Modeliranje relacijske baze podataka iz nerelacijskih izvora podataka

Janković, Jakša

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Završni rad

Modeliranje relacijske baze podataka iz nerelacijskih izvora podataka

Jakša Janković

Split, rujan 2024.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

Modeliranje relacijske baze podataka iz nerelacijskih izvora podataka

Jakša Janković

Sažetak

U ovom radu opisana je baza podataka za pohranu informacija o studijima i kolegijima koji se izvode na Prirodoslovno-matematičkom fakultetu u Splitu, kao i JSON u SQL pretvarač koji služi za pohranu podataka iz postojeće NoSQL baze podataka u JSON formatu u novostvorenu relacijsku bazu podataka. Također, rad uključuje skriptu koja uspoređuje vrijeme izvršavanja pet upita na SQL i NoSQL bazi podataka, čime se analiziraju performanse oba modela. Baza podataka izrađena je korištenjem Microsoft SQL Server Management Studio 20, dok je pretvarač razvijen u programskom jeziku Python koristeći uređivač Visual Studio Code.

Ključne riječi: baza podataka, SQL, NoSQL, JSON, Python

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

Rad sadrži: 31 stranicu, 16 slika, 9 literaturnih navoda

Izvornik je na hrvatskom jeziku.

Mentor: **Doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

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

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

Rad prihvaćen: 18.9.2024.

Basic documentation card

Thesis

University of Split

Faculty of Science

Department of computer science

Ruđera Boškovića 33, 21000 Split, Hrvatska

Modeling a Relational Database from Non-relational Data Sources

Jakša Janković

Abstract

This thesis describes a database designed to store information about the study programs and courses offered at the Faculty of Science, University of Split, as well as a JSON to SQL converter that transfers data from the existing NoSQL database in JSON format to a newly created relational database. Additionally, the thesis includes a script that compares the execution time of five queries on both the SQL and NoSQL databases, providing an analysis of the performance of both models. The database was created using Microsoft SQL Server Management Studio 20, while the converter was developed in Python using the Visual Studio Code editor.

Key words: database, SQL, NoSQL, JSON, Python

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

Thesis consists of: 31 pages, 16 pictures, 9 references

Language of origin: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant professor of Faculty of Science, University of Split*

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

Antonela Prnjak, mag. educ. inf., *Instructor of Faculty of Science, University of Split*

Thesis accepted: 18.9.2024.

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni rad s naslovom **Baza podataka za fakultet** izradio samostalno pod voditeljstvom Gorana Zaharije, doc. dr. sc. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student

_____

Jakša Janković

Sadržaj

Uvod

U suvremenom poslovanju, gotovo svaka veća tvrtka, institucija i organizacija pohranjuje značajan dio svojih podataka digitalno, umjesto u tradicionalnim papirnatim spisima. Zbog praktičnosti i nižih početnih troškova, mnoge se organizacije odlučuju za pohranu podataka kod vanjskih pružatelja usluga. Međutim, postoje brojni slučajevi u kojima su takvi vanjski servisi podbacili, što je dovelo do trajnog gubitka velikih količina krucijalnih podataka. Iz tih razloga razvijena je ideja ovog projekta – lokalna pohrana podataka koji su trenutno dostupni samo putem vanjskih servisa, s posebnim naglaskom na potrebe Prirodoslovno-matematičkog fakulteta u Splitu.

Jedan od ključnih ciljeva ovog projekta je kreirati novu SQL bazu podataka i u nju pohraniti podatke koji su trenutačno pohranjeni u jednoj NoSQL datoteci, konkretno u JSON formatu. Podaci pohranjeni u ovoj JSON datoteci, preuzeti s vanjskog servisa, bit će pretvoreni i strukturirani u SQL bazu podataka. Time će se osigurati bolja organizacija, lakše dohvaćanje informacija te veća sigurnost i pouzdanost podataka.

U nastavku rada bit će objašnjeni koncepti baza podataka, njihove vrste te njihove prednosti i nedostaci. Opisat će se i proces izrade baze podataka te alata za pretvorbu podataka, koji će omogućiti pohranu podataka iz JSON datoteke u novokreiranu SQL bazu podataka. Glavni cilj rada je prikazati cjelokupan proces izrade baze podataka i alata za pretvorbu podataka te konačni rezultat – funkcionalnu bazu podataka s pohranjenim podacima o svim studijima i kolegijima koji se pružaju na Prirodoslovno-matematičkom fakultetu u Splitu. Nadalje, u radu će biti uspoređena brzina izvršavanja upita na postojećoj NoSQL bazi podataka u odnosu na novonastalu SQL bazu podataka. Baza podataka izrađena je korištenjem SQL Server Management Studija 20, dok je skripta za pretvorbu razvijena u Pythonu.

1. Baze podataka

„Baza podataka je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim programima. Ubacivanje, promjena, brisanje i čitanje podataka obavlja se posredstvom zajedničkog softvera. Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka, već se referenciraju na logičku strukturu baze“[1]

Baza podataka je strukturirani sustav za pohranu i upravljanje podacima te omogućuje učinkovitu pohranu velikih količina informacija na organiziran način. Baza podataka je temelj velikog broja softverskih sustava, od malih kućnih projekata do složenih korporacijskih rješenja.

Sustavi za upravljanje bazama podataka (DBMS - Database Management Systems) su softverski alati koji se koriste za izradu, upravljanje i manipulaciju bazama podataka. DBMS korisniku pruža sučelje za korištenje baze podataka te znatno olakšava izvršavanje operacija na samoj bazi podataka. Popularni DBMS sustavi uključuju Microsoft SQL Server, MySQL, PostgreSQL, Oracle i SQLite, a u ovom radu korišten je Microsoft SQL Server.

1.1. Modeli

Modeli baza podataka predstavljaju strukture unutar baze podataka. Modeli definiraju na koji način se podaci pohranjuju, na koji se način podacima može pristupiti te kako se podacima može manipulirati. Različiti modeli baza podataka imaju razne prednosti i mane. Odabir modela za stvaranje baze podataka ovisi o brojnim faktorima, ali