u korištenju relacijskih baza podataka. Tržišna upotreba relacijskih sustava dovela je do porasta njihove upotrebe i popularnosti, pri čemu je SQL postao standardni jezik za baze podataka [2]. Došlo je do razvoja danas mnogo poznatih sustava za upravljanje relacijskom bazama podataka, poput Oracle, Microsoft SQL Server, MySQL i drugih.

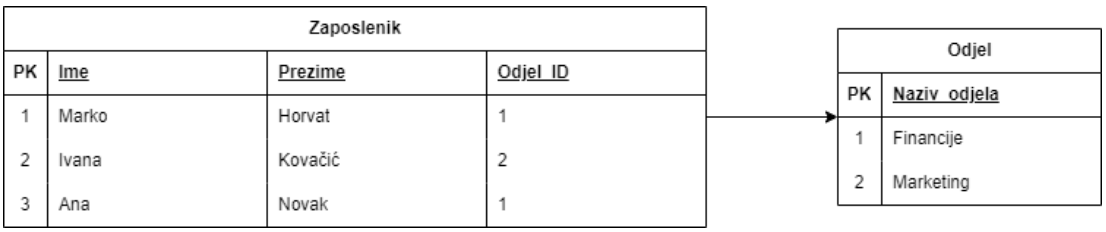
U današnje vrijeme, baze podataka kontinuirano napreduju, kompatibilnije su sa sve više modernih tehnologija, pružajući podršku za velike količine podataka, te za ključno funkcioniranje informacijskih sustava i poslovnih procesa.

1.2. Modeli baza podataka

Modeli baza podataka su od iznimne važnosti u sustavima za upravljanje bazama podataka, zbog toga što nude različite načine za strukturiranje i organizaciju podataka.

1.2.1. Relacijski model

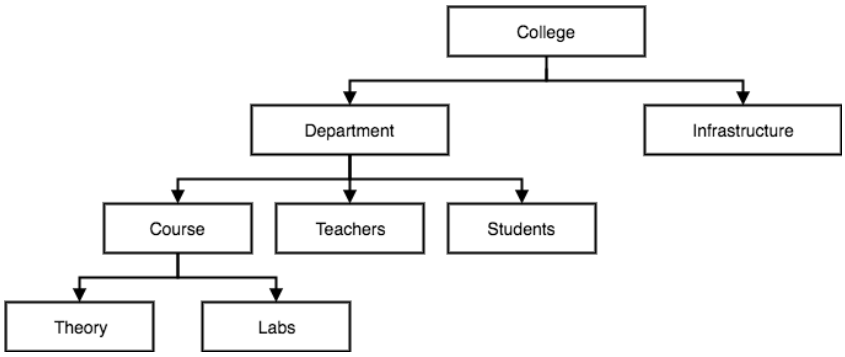
Najčešći model, relacijski model, razvrstava podatke u tablice, također poznate kao relacije, od kojih se svaka sastoji od stupaca i redaka [3]. Svaki stupac, odnosno atribut, ima svoj naziv i tip podatka, primjerice ime, gdje će u tablici u tom stupcu biti izlistana sva unesena imena u tu tablicu u bazi podataka. Neki određeni atribut ili skup atributa može predstavljati primarni ključ, koji odražava jedinstvenost zapisa u tablici, te spajanjem sa stranim ključem iz druge tablice omogućava stvaranje veza (relacija) među tablicama. Jedan redak u tablici sadrži podatke o jednoj instanci entiteta, primjerice o zaposleniku u tvrtki. Nešto više o ovome modelu će biti rečeno u zasebnom poglavlju.



Slika 1 Logički model relacijske baze podataka

1.2.2. Hijerarhijski model

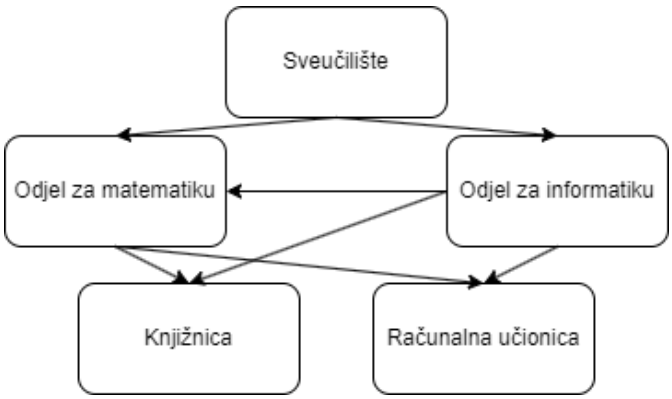
Hijerarhijski model organizira podatke u strukturu poput stabla, gdje svaki zapis ima jednog roditelja ili korijen. Srodni zapisi sortirani su određenim redoslijedom, koji se koristi kao fizički redoslijed za pohranu baze podataka [3]. Ovaj model su u samim počecima koristili IBM-ovi IMS sustavi tijekom 60-ih i 70-ih godina, zbog toga što je ovaj model bio iznimno dobar za opisivanje mnogih odnosa u stvarnom svijetu, no danas se rijetko pojavljuju iz razloga određenih neučinkovitosti.



Slika 2 Hijerarhijski model baze podataka (preuzeto iz [4])

1.2.3. Mrežni model

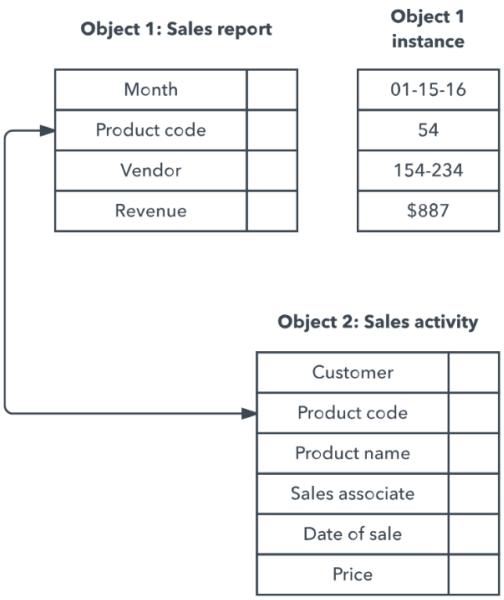
Mrežni model temelji se na hijerarhijskom modelu dopuštajući odnose više-prema-više između povezanih zapisa, što podrazumijeva više nadređenih zapisa [3]. Ovaj model je konstruiran u obliku skupova povezanih zapisa, gdje se svaki skup sastoji od jednog roditeljskog zapisa te jednog ili više podređenih zapisa. Također jedan zapis može biti član u više skupova, što ovome modelu daje više fleksibilnosti te mu omogućuje prenošenje složenih odnosa. No ipak treba imati na umu da unatoč ovim prednostima se opet mogu javiti određeni problemi kod modeliranja zahtjevnijih odnosa.



Slika 3 Mrežni model baze podataka

1.2.4. Objektno orijentirani model

Ovaj model definira bazu podataka kao zbirku objekata ili softverskih elemenata koji se mogu ponovno koristiti, s pripadajućim značajkama i metodama [3]. Zanimljivost kod ovog modela baze podataka jest ta da uključuje tablice, ali nije ograničen samo na njih, nego može kombinirati više pristupa, poput relacijskih te NoSQL, što ga čini hibridnim modelom baze podataka.



Slika 4 Objektno orijentirani model baze podataka (preuzeto iz [5])

1.2.5. Nerelacijski (NoSQL model)

Nerelacijski model baza podataka, za razliku od relacijskih baza podataka, dizajniran je za rad s izuzetno velikim količinama nestrukturiranih ili polustrukturiranih podataka. NoSQL baze podataka dolaze u različitim vrstama na temelju njihovog modela podataka, od kojih među glavne spadaju: dokument, ključ-vrijednost, široki stupac i grafikon. NoSQL modeli pružaju fleksibilne sheme i lako se skaliraju s velikim količinama velikih podataka i velikim korisničkim opterećenjem [6].

Naravno, ovi navedeni modeli nisu jedini modeli koji se mogu koristiti u izradi baza podataka. Postoji još raznih modela baza podataka, no za potrebe ovog završnog rada glavni fokus se stavlja na relacijskom modelu baza podataka, kojega ćemo se dotaknuti više u sljedećem poglavlju.

2. Relacijske baze podataka

Relacijske baze podataka predstavljaju jedan od najvažnijih i najraširenijih sustava za upravljanje podacima različitih vrsta i složenosti, s primjenom u raznim područjima. Korištenjem tablica kao osnova za spremanje podataka omogućuju korisnicima te raznim organizacijama jednostavniju manipulaciju velikim količinama informacija te intuitivniju obradu i pristup samima. Iz tog razloga ovaj model baze podataka nije nepoznanica u današnjem tehnološkom svijetu. Zbog svoje fleksibilnosti kod upravljanja i obrade informacija u širokoj su primjeni u raznim industrijama, od IT-ja, telekomunikacija, zdravstva, pa sve i do trgovina i maloprodaje. Zahvaljujući svim mogućnostima koje ovaj model baze podataka nudi, može se reći da je zapravo jedan od najmoćnijih sustava za upravljanje bazama podataka.

2.1. Definicija i ključni pojmovi

Relacijska baza podataka je vrsta baza podataka koja omogućava pohranu i pristup podatkovnim točkama koje su međusobno povezane. Relacijske baze podataka temelje se na relacijskom modelu, koji je intuitivan i jednostavan način predstavljanja podataka u tablicama. U relacijskoj bazi podataka, svaki redak u tablici predstavlja zapis s jedinstvenim ID-om koji se naziva ključ. Stupci u tablici sadrže atribute koji opisuju podatke, a svaki zapis obično ima vrijednost za svaki atribut, što olakšava uspostavljanje odnosa među podatkovnim točkama [7].

2.1.1. Tablica (Entitet)

Tablica (entitet) u području baza podataka predstavlja osnovnu strukturu za pohranu podataka. Organizacija podataka u tablice omogućuje jednostavno te logičko pohranjivanje istih, te nam također omogućuje lako dohvaćanje tih podataka te upravljanje njima.

Kao primjer možemo navesti tablicu pod imenom „Zaposlenici“, koja će sadržavati podatke o zaposlenicima koji rade u nekoj trgovini. Ta tablica će se sastojati od stupaca i redaka, na koje ćemo se osvrnuti u sljedećim potpoglavljima.

2.1.2. Atribut (Stupac)

Atribut u bazi podataka bi predstavljao stupac u tablici čija je svrha opisivanje svojstva entiteta koji je pohranjen u tablici. Svaki atribut može biti definiran bilo kojim dostupnim tipom podataka.

Kao primjer možemo uzeti da imamo tablicu „Zaposlenici“, gdje atributi mogu biti „Ime“, „Prezime“, „Odjel“, „Datum_Zaposlenja“. Primjerice, atribut „Ime“ sadrži popisana sva imena zaposlenika koja postoje u bazi podataka.

2.1.3. Zapis (Redak)

Zapis u bazi podataka možemo jednostavno prikazati kao redak u tablici. Drugim riječima, jedan redak u tablici bi predstavljao jednu instancu entiteta koji je pohranjen u toj tablici. Svaki zapis je definiran specifičnim vrijednostima koje su definirane u tablici kao atributi, o kojima ćemo više u sljedećem poglavlju.

Za primjer ćemo uzeti jedan zapis u već navedenoj tablici „Zaposlenici“ koji će sadržavati sljedeće podatke o zaposleniku (npr. Ana, Anić, Kozmetika, 2020-01-15…).

2.1.4. Ključevi i veze

Ključevi u bazama podataka bi bili specijalni atributi, čija je glavna svrha identifikacija zapisa te stvaranje relacija (odnosa) među različitim tablicama. Dvije glavne vrste ključeva koje postoje su primarni ključ (engl. *Primary Key*) i strani ključ (engl. *Foreign Key*).

Primarni ključ je jedinstveni identifikator za svaki zapis u tablici. Njegova vrijednost uvijek mora biti jedinstvena i nikada ne smije biti prazna. Primjer u bazi podataka bi bio atribut „Zaposlenik_ID“, koji bi predstavljao primarni ključ u tablici „Zaposlenik“, jer svaki zaposlenik je jedinstven te ima svoj jedinstveni ID. Takav ključ često je arbitraran (engl. *Arbitrary Key*), što znači da nema posebno značenje osim identifikacije zapisa. Suprotno tome, prirodan ključ (engl. *Natural Key*) koristi stvarni podatak, poput OIB-a ili broja putovnice, koji može biti podložan promjenama, uzrokovati duplikate te ugroziti privatnost. Arbitrarni ključevi su češće u upotrebi jer nisu podložni promjenama poput stvarnih podataka, što ih čini fleksibilnijima i lakšima za održavanje.

Strani ključ bi bio atribut u tablici koji bi se povezivao s primarnim ključem iz druge tablice, čime se ostvaruje veza između dviju tablica. Na taj način se preko stranog ključa povezuju zapis iz jedne tablice sa zapisom iz druge tablice. Kao primjer ovoga možemo uzeti tablice „Zaposlenici“ i „Odjeli“, gdje atribut „Odjel_ID“ u tablici „Zaposlenik“ može predstavljati strani ključ koji se povezuje s primarnim ključem „Odjel_ID“ u tablici „Odjeli“, čime se ostvaruje veza između tih tablica.

Također možemo navesti i kompozitni primarni ključ, koji se sastoji od dva ili više atributa u tablici. Ovakav tip ključa je koristan u situacijama kada nijedan pojedinačni atribut ne može jedinstveno identificirati podatak u tablici, dok kombinacija više atributa može osigurati da kombinacija vrijednosti iz više stupaca ostane jedinstvena za svaki redak.

Veze (relacije) u relacijskim bazama podataka služe za definiranje odnosa između podataka u različitim tablicama. Ovaj pristup nam omogućuje lakše razumijevanje te shvaćanje modela radi hijerarhijske organizacije podataka, zbog jednostavnijeg određivanja povezanosti između entiteta. Tri glavne vrste veza koje postoje su:

- **Jedan prema jedan (1:1)**, gdje je svaki redak iz jedne tablice povezan s točno jednim retkom u drugoj tablici. Primjerice, tablica „Zaposlenici“ može biti vezana s tablicom „Automobili“, gdje će svaki zaposlenik imati točno jedan poslovni automobil, te tako isti automobil ne može voziti više zaposlenika.
- **Jedan prema više (1:N)**, gdje svaki redak iz jedne tablice može biti povezan s više redaka u drugoj tablici. Kao što je navedena povezanost tablica „Zaposlenici“ i „Odjeli“, te dvije tablice će biti povezane vezom jedan prema više, čime se daje do znanja da svaki odjel može imati više zaposlenika, dok jedan zaposlenik može raditi na samo jednom odjelu.
- **Više prema više (M:N)**, gdje svaki redak u jednom tablici može biti vezan s više redaka u drugoj tablici, i obrnuto. Kao primjer možemo uzeti tablice „Proizvodi“ i „Računi“, gdje se jedna vrsta proizvoda može naći više računa, te se na jednom računu može naći više proizvoda.

Ovakav način korištenja ključeva te definiranja veza između tablica omogućavaju stvaranje složenijih modela podataka te olakšavaju traženje informacija pohranjenih u bazi podataka, te predstavljaju temelj dobro strukturiranih relacijskih baza podataka.

2.1.5. SQL (Structured Query Language)

Structured Query Language (SQL) je standardizirani programski jezik koji se koristi za upravljanje relacijskim bazama podataka i izvođenje različitih operacija nad podacima u njima. U početku stvoren 1970-ih, SQL redovito koriste ne samo administratori baza podataka, već i programeri koji pišu skripte za integraciju podataka i analitičari podataka koji žele postaviti i pokrenuti analitičke upite [8]. Na taj način SQL omogućuje raznim korisnicima stvaranje, modificiranje te upravljanje bazama podataka preko različitih vrsta upita.

SQL upiti i druge operacije imaju oblik naredbi napisanih kao izjave i agregiraju se u programe koji korisnicima omogućuju dodavanje, izmjenu ili dohvaćanje podataka iz tablica baze podataka [8]. Preko tih upita SQL može obavljati razne operacije nad podacima, od kojih ćemo izdvojiti jednostavnije operacije:

- **Stvaranje podataka (CREATE):** ova naredba omogućuje stvaranje nove baze podataka, te novih tablica u postojećoj bazi podataka.

```
CREATE DATABASE Tvrtka;
```

Kôd 1 SQL CREATE DATABASE naredba pomoću koje se kreira nova baza podataka „Tvrtka“

```
CREATE TABLE Zaposlenik (
      ID_Zaposlenik int,
      ime varchar(50),
      prezime varchar(100)
);
```

Kôd 2 SQL CREATE TABLE naredba pomoću koje se kreira nova tablica „Zaposlenik“

- **Dohvaćanje podataka (SELECT):** ova naredba se koristi za dohvaćanje nekog ili svih podataka iz neke tablice. Ovom operacijom se određeni podatak ili više njih pretražuje u bazi podataka, koji se vraćaju kao rezultat neke specifične SELECT naredbe.

```
SELECT ime FROM Zaposlenik;
```

Kôd 3 SQL SELECT naredba pomoću koje se iz tablice „Zaposlenik“ dohvaća ime zaposlenika, odnosno podaci iz stupca „ime“

- **Umetanje podataka (INSERT):** naredbom INSERT omogućeno je dodavanje novih podataka u tablicu baze podataka.

```
INSERT INTO Zaposlenik (
        ime,
        prezime )
VALUES (
        'Marko',
        'Grčić' );
```

Kôd 4 SQL INSERT naredba pomoću koje se u tablicu „Zaposlenik“ unose podaci

- **Ažuriranje podataka (UPDATE):** naredba koja služi za izvršavanje promjena redaka s podacima u nekoj tablici, ako se naknadno odluči promijeniti neke podatke u bazi podataka.

```
UPDATE Zaposlenik
SET prezime = 'Jurič',
WHERE prezime = 'Jurić';
```

Kôd 5 SQL UPDATE naredba pomoću koje se mijenja vrijednost u tablici „Zaposlenik“ u stupcu „prezime“

- **Brisanje podataka (DELETE):** naredba koja se koristi u svrhu uklanjanja redaka iz određene tablice, ako te podatke više nije potrebno čuvati u bazi podataka.

```
DELETE FROM Zaposlenik WHERE prezime = 'Jurič';
```

Kôd 6 SQL DELETE naredba pomoću koje se uklanjaju svi zaposlenici sa prezimenom „Jurič“

2.2. Karakteristike relacijskih baza podataka

Relacijske baze podataka osiguravaju dosljednost i fleksibilnost u radu s velikim količinama podataka te predstavljaju osnovu za rad u različitim industrijama, zahvaljujući [7] [9]:

- **SQL-u i upitima:** relacijske baze podataka koriste SQL kao standardni jezik za upravljanje podacima, čime se omogućuje jednostavnije i efikasnije izvođenje različitih vrsta upita nad podacima, te očuvanje njihove točnosti i dosljednosti.
- **Normalizaciji podataka:** korištenjem postupka normalizacije omogućuje se smanjivanje pojave duplikata među podacima te se omogućuje njihovo jednostavnije održavanje. O ovom postupku će se govoriti više u nastavku rada.
- **Referencijalnom integritetu:** održavanje referencijalnog integriteta predstavlja glavnu stavku kod očuvanja dosljednosti veza među tablicama. Primjerice, ako se neki zapis iz jedne tablice briše, baza podataka može automatski izbrisati ili ažurirati

zapise u drugoj tablici koji su povezani sa zapisima iz prve, kako bi se očuvao integritet podataka.

- **Organizaciji podataka:** relacijske baze podataka omogućuju strukturiranu pohranu podataka u redove i stupce, što olakšava upravljanje, pretraživanje i logičko povezivanje podataka.
- **ACID svojstvima:** relacijske baze podataka podržavaju ACID (engl. *Atomicity, Consistency, Isolation, Durability*) svojstva. Ona osiguravaju pouzdanost te integritet podataka, tako da osiguravaju da sve transakcije u bazi podataka budu pravilno izvršene, čak i unatoč pogreškama koje se mogu pojaviti.

No, unatoč ovim prednostima relacijskih baza podataka, moramo imati na umu da postoje i ograničenja o kojima je potrebno voditi računa prilikom početka rada s relacijskim bazama podataka, kao što su [9]:

- **Složenost upita:** Kako baza raste, upiti postaju složeniji, posebno kod spajanja (JOIN), što može usporiti performanse.
- **Rad s nestrukturiranim podacima:** Relacijske baze nisu idealne za nestrukturirane podatke (npr. tekstove ili slike), gdje su NoSQL baze fleksibilniji izbor.
- **Održavanje normalizacije:** Normalizacija smanjuje duplikate, ali povećava složenost baze i zahtijeva više spajanja, što može usporiti upite.
- **Teže skaliranje:** proces skaliranja predstavljalo bi postupak povećanja kapaciteta sustava kako bi se moglo jednostavnije upravljati s velikom količinom podataka [10]. Skaliranje relacijskih baza, posebno horizontalno, može biti skupo i izazovno, te utjecati na performanse.

Unatoč ograničenjima koja se javljaju prilikom rada s relacijskim bazama podataka, one ipak posjeduju više prednosti kod njihovog korištenja, te nisu bez razloga među prvim izborima za pohranu i upravljanje podacima.

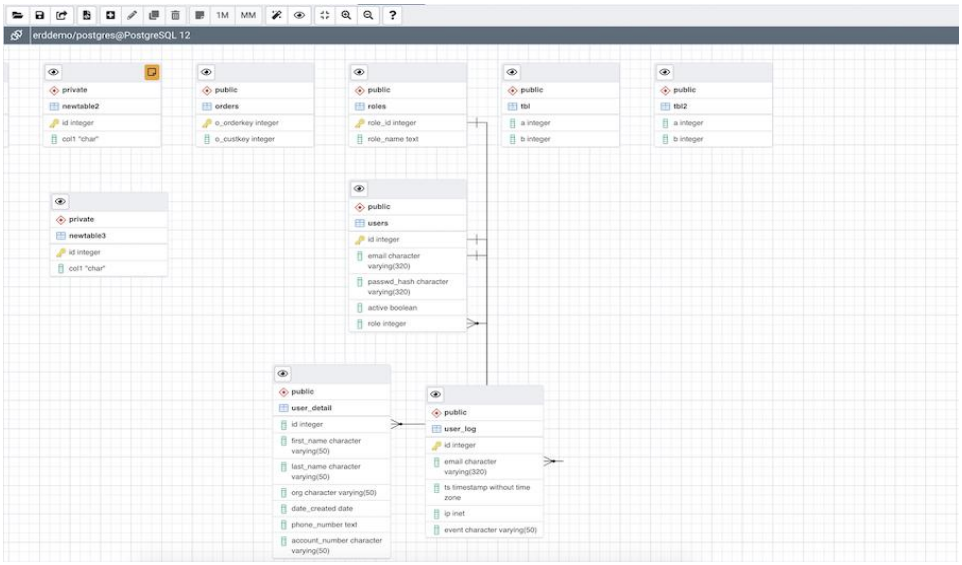
2.3. PostgreSQL

PostgreSQL predstavlja snažan sustav objektno-relacijskog upravljanja bazama podataka otvorenog koda. On koristi te proširuje SQL jezik kombinirajući ga s brojnim značajkama koje omogućuju sigurno pohranjivanje i skaliranje najsloženijih radnih opterećenja s podacima. PostgreSQL datira već od 1986. godine kao dio projekta POSTGRES na

Kalifornijskom Sveučilištu u Berkeleyju, s više od 35 godina aktivnog rada i razvoja na osnovnoj platformi [11]. PostgreSQL je razvijen kako bi pružio pouzdanost te fleksibilnost u radu s podacima, čime postaje jedan od idealnijih izbora za široko područje IT-ja. Kod brojnih tvrtki upotrebljava se za rad poslovnih aplikacija, analizu podataka, kod backend dijela web aplikacija te u raznim sustavima.

PostgreSQL je moćan sustav koji podržava brojne napredne tipove podataka, poput JSON, XML te prostorne (GIS) podatke. Usklađen je s ACID svojstvima što osigurava pouzdanost transakcija, te njegova proširivost pruža podršku za proceduralne jezike poput PL/pgSQL i Python. Unatoč tome što je prvenstveno relacijski sustav, on nudi i NoSQL značajke, čime omogućuje rad s nestrukturiranim podacima. Sposobnost skaliranja te sigurnosne značajke pružaju visok stupanj sigurnosti, te aktivna zajednica kojoj se može pridružiti te obratiti za pomoć dodatno osnažuje njegovu poziciju jednog od najpouzdanijih sustava za upravljanje bazama podataka [11].

Mnoge prednosti i značajke ovog sustava za upravljanje bazama podataka bile su ključ kod odabira sustava za izradu praktičnog dijela ovog završnog rada.



Slika 5 Prikaz ER dijagrama u PostgreSQL (preuzeto iz [12])

3. Normalizacija baza podataka

Razne poslovne organizacije, kao što su primjerice informatičke tvrtke, za cilj imaju da se podaci mogu hijerarhijski organizirati te da baza podataka bude jednostavna za održavati. Kada ne bi postojala nekakva pravila za oblikovanje relacijskog modela baze podataka, kao problem bi se javljala redundancija, odnosno pojavljivanje dupliciranih podataka, čime bi baza podataka bili iznimno kompliciranа za upravljanje i održavanje. Također, raspored tih podataka u većini slučajeva ne bi imao nikakvog smisla, te ne bi logički mogli razabrati što je s čime povezano. Kao rješenje toga problema javlja se pojam „normalizacije“ baza podataka, koji predstavlja ključni proces u dizajnu relacijskih sustava za manipulaciju podacima. Taj proces obuhvaća niz koraka, poznatih kao normalne forme, koje će postupno uklanjati različite anomalije koje se mogu pojavljivati, čime optimiziraju strukturu baze podataka smanjenjem redundantnosti podataka te osiguravanjem njihovog integriteta.

3.1. Definicija i karakteristike

Normalizacija baza podataka predstavlja proces korišten kod dizajna relacijskih baza podataka s ciljem učinkovite organizacije podataka te smanjivanja redundancije istih, dok se istovremeno osigurava integritet podataka. Ovaj proces uključuje razbijanje velikih tablica u manje, povezane tablice te definiranje odnosa među njima. Glavni ciljevi postupka normalizacije baze podataka su eliminacija anomalija podataka, smanjenje redundantnih podataka i olakšavanje upravljanja bazom podataka [13].

Proces normalizacije baza podataka pruža mogućnost jednostavnijeg održavanja baze podataka koja se koristi u raznim sustavima, zbog toga što su podaci organizirani na način koji smanjuje mogućnost pojave nekonzistentnosti te anomalijama pri promjenama koje vršimo nad njom.

Najbitnija stavka koja se može navesti za proces normalizacije baza podataka jest eliminacija redundantnih podataka. Proces normalizacije kao glavnu svrhu ima smanjivanje, ili u nekim slučajevima uklanjanje dupliciranih podataka unutar tablica, što rezultira u smanjivanju potrebe za višestrukim unosom podataka, čime se smanjuje mogućnost pogrešaka u bazi podataka.

Normalizacija također osigurava da su podaci koji se čuvaju u bazi podataka ispravno pohranjeni u kvalitetno definiranim tablicama s jasno određenim relacijama među njima, što osigurava konzistentnost podataka te točnost informacija kroz cijelu bazu podataka.

Također, implementacijom pravila normalizacije garantira se efikasnije dohvaćanje podataka, što je pogodno u slučajevima kada je potrebno u što kraćem vremenskom roku pristupiti podacima. Iako taj proces može zahtijevati pisanje složenijih upita iz razloga povećanog broja spajanja (JOIN) tablica, on smanjuje potrebu za pohranom viška podataka, čime dugoročno može poboljšati performanse baze podataka.

Normalizacija kao proces pri modeliranju relacijskih baza podataka predstavlja temeljni koncept u njihovom dizajnu, omogućujući stvaranje sustava za upravljanje podacima koji su učinkoviti, održivi i skalabilni.

3.2. Normalizacijske forme

Normalizacijske forme predstavljaju skup pravila, iliti ga normi, koja se primjenjuju prilikom procesa normalizacije podataka s ciljem osiguravanja optimalne organizacije i integriteta podataka. Svaka od tih formi ima određene zahtjeve i ograničenja kojima se postepeno kroz svaku od formi smanjuje redundantnost te pojava anomalija među podacima. Dalje ćemo u ovome poglavlju ćemo objasniti svaku od ovdje navedenih normalnih formi (1NF, 2NF, 3NF, BCNF), te ćemo na primjeru videoteke pokazati kako primjenom ovih normi možemo postići višu razinu normalizacije baze podataka.

Ovdje je prikazan primjer jedne tablice „Zaposlenik “u bazi podataka za tvrtku:

Zaposlenik				
Zap_ID	Ime	Prezime	Odjel	Kvalifikacije
1	Marko	Horvat	Marketing	IT, Marketing
2	Ivana	Kovačić	Financije	Računovodstvo
3	Ana	Novak	Marketing	IT

Slika 6 Tablica "Zaposlenik" u bazi podataka za tvrtku

3.2.1. Prva Normalna Forma (1NF)

Prva normalna forma (1NF) zahtijeva da svi atributi u tablici budu atomarni, odnosno da u svakome stupcu bude samo jedna vrijednost [14]. U tablici „Zaposlenik“ sadržan je stupac

„Kvalifikacije“, gdje za jednog zaposlenika mogu biti navedene više kvalifikacija odjednom (npr. IT, Marketing). Problem koji nastaje u ovakvoj situaciji je što više vrijednosti unutar jednog stupca krši pravilo 1NF i otežava rad s podacima. Primjenom prve normalne forme, svaka kvalifikacija za zaposlenika stavlja se u zaseban redak u tablici, čime se osigurava atomarnost podataka:

Zaposlenik				
Zap_ID	Ime	Prezime	Odjel	Kvalifikacije
1	Marko	Horvat	Marketing	IT
1	Marko	Horvat	Marketing	Marketing
2	Ivana	Kovačić	Financije	Računovodstvo
3	Ana	Novak	Marketing	IT

Slika 7 Tablica "Zaposlenik" nakon primjene 1NF

3.2.2. Druga Normalna Forma (2NF)

Druga normalna forma (2NF) zahtijeva da tablica bude u 1NF i da svi neključni atributi ovise o cijelom primarnom ključu, a ne samo o dijelu tog primarnog ključa [14]. U ovom primjeru, stupac „Kvalifikacije“ ovisi o zaposleniku, a ne o njegovom odjelu, što znači da postoji djelomična ovisnost (Slika 7). Kako bi se to ispravilo i prešlo u 2NF, kvalifikacije se premještaju u zasebnu tablicu, koja je povezana sa zaposlenicima preko njihovog ID-a:

Zaposlenik			
Zap_ID	Ime	Prezime	Odjel
1	Marko	Horvat	Marketing
2	Ivana	Kovačić	Financije
3	Ana	Novak	Marketing

Kvalifikacija	
Zap_ID	Kvalifikacija
1	IT
1	Marketing
2	Računovodstvo
3	IT

Slika 8 Tablice "Zaposlenik" i "Kvalifikacija" nakon primjene 2NF

Ovim načinom se uklanjaju djelomične ovisnosti i podaci su sada potpuno ovisni o primarnom ključu (Slika 8).

3.2.3. Treća Normalna Forma (3NF)

Treća normalna forma (3NF) zahtijeva da tablica već bude u 2NF i da svi neključni atributi ovise izravno o primarnom ključu, a ne o drugom neključnom atributu [14]. U ovom primjeru, stupac „Naziv_odjela“ ovisi o stupcu „Odjel_ID“, a ne izravno o zaposleniku. Kako bi se to ispravilo i prešlo u 3NF, naziv odjela se premješta u zasebnu tablicu koja sadrži informacije o odjelima, čime se uklanjaju tranzitivne ovisnosti i postiže potpuna neovisnost atributa:

Zaposlenik			
Zap_ID	Ime	Prezime	Odjel_ID
1	Marko	Horvat	1
2	Ivana	Kovačić	2
3	Ana	Novak	1

Odjel	
Odj_ID	Naziv_Odjel
1	Marketing
2	Financije

Kvalifikacija	
Zap_ID	Ime
1	IT
1	Marketing
2	Računovodstvo
3	IT

Slika 9 Tablice "Zaposlenik", "Odjel" i "Kvalifikacija" nakon primjene 3NF

Naravno, ovo nisu jedine normalne forme koje postoje. Postoje i neke još više normalne forme, kako bi se postigao još viši normalizirani oblik, no za potrebe ovog završnog rada u više norme se neće ulaziti.

Dizajneri baza podataka koriste ove normalne forme kao smjernice za učinkovito strukturiranje baza podataka, ovisno o specifičnim zahtjevima njihovih aplikacija ili sustava. Iako postizanje viših normalnih formi može smanjiti redundanciju podataka i poboljšati njihov integritet, važno je pronaći ravnotežu između normalizacije i performansi, zbog toga što visoko normalizirane baze podataka mogu zahtijevati složenije upite i više spajanja (JOIN). U praksi, razina normalizacije primijenjena na bazu podataka ovisi o specifičnom slučaju upotrebe i razmatranjima performansi [13].

4. Denormalizacija baza podataka

Kao rješenje problema povećane potrebe za spajanjem tablica te pada performansi javlja se proces potpuno obrnut od procesa normalizacije, pod nazivom „denormalizacija“. Kako se procesom normalizacije nastoji smanjiti broj redundantnih podataka unutar baze podataka, tako proces denormalizacije ponovno uvodi pojavljivanje dupliciranih podataka te spajanja manjih tablica u veće tablice. Zašto bi se ponovno uvodili redundantni podaci? Ponovnim uvođenjem dupliciranih podataka te spajanja tablica performanse baza podataka se optimiziraju, što je ključno u situacijama kada je brzina dohvaćanja podataka kritična, kako bi sustavi mogli ispravno funkcionirati [15].

4.1. Definicija i karakteristike

Denormalizacija podataka je proces uvođenja određene redundancije u prethodno normalizirane baze podataka s ciljem optimizacije performansi upita baze podataka. Uvodi određenu unaprijed izračunatu redundanciju koristeći različite tehnike za rješavanje problema u normaliziranim podacima [16]. Kao što je već spomenuto, dok proces normalizacije teži smanjiti redundantnost podataka te optimizirati integritet baze podataka njihovim razdvajanjem u više tablica, proces denormalizacije svjesno uvodi redundantne podatke ili kombinira tablice kako bi se smanjila složenost upita te kako bi pristup podacima bio dosta brži.

Denormalizacija se koristi u specifičnim slučajevima kada je optimizacija performansi od iznimne važnosti, posebno kada su zahtjevi za brzinom dohvaćanja podataka prioritetni. Ovaj pristup zahtijeva pažljivo planiranje kako bi se ostvarilo postizanje ravnoteže između brzine i održavanja baze podataka. Pri denormalizaciji, važno je imati na umu moguće izazove koji se mogu pojaviti, poput povećane redundancije podataka. Iako je brzo dohvaćanje podataka bitna stavka kod mnogih sustava, opet treba pripaziti da se s procesom denormalizacije ne ode u neku dalju krajnost. Denormalizacija može biti iznimno korisna strategija kod optimizacije performansi, gdje složeni upiti te veliki broj spajanja (JOIN) može negativno utjecati na brzinu i učinkovitost baze podataka, ali uz dovoljan oprez te kombinaciju s drugim metodama dizajniranja relacijskih baza podataka [17].

4.2. Tehnike denormalizacije

Primjena procesa denormalizacije prilikom dizajniranja relacijskog modela uključuje odabir određenih tehnika kako bi se postigla optimizacija performansi baze podataka.

Postoji nekoliko tehnika denormalizacije koje se mogu koristiti u različitim kontekstima ovisno o tome što određeni sustav zahtijeva, poput kombiniranja tablica, dodavanja redundantnih podataka, denormalizacije agregiranih podataka, te mnogih drugih pristupa koji pomažu kod optimizacije performansi.

Prilikom pokazivanja načina na koji tehnike denormalizacije funkcioniraju, izrađene su skice tablica, kako bi se lakše savladao teorijski dio.

4.2.1. Kombiniranje Tablica

Jedna od najčešće korištenih tehnika koja se koristi prilikom denormalizacije jest kombiniranje tablica (engl. *Table Merging*). Ona uključuje spajanje podataka iz više normaliziranih tablica u jednu tablicu s ciljem smanjivanja potrebe za složenijim spajanjima (JOIN) tablica, čime se postiže primarni cilj procesa denormalizacije, optimiziranje performansi baze podataka [16].

Za primjer kombiniranja tablica možemo zamisliti da imamo tablice „Zaposlenici“ i „Odjeli“ u bazi podataka jedne tvrtke:

Zaposlenik			
Zap_ID	Ime	Prezime	Odjel_ID
1	Marko	Horvat	1
2	Ivana	Kovačić	2

Odjel	
Odj_ID	Naziv_Odjela
1	Marketing
2	Financije

Slika 10 Tablice "Zaposlenik" i "Odjel" u bazi podataka

U ovom normaliziranom modelu, za dohvaćanje podataka o zaposlenicima i njihovim odjelima, potrebno je koristiti JOIN operacije između ovih tablica. Kako bi se izbjeglo korištenje JOIN operacija, te dvije tablice ćemo spojiti u jednu veću tablicu

„Zaposlenik_Odjel“. Ovim spajanjem se ubrzavaju upiti koji često dohvaćaju podatke o zaposlenicima i njihovim odjelima, jer sada svi potrebni podaci dolaze iz jedne tablice:

Zaposlenik_Odjel			
Zap_ID	Ime	Prezime	Odjel
1	Marko	Horvat	Marketing
2	Ivana	Kovačić	Financije

Slika 11 Tablica "Zaposlenik_Odjel" nakon primjene tehnike kombiniranja tablica

Kombiniranje tablica je učinkovito kada je struktura podataka relativno jednostavna i kada podaci ne zahtijevaju česta ažuriranja. Međutim, ova tehnika može dovesti do povećanih zahtjeva za prostorom za pohranu, a također postoji rizik od anomalija ako podaci postanu nekonzistentni tijekom ažuriranja.

4.2.2. Dodavanje Redundantnih Podataka

Dodavanje redundantnih podataka (engl. *Adding Redundant Data*), odnosno njihovih duplikata, je tehnika denormalizacije koja isto kao i prethodna tehnika za cilj ima smanjenje broj potrebnih spajanja (JOIN) između tablica, čime se ubrzava dohvaćanje podataka. Ova tehnika je poprilično jednostavna, stvara se duplikat podatka iz jedne tablice, koji će se ubaciti u drugu tablicu kako bi se olakšao pristup ako je taj podatak često korišten, smanjujući složenost upita i vrijeme izvršavanja [16].

Za primjer korištenja ove tehnike možemo uzeti tablice „Zaposlenik“ i „Kvalifikacija“:

Zaposlenik			
Zap_ID	Ime	Prezime	Odjel
1	Marko	Horvat	Marketing
2	Ivana	Kovačić	Financije

Kvalifikacija	
Zap_ID	Kvalifikacija
1	IT
1	Marketing
2	Računovodstvo

Slika 12 Tablice "Zaposlenik" i "Kvalifikacija" u bazi podataka

U normaliziranom modelu, kvalifikacije zaposlenika nalaze se u zasebnoj tablici kako bismo izbjegli duplikate i pojednostavili održavanje podataka. Međutim, ako su informacije o

zaposlenicima i njihovim kvalifikacijama zajedno često potrebne, korištenje JOIN operacija može usporiti upite. Kako bi se ubrzalo izvršavanje upita, kvalifikacije se dodaju direktno u tablicu „Zaposlenik“, duplicirajući te podatke i uklanjajući potrebu za JOIN operacijama.

Zaposlenik				
Zap_ID	Ime	Prezime	Odjel	Kvalifikacija
1	Marko	Horvat	Marketing	IT, Marketing
2	Ivana	Kovačić	Financije	Računovodstvo

Slika 13 Tablica "Zaposlenik" nakon primjene tehnike dodavanja redundantnih podataka

Dodavanje redundantnih podataka ubrzava upite, ali povećava rizik od nekonzistentnih podataka jer svaka promjena kvalifikacija sada mora biti ručno ažurirana u više zapisa. Ova tehnika je korisna u situacijama kada su performanse važnije od preciznog održavanja konzistencije podataka.

4.2.3. Denormalizacija Agregiranih Podataka

Denormalizacija agregiranih podataka (engl. *Precomputed Aggregates*) je tehnika denormalizacije koja uključuje unaprijed izračunate vrijednosti te pohranu tih rezultata u tablici, umjesto da se te vrijednosti računaju svaki put kada se pokrene upit. Primjerice, ako se neki upit izvršava više puta, a njime želimo dohvatiti ukupan broj ili prosjek nekih brojčanih podataka, dodat ćemo novi stupac u tablicu u kojem će se pohranjivati te izračunate vrijednosti. Ovom tehnikom se znatno može smanjiti vrijeme izvršavanja upita, jer nema potrebe da se potrebna vrijednost ponovno računa svaki put kada izvršavamo upit [18].

Za primjer denormalizacije agregiranih podataka ćemo uzeti tablice „Zaposlenici“ i „Odjeli“:

Zaposlenik			
Zap_ID	Ime	Prezime	Odjel_ID
1	Marko	Horvat	1
2	Ivana	Kovačić	2
3	Ana	Novak	1

Odjel	
Odj_ID	Naziv_Odjela
1	Marketing
2	Financije

Slika 14 Tablice "Zaposlenik" i "Odjel" u bazi podataka

U normaliziranom modelu, svaki put kada je potrebno saznati broj zaposlenika na odjelu, koristi se SQL funkcija COUNT(), što može biti sporo ako često koristimo te podatke. Kako bi se ubrzalo izvršavanje takvih upita unaprijed ćemo pohraniti broj zaposlenika u svakom odjelu u tablici „Odjeli“:

Odjel		
Odj_ID	Naziv_Odjela	Broj_zaposlenika
1	Marketing	2
2	Financije	1

Slika 15 Tablica "Odjel" nakon primjene tehnike denormalizacije agregiranih podataka

Ova tehnika poboljšava performanse upita koji često trebaju agregirane podatke, ali također zahtijeva ručno ažuriranje svakog puta kada se promijeni broj zaposlenika u odjelu. To može dovesti do nedosljednosti podataka ako promjene nisu pravilno sinkronizirane.

4.3. Kada i kako primijeniti denormalizaciju?

Denormalizacija baza podataka u nekim slučajevima nije nužna strategija u modeliranju relacijskih baza podataka, no u određenim situacijama može biti ključna za postizanje boljih performansi. Ponovno, cilj kod primjene ovog procesa nije kompletna denormalizacija baze podataka, jer bi to rezultiralo u prezahtjevnom održavanju sustava, velike nepreglednosti te nelogičnosti, čime bi se umjesto postizanja optimiziranih performansi vratili na početni problem usporenosti sustava. Odabir trenutka u kojem ćemo primijeniti postupak denormalizacije ovisi o zahtjevima koje sustav zatražuje, njegovom opterećenju zahtjevima, složenosti upita itd. Stoga je važno znati kada i u kolikoj mjeri je potrebno primijeniti ovaj postupak.

Denormalizaciju bi bilo korisno primijeniti u slučajevima kada [18]:

- **Performanse su prioritet:** proces denormalizacije nam može pomoći u smanjenju vremena izvršavanja složenijih upita, posebno ako je sustav opterećen velikim brojem čitanja podataka te kada brzina upita postaje kritična. Ovo je od iznimne važnosti u aplikacijama koje moraju odgovarati na zahtjeve u stvarnom vremenu, primjerice u analitičkim sustavima te u aplikacijama čiji je ključ brzu obrada velike količine podataka.

- **Postoji visoka učestalost spajanja (JOIN):** kada su u upitima često prisutna spajanja između više tablica kako bi se dohvatili potrebni podaci, to može značajno usporiti performanse baza podataka. Kroz denormalizaciju, više manjih tablica se kombinira u jednu veću tablicu, te se uvode duplikati podataka na potrebna mjesta. Ovim načinom se u samo par tablica smještaju svi potrebni podaci, kako bi se upiti mogli pisati bez potrebe za korištenjem JOIN operacija.
- **Postoje sustavi samo s mogućnosti čitanja:** u takvim sustavima, gdje unos podataka nije moguć, procesom denormalizacije uvodimo redundantne podatke kako bi ubrzali čitanje podataka, zato što je u tim slučajevima cijena procesa denormalizacije prihvatljiva.
- **Postoje složeni i često korišteni upiti:** ako sustav često izvršava složene upite koji uključuju više tablica, te upite koji se višestruko izvršavaju, korištenjem tehnika denormalizacije složenost upita se može znatno smanjiti te se mogu puno brže izvršiti.

Kada zaključimo u kojem trenutku je korisno primijeniti denormalizaciju, bitno je znati i kako je primijeniti. Koraci koji mogu pomoći shvatiti kako primijeniti denormalizaciju su [18]:

- **Identificiranje kritičnih upita:** prvi ključan korak bi bio identificirati koji su to upiti koji najviše opterećuju sustav ili zahtijevaju poboljšanje performansi kako bi se brže izvršili. Najbolji korak bi bio identificirati tablice koje se često koriste u složenim upitima.
- **Odabir tehnike denormalizacije:** nakon identificiranja složenih upita potrebno je odabrati odgovarajuću tehniku denormalizacije, primjerice kombiniranje tablica, dodavanje redundantnih podataka, preagregacija podataka itd...
- **Testiranje:** Kako bi se uvjerili da odabrana tehnika denormalizacije daje željene rezultate, korisno bi bilo testirati performanse na ograničenom skupu podataka te ih usporediti s performansama prije denormalizacije kako bi se uvjerili da promjene donose željene rezultate.
- **Održavanje konzistentnosti:** ključan korak kod primjene denormalizacije bi bio postavljanje dodatnih pravila ili mehanizama za održavanje integriteta kako bi izbjegli nastajanje anomalija zbog redundancije.

- **Kontinuirano praćenje i optimizacija:** nakon implementacije odgovarajuće tehnike denormalizacije važno je kontinuirano pratiti performanse izvršavanja. Kada se baza podataka jednom denormalizira, to ne mora značiti da je kraj. Kako sustavi rastu i razvijaju se, može biti potrebno izvršavati daljnje optimizacije, kako bi sustav ispravno funkcionirao te da bi se osigurao kontinuitet brzog dohvaćanja podataka.

Primjenom denormalizacije u pravom trenutku i na pravi način mogu se postići značajna poboljšanja performansi, ali važno je imati na umu zamke koje se mogu pojaviti pretjeranom primjenom tog procesa.

5. Praktični primjer baze podataka: IT Tvrtka

5.1. Opis problema

IT tvrtka (engl. Information Technology Company) predstavlja kompleksnu poslovnu organizaciju koja se koristi vrlo širokim spektrom informacijsko-komunikacijskih tehnologija za izradu projekata razne vrste i složenosti u svrhu upotrebe u različitim industrijama i svakodnevnim poslovnim procesima.

Tvrtka zapošljava stručnjake specijalizirane za rad u različitim IT domenama, koji rade na projektima usmjerenim prema tehnološkim inovacijama, kao i na tradicionalnim IT te marketinškim uslugama. U tvrtki postoji više raznih odjela, u kojima se nalaze zaposlenici prema svojim kvalifikacijama koje su potrebne na tom odjelu. Ne smijemo isključiti mogućnost da jedan zaposlenik može na neko određeno vrijeme raditi na više odjela, čime se postiže fleksibilnost u poslu.

Tvrtka posluje s raznim klijentima, bilo da je u pitanju druga tvrtka ili pojedinac, koji mogu ovisno o svom proračunu zatražiti izradu jednog ili više projekata, ovisno o potrebnim uslugama. Nakon što klijent zatraži obavljanje određene usluge, formiraju se timovi za izradu projekta. Jedan tim može raditi na više projekata, a također zaposlenici iz različitih odjela mogu surađivati ovisno o potrebama projekta. Nakon završetka izrade projekta, tvrtka izdaje klijentu fakturu koja sadrži status plaćanja i datum dospijeća, prema kojem će klijent isplatiti tvrtku za obavljene usluge. Potrebno je bilo izraditi model koji povezuje zaposlenike s odjelima i timovima, te klijente i njihove usluge s fakturom nakon dovršenih projekata.

5.2. Normalizirani model baze podataka

5.2.1. O modelu

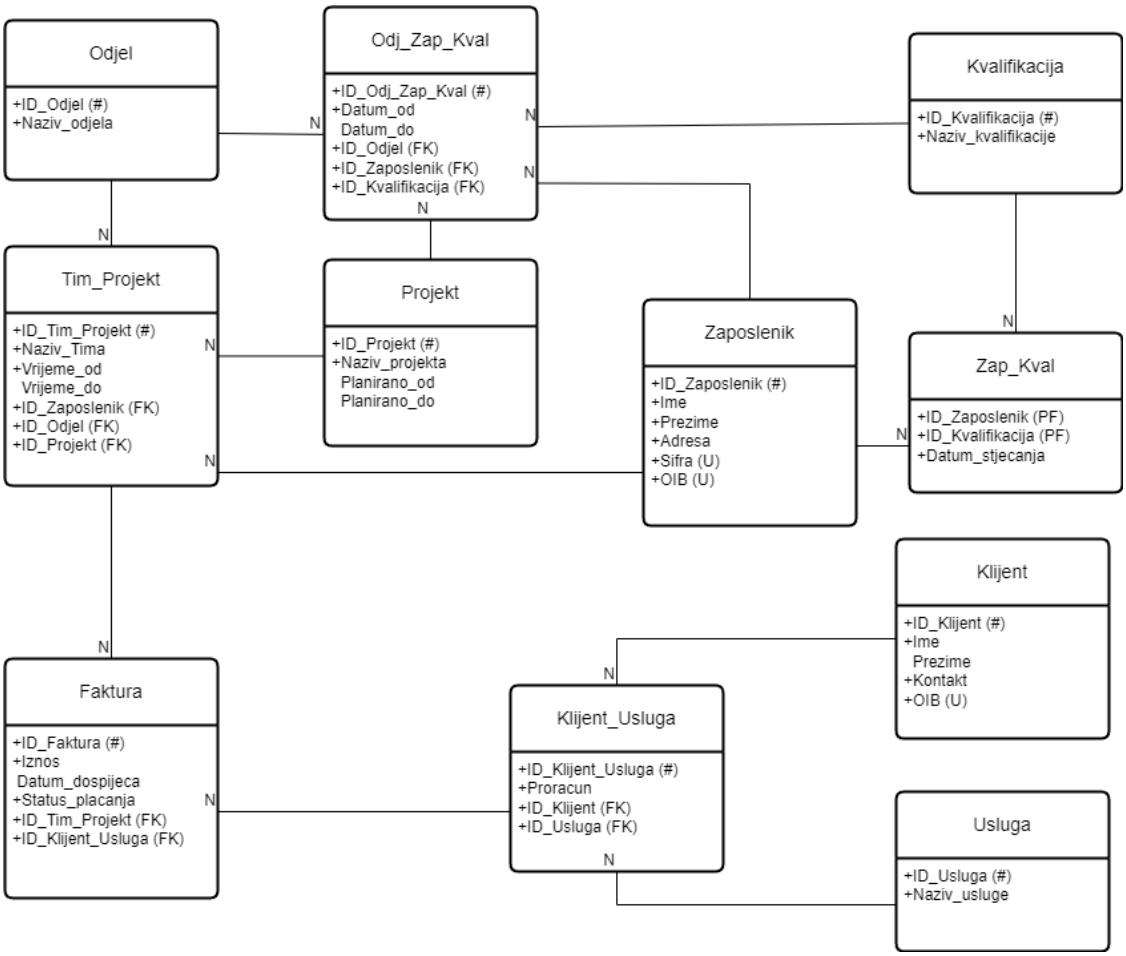
Normalizirani model baze podataka za IT tvrtku temeljen je na principima relacijskog dizajna, s ciljem smanjenja redundantnosti podataka i očuvanja integriteta istih. Podaci su organizirani u 11 tablica, od kojih svaka predstavlja različit entitet unutar tvrtke (Slika 16).

Središnja tablica ove baze podataka jest „Zaposlenik“, koja sadrži sve ključne atribute vezane uz zaposlenike. Također, implementiran je i **unique constraint** (U) za šifre

zaposlenika te OIB vrijednosti zaposlenika te klijenata, kako bi se osiguralo da se te vrijednosti ne ponavljaju unutar sustava. Svaki zaposlenik može biti kvalificiran za rad u jednom ili više IT područja, te je prema tome dodijeljen odgovarajućem odjelu, što je omogućeno međutablicom „Odj_Zap_Kval“, koja povezuje tablice „Zaposlenik“, „Odjel“ i „Kvalifikacija“. Unutar odjela se formiraju timovi za rad na projektima, gdje je upravljanje resursima i projektima omogućeno preko međutablice „Tim_Projekt“, koja povezuje tablice „Zaposlenik“, „Odjel“ i „Projekt“. Tablica „Faktura“ povezuje klijente, usluge i projekte čiju izradu zatraže, omogućujući praćenje narudžbi usluga, timova koji su radili na projektima, te statusa plaćanja (Slika 16).

Ovaj model normaliziran je do treće normalne forme (3NF), osiguravajući da neključni atributi ovise samo o primarnom ključu tablice, čime se smanjuje redundantnost podataka. Važan aspekt ovog normaliziranog modela jest upotreba JOIN operacija prilikom izvršavanja složenih upita, zbog velike raspodijeljenosti podataka po tablicama. Međutim, veliki broj JOIN operacija može rezultirati padom performansi pri radu s velikim količinama podataka.

5.2.2. ER dijagram normalizirane baze podataka



Slika 16 ER dijagram normaliziranog modela baze podataka za IT Tvrtku

5.3. Denormalizirani model baze podataka

5.3.1. O modelu

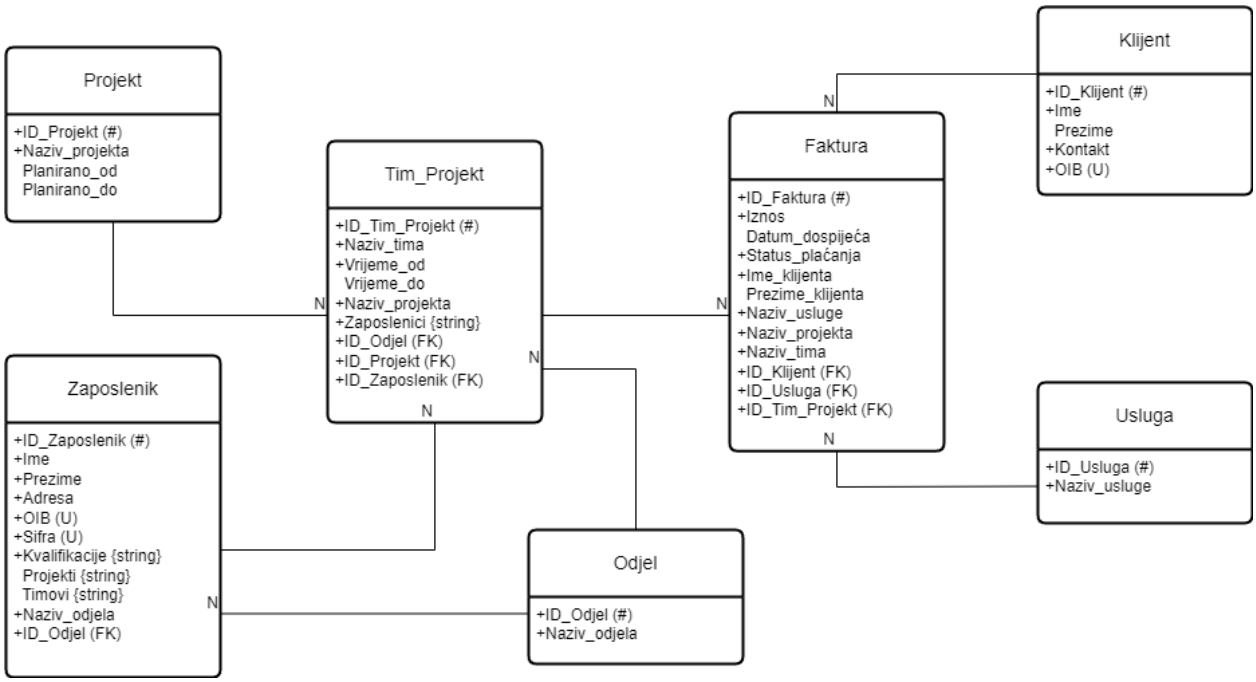
Denormalizirani model baze podataka za IT tvrtku predstavlja optimiziranu verziju normaliziranog modela, s ciljem smanjenja korištenja JOIN operacija te ubrzavanja dohvaćanja podataka.

Korištenjem tehnike dodavanja redundantnih podataka (engl. *Adding Redundant Data*), u tablicu „Tim_Projekt“ dodani su atributi poput „Naziv_Projekta“ i „Zaposlenici“ kako bi se izbjeglo nepotrebno spajanje tablica „Projekt“ i „Zaposlenici“, čime se ubrzava izvršavanje upita. Pod atributom „Zaposlenici“ se sprema popis zaposlenika koji se nalaze u određenom timu, uklanjajući potrebu za korištenjem JOIN operacije za spajanje s tablicom

„Zaposlenik“. Što se tiče tablice „Zaposlenik“, dodani su atributi „Kvalifikacije“, „Projekti“ i „Timovi“, koji spremaju popise kvalifikacija koje zaposlenici imaju, projekata na kojima rade, te timova u kojima se nalaze, te „Naziv_Odjela“, kako bi se izbjeglo korištenje JOIN operacije za spajanje s tablicom „Odjel“. Također je proširena i tablica „Faktura“ dodavanjem atributa „Ime_klijenta“, „Naziv_usluge“, „Naziv_projekta“ i „Naziv_tima“, naravno s istim ciljem kao i kod prethodnih tablica (Slika 17).

Pogledom na ova dva modela, može se uočiti da denormalizirani model izgleda mnogo jednostavnije nakon primjene specifičnih tehnika denormalizacije. Može se uočiti da se broj tablica u bazi podataka smanjio nakon što smo izbacili neke tablice te dodali određene atribute na potrebna mjesta. Denormalizirani model bi imao svrhe kod sustava s velikom količinom podataka, kada je brzina dohvaćanja podataka kritična, izuzevši povećanu kompleksnost održavanja strukture baze podataka te čistoće podataka.

5.3.2. ER dijagram denormalizirane baze podataka



Slika 17 ER dijagram denormaliziranog modela baze podataka za IT Tvrtku

5.4. Metodologija mjerenja performansi

Kako bi se procijenilo koji od modela baze podataka je učinkovitiji, provest će se testiranje performansi, a rezultati će se potom analizirati. Cilj ovog praktičnog dijela je testirati oba modela kroz pet pažljivo odabranih SQL upita te prikupiti podatke o vremenu koje je potrebno za izvršenje tih upita i broju korištenih JOIN operacija. Ova metodologija će omogućiti dublji uvid u način na koji struktura modela utječe na performanse baze podataka u stvarnim uvjetima rada. Očekuje se da će denormalizirani model pokazati bolje performanse izvršavanja zbog smanjenog korištenja JOIN operacija, no stvarne razlike u performansama bit će jasne tek nakon obavljenog testiranja. Nakon završenog testiranja, identificirati će se prednosti i nedostaci obaju modela u kontekstu dohvaćanja podataka.

Koraci metodologije koji će se provesti su:

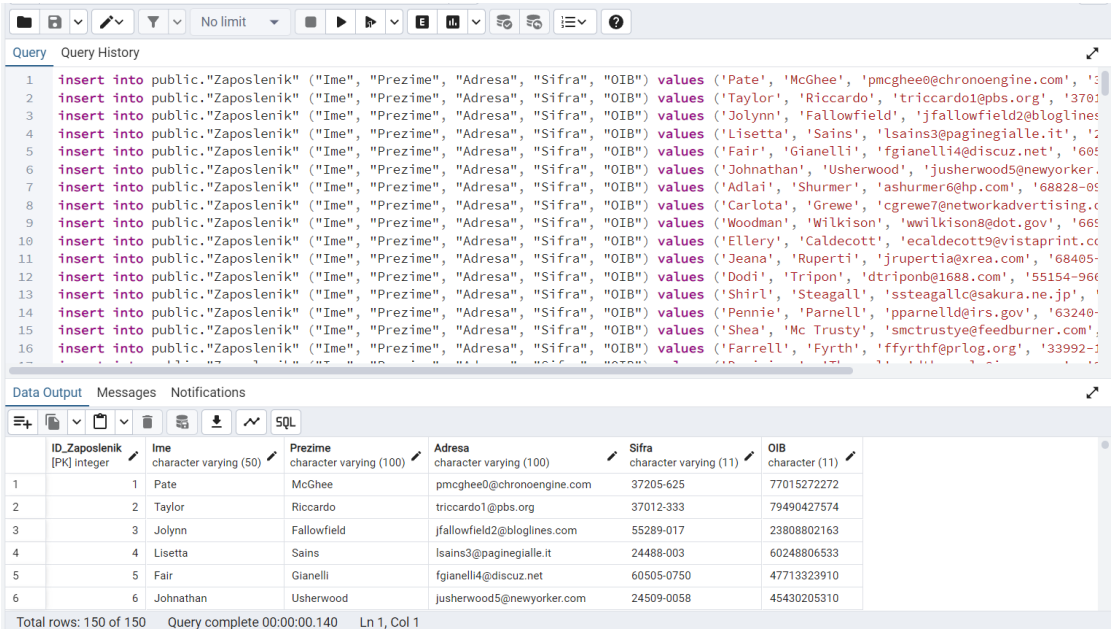
1. **Priprema podataka:** Za potrebe testiranja obje baze podataka će biti popunjene s oko 1900 zapisa sveukupno (1000 za normalizirani, 900 za denormalizirani model) generiranih pomoću alata **Mockaroo**, kako bi simulirali stvarne poslovne scenarije i osigurali konzistentne uvjete za usporedbu performansi.
2. **Odabrani upiti:** Testiranje će se provesti pomoću pet SQL upita, koji su odabrani kako bi se obuhvatili ključni poslovni procesi tvrtke:
   1) Dohvatite sve podatke o zaposlenicima koji rade na odjelu *"Software Development"*.
   2) Dohvatite imena i prezimena zaposlenika iz odjela *"Software Development"*, zajedno s nazivima projekata na kojima rade.
   3) Dohvatite imena i prezimena zaposlenika iz odjela *"Software Development"* zajedno s nazivima projekata i nazivima usluga koje su radili za određene klijente.
   4) Ažurirajte naziv projekta za sve projekte na kojima rade zaposlenici s kvalifikacijom *"Software Engineer"*.
   5) Ažurirajte status plaćanja za sve fakture povezane s projektima čiji su nazivi ažurirani u prethodnom upitu, pod uvjetom da su ti projekti također povezani s privatnim klijentima, a ne tvrtkama (koji imaju uneseno prezime).

3. **Mjerenje performansi:** Za mjerenje vremena izvršavanja koristit će se PostgreSQL-ova naredba `EXPLAIN ANALYZE`, koja omogućuje detaljan pregled performansi. Također će se pratiti koliko je JOIN operacija korišteno u svakom upitu.
4. **Analiza rezultata:** Nakon prikupljanja podataka o vremenu izvršavanja i broju JOIN operacija za svaki upit, rezultati će se usporediti između oba modela putem tablice. Ova usporedba omogućit će uvid u to koliko denormalizacija poboljšava performanse baze podataka.

5.5. Izvršavanje i mjerenje performansi upita

U ovom potpoglavlju bit će prikazano kako su testirani upiti nad normaliziranim i denormaliziranim modelom baze podataka. Metrike koje će se koristiti za analizu performansi, su vrijeme izvršavanja upita te broj korištenih JOIN operacija, kako bi se uvidjele razlike u performansama između oba modela. Za potrebe mjerenja performansi koristit će se PostgreSQL naredba `EXPLAIN ANALYZE`, koja pruža detaljan uvid u način na koji se podaci dohvaćaju i obrađuju.

Prije početka testiranja, obje baze podataka trebalo je napuniti podacima. Taj je postupak izveden korištenjem web alata **Mockaroo**, putem kojeg su generirane SQL skripte za unos podataka u svaku tablicu.



Slika 18 Uneseni podaci za tablicu "Zaposlenik" u PostgreSQL-u, generirani putem web-alata Mockaroo

5.5.1. SELECT upiti

U ovom dijelu bit će prikazani odabrani SELECT upiti korišteni za testiranje performansi oba modela baze podataka. Fokus ovih SELECT upita u normaliziranom modelu jest na dohvaćanju podataka iz više tablica, dok denormalizirani model smanjuje potrebu za korištenjem JOIN operacija zbog pohrane dupliciranih podataka unutar tablica. Svaki upit će se izvršiti tri puta kako bi se preciznije dobilo vrijeme izvršavanja, s obzirom na to da prvo pokretanje može trajati dulje zbog inicijalnog učitavanja podataka. Ova usporedba omogućuje procjenu složenosti i vremena izvršavanja u oba modela, te posebno stavlja naglasak na broj potrebnih JOIN operacija i utjecaj denormalizacije na optimizaciju performansi.

1) **U1-Dohvaćanje svih podataka o zaposlenicima koji rade na odjelu *"Software Development"*.**

```
EXPLAIN ANALYZE
SELECT z.*
FROM public.“Zaposlenik“ z
JOIN public.“Odj_Zap_Kval“ ozk ON z.“ID_Zaposlenik“ = ozk.“ID_Zaposlenik“
JOIN public.“Odjel“ o ON ozk.“ID_Odjel“ = o.“ID_Odjel“
WHERE o.“Naziv_odjela“ = 'Software Development';
```

Kôd 7 SQL SELECT upit za dohvaćanje svih podataka o zaposlenicima koji rade na odjelu "Software Development" u normaliziranoj bazi podataka (PostgreSQL)

```
EXPLAIN ANALYZE
SELECT *
FROM public."Zaposlenik" z
WHERE z."Naziv_odjela" = 'Software Development';
```

Kôd 8 SQL SELECT upit za dohvaćanje svih podataka o zaposlenicima koji rade na odjelu "Software Development" u denormaliziranoj bazi podataka (PostgreSQL)

2) **U2-Dohvaćanje imena i prezimena zaposlenika iz odjela *"Software Development"*, zajedno s nazivima projekata na kojima rade.**

```
EXPLAIN ANALYZE
SELECT z."Ime", z."Prezime", p."Naziv_projekta"
FROM public."Zaposlenik" z
JOIN public."Odj_Zap_Kval" ozk ON z."ID_Zaposlenik" = ozk."ID_Zaposlenik"
JOIN public."Odjel" o ON ozk."ID_Odjel" = o."ID_Odjel"
JOIN public."Tim_Projekt" tp ON ozk."ID_Zaposlenik" = tp."ID_Tim_Projekt"
```

```
JOIN public."Projekt" p ON tp."ID_Projekt" = p."ID_Projekt"
WHERE o."Naziv_odjela" = 'Software Development';
```

Kôd 9 SQL SELECT upit za dohvat imena i prezimena zaposlenika iz odjela "Software Development" s nazivima projekata u normaliziranoj bazi (PostgreSQL)

```
EXPLAIN ANALYZE
SELECT z."Ime", z."Prezime", unnest(z."Projekti") AS "Naziv_projekta"
FROM public."Zaposlenik" z
WHERE z."Naziv_odjela" = 'Software Development';
```

Kôd 10 SQL SELECT upit za dohvat imena i prezimena zaposlenika iz odjela "Software Development" s nazivima projekata u denormaliziranoj bazi (PostgreSQL)

3) **U3-Dohvaćanje imena i prezimena zaposlenika iz odjela *"Software Development"* zajedno s nazivima projekata i nazivima usluga koje su radili za određene klijente.**

```
EXPLAIN ANALYZE
SELECT z."Ime", z."Prezime", p."Naziv_projekta", u."Naziv_usluge",
k."Ime" AS "Ime_klijenta", k."Prezime" AS "Prezime_klijenta"
FROM public."Zaposlenik" z
JOIN public."Odj_Zap_Kval" ozk ON z."ID_Zaposlenik" = ozk."ID_Zaposlenik"
JOIN public."Odjel" o ON o."ID_Odjel" = ozk."ID_Odjel"
JOIN public."Tim_Projekt" tp ON tp."ID_Odjel" = o."ID_Odjel"
JOIN public."Faktura" f ON f."ID_Tim_Projekt" = tp."ID_Tim_Projekt"
JOIN public."Klijent_Usluga" ku ON ku."ID_Klijent_Usluga" =
f."ID_Klijent_Usluga"
JOIN public."Usluga" u ON u."ID_Usluga" = ku."ID_Usluga"
JOIN public."Klijent" k ON k."ID_Klijent" = ku."ID_Klijent"
JOIN public."Projekt" p ON p."ID_Projekt" = tp."ID_Projekt"
WHERE o."Naziv_odjela" = 'Software Development';
```

Kôd 11 SQL SELECT upit za dohvat imena i prezimena zaposlenika iz odjela "Software Development", s projektima i uslugama koje su radili za klijente u normaliziranoj bazi (PostgreSQL)

```
EXPLAIN ANALYZE
SELECT f."Ime_klijenta", f."Prezime_klijenta", f."Naziv_projekta",
f."Naziv_usluge"
FROM public."Faktura" f
JOIN public."Tim_Projekt" tp ON f."ID_Tim_Projekt" = tp."ID_Tim_Projekt"
JOIN public."Odjel" o ON tp."ID_Odjel" = o."ID_Odjel"
```

```
WHERE o."Naziv_odjela" = 'Software Development';
```

Kôd 12 SQL SELECT upit za dohvat imena i prezimena zaposlenika iz odjela "Software Development", s projektima i uslugama koje su radili za klijente u denormaliziranoj bazi (PostgreSQL)

5.5.2. UPDATE upiti

U ovom dijelu bit će prikazani odabrani UPDATE upiti čija je svrha mijenjanje podataka u normaliziranom i denormaliziranom modelu. Testiranjem ovih upita moći će se procijeniti koliko brzo i učinkovito se podaci mogu ažurirati u oba modela, također s posebnim naglaskom na broj korištenih JOIN operacija i njihov utjecaj na performanse. Također, korišteni su **BEGIN** i **ROLLBACK TRANSACTION** kako bi se omogućilo izvršavanje upita po tri puta bez trajnih promjena podataka, što je omogućilo preciznije mjerenje prosječnog vremena izvršavanja.

1) **U4-Ažuriranje naziva projekta za sve projekte na kojima rade zaposlenici s kvalifikacijom *"Software Engineer"*.**

```
EXPLAIN ANALYZE
UPDATE public."Projekt" p
SET "Naziv_projekta" = 'Ažurirani Naziv Projekta'
WHERE p."ID_Projekt" IN (
      SELECT tp."ID_Projekt"
      FROM public."Tim_Projekt" tp
      JOIN public."Odjel" o ON tp."ID_Odjel" = o."ID_Odjel"
      JOIN public."Odj_Zap_Kval" ozk ON o."ID_Odjel" = ozk."ID_Odjel"
      JOIN public."Zaposlenik" z ON ozk."ID_Zaposlenik" =
      z."ID_Zaposlenik"
      JOIN public."Kvalifikacija" k ON ozk."ID_Kvalifikacija" =
      k."ID_Kvalifikacija"
      WHERE k."Naziv_kvalifikacije" = 'Software Engineer'
);
```

Kôd 13 SQL UPDATE upit za ažuriranje naziva projekata za zaposlenike s kvalifikacijom "Software Engineer" u normaliziranoj bazi podataka (PostgreSQL)

```
EXPLAIN ANALYZE
UPDATE public."Zaposlenik" z
SET "Projekti" = array(
      SELECT 'Ažurirani Naziv Projekta'
```

```
      FROM unnest(z."Projekti") p
)
WHERE 'Software Engineer' = ANY(z."Kvalifikacije");
```

Kôd 14 SQL UPDATE upit za ažuriranje naziva projekata za zaposlenike s kvalifikacijom "Software Engineer" u denormaliziranoj bazi podataka (PostgreSQL)

2) **U5-Ažuriranje statusa plaćanja za sve fakture povezane s projektima čiji su nazivi ažurirani u prethodnom upitu, pod uvjetom da su ti projekti također povezani s privatnim klijentima, a ne tvrtkama (koji imaju uneseno prezime).**

```
EXPLAIN ANALYZE
UPDATE public."Faktura" f
SET "Status_placanja" = true
WHERE f."ID_Tim_Projekt" IN (
      SELECT tp."ID_Tim_Projekt"
      FROM public."Tim_Projekt" tp
      JOIN public."Projekt" p ON tp."ID_Projekt" = p."ID_Projekt"
      WHERE p."Naziv_projekta" = 'Ažurirani Naziv Projekta'
)
AND EXISTS (
      SELECT 1
      FROM public."Klijent" k
      JOIN public."Klijent_Usluga" ku ON k."ID_Klijent" = ku."ID_Klijent"
      WHERE ku."ID_Klijent_Usluga" = f."ID_Klijent_Usluga"
      AND k."Prezime" IS NOT NULL
);
```

Kôd 15 SQL UPDATE upit za ažuriranje statusa plaćanja faktura za projekte s ažuriranim nazivima, pod uvjetom da su povezani s privatnim klijentima u normaliziranoj bazi podataka (PostgreSQL)

```
EXPLAIN ANALYZE
UPDATE public."Faktura" f
SET "Status_placanja" = true
WHERE f."Naziv_projekta" = 'Ažurirani Naziv Projekta'
AND f."Prezime_klijenta" IS NOT NULL;
```

Kôd 16 SQL UPDATE upit za ažuriranje statusa plaćanja faktura za projekte s ažuriranim nazivima, pod uvjetom da su povezani s privatnim klijentima u denormaliziranoj bazi podataka (PostgreSQL)

5.6. Usporedba performansi normalizirane i denormalizirane baze podataka

Nakon obavljenog testiranja performansi izvršavanja upita, potrebno je provesti analizu, odnosno usporedbu rezultata testiranja koje smo dobili. Kroz ovo testiranje tri puta je izmjereno vrijeme izvršavanja svakog upita s ciljem dobivanja prosječnih vrijednosti, te je prebrojan i broj korištenih JOIN operacija, što je ključno za uvid u razliku u performansama između ova dva modela.

Što se tiče normaliziranog modela, svih pet upita se temeljilo na korištenju većeg broja JOIN operacija kako bi se dohvatili podaci iz različitih tablica. Ovakav pristup olakšava održavanje te manipulaciju podacima smanjujući redundanciju te osiguravajući konzistentnost podataka. Međutim, glavni problem koji se javlja kod normalizacije su sporije performanse prilikom korištenja velikog broja JOIN operacija. Za primjer možemo uzeti upit U3, koji je zahtijevao čak osam JOIN operacija, te je njegovo prosječno vrijeme izvršavanja bilo 0.978 ms, što je znatno duže usporedbi s upitima koji zahtijevaju manje spajanja (Tablica 1).

S druge strane, proces denormalizacije omogućava djelomičnu, u nekim slučajevima čak i potpunu eliminaciju JOIN operacija. U ovom denormaliziranom modelu, svi podaci koji su potrebni za izvođenje upita su smješteni unutar jedne ili dviju tablica, čime se smanjuje potreba za spajanjem, te se vrijeme izvršavanja upita znatno skraćuje. Primjerice, u upitima U1, U2, U4 i U5 nije bilo potrebe za JOIN operacijama, te je vrijeme izvršavanja bilo znatno kraće nego kod normaliziranog modela (Tablica 1). Čak i upit U3, koji je zahtijevao korištenje dviju JOIN operacija, se pokazao dosta bržim u odnosu na upit kod normaliziranog modela, koji je zahtijevao osam spajanja tablica (Tablica 1).

Upit	Normalizirani model					Denormalizirani model				
	Vrijeme izvršavanja (ms)				JOIN (n)	Vrijeme izvršavanja (ms)				JOIN (n)
	1.	2.	3.	x̄		1.	2.	3.	x̄	
U1	0.125	0.123	0.112	**0.120**	**2**	0.055	0.049	0.045	**0.049**	**0**
U2	0.173	0.164	0.144	**0.160**	**4**	0.051	0.050	0.048	**0.049**	**0**
U3	1.004	0.964	0.968	**0.979**	**8**	0.288	0.270	0.240	**0.266**	**2**
U4	0.572	0.431	0.365	**0.456**	**4**	0.297	0.229	0.194	**0.240**	**0**
U5	0.469	0.428	0.332	**0.410**	**2**	0.060	0.050	0.049	**0.053**	**0**

Tablica 1 Usporedba performansi izvršavanja upita za normalizirani i denormalizirani model baze podataka

Zaključak

Kao što je u samom uvodu navedeno, glavni cilj ovog rada bio je usporediti performanse normalizirane i denormalizirane baze podataka kako bi se donijela odluka o tome koji model je optimalniji za specifične potrebe sustava. U fokusu istraživanja bili su mjerenje vremena izvršavanja upita, broj JOIN operacija korištenih u njima te utjecaj obaju modela na efikasnost upravljanja podacima. Kroz testiranja i analizu rezultata omogućen je bolji uvid u prednosti i nedostatke oba pristupa.

Kroz analizu vremena izvršavanja upita, procesu denormalizacije se sa sigurnošću daje prednost kada se radi o brzini izvršavanja, posebice kod upita koji ne zahtijevaju mnogo spajanja tablica. Testiranja su pokazala da se smanjenjem broja JOIN-ova značajno smanjuje vrijeme potrebno za izvršavanje upita, čime se potvrđuje teorija o negativnom utjecaju složenih spajanja na performanse, odnosno da JOIN operacije igraju značajnu ulogu kod brzine izvršavanja upita.

Iz ovoga se može zaključiti da denormalizacija može biti itekako korisna strategija za optimizaciju performansi u sustavima gdje je brzina dohvaćanja od iznimne važnosti. Međutim, s ovim procesom treba postupati oprezno. Potrebno je pažljivo planiranje svake promjene kako bi se izbjegle nekonzistentnosti, te ograničeno korištenje postupka denormalizacije, jer prekomjerna denormalizacija može dovesti do problema s održavanjem i integritetom podataka. Također može doći do povećanih troškova pohrane zbog velike količine dupliciranih podataka.

U zaključku, optimalan pristup leži u kombinaciji oba procesa – normalizacije za lakšu održivost i dosljednost, te denormalizacije za postizanje boljih performansi. Pravilnim balansiranjem ova dva pristupa sustavu se omogućuje iskorištavanje njihovog punog potencijala, što u konačnici može rezultirati efikasnim upravljanjem podacima i stabilnim, pouzdanim performansama.

Literatura

[1] H. Darwen, An Introduction to Relational Database Theory, Oxford: Bookboon, 2009.

[2] ThinkAutomation, »The History Of Databases,« [Mrežno]. Available: https://www.thinkautomation.com/histories/the-history-of-databases.

[3] LucidChart, »What Is a Database Model,« [Mrežno]. Available: https://www.lucidchart.com/pages/database-diagram/database-models.

[4] t. i. one, »Hierarchy Data Model, Network Data Model,« [Mrežno]. Available: https://theintactone.com/2019/03/26/dbms-u1-topic-5-hierarchy-data-model-network-data-model/.

[5] O'REILLY, »Object-oriented database model,« [Mrežno]. Available: https://www.oreilly.com/library/view/hands-on-big-data/9781788620901/b2b76205-d932-4afa-957f-7013376c553e.xhtml.

[6] MongoDB, »What is NoSQL?,« [Mrežno]. Available: https://www.mongodb.com/resources/basics/databases/nosql-explained.

[7] Oracle, »What is a Relational Database (RDBMS)?,« [Mrežno]. Available: https://www.oracle.com/database/what-is-a-relational-database/.

[8] TechTarget, »What is Structured Query Language (SQL)?,« [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/definition/SQL.

[9] DatabaseTown, »Relational Database Benefits and Limitations (Advantages & Disadvantages),« [Mrežno]. Available: https://databasetown.com/relational-database-benefits-and-limitations/.

[10] MongoDB, »Database Scaling,« [Mrežno]. Available: https://www.mongodb.com/resources/basics/scaling.

[11] PostgreSQL, »About,« [Mrežno]. Available: https://www.postgresql.org/about/.

[12] c. data, »ERD Tool - pgAdmin 4 4.30 documentation,« [Mrežno]. Available: https://access.crunchydata.com/documentation/pgadmin4/4.30/erd_tool.html.

[13] N. A. R. Amato, »Mastering database normalization: A comprehensive exploration of normal forms,« u *ResearchGate*.

[14] Guru99, »DBMS Normalization: 1NF, 2NF, 3NF Database Example,« [Mrežno]. Available: https://www.guru99.com/database-normalization.html.

[15] GeeksforGeeks, »The Problem of Redundancy in Database,« [Mrežno]. Available: https://www.geeksforgeeks.org/the-problem-of-redundancy-in-database/?ref=lbp.

[16] Splunk, »Data Denormalization: The Complete Guide,« [Mrežno]. Available: https://www.splunk.com/en_us/blog/learn/data-denormalization.html.

[17] DEV, »The Trade-offs Between Database Normalization and Denormalization,« [Mrežno]. Available: https://dev.to/er_dward/the-trade-offs-between-database-normalization-and-denormalization-4kdo.

[18] RubyGarage, »When and How You Should Denormalize a Relational Database,« [Mrežno]. Available: https://rubygarage.org/blog/database-denormalization-with-examples#article_title_4.

Popis slika

Popis kôdova

Popis tablica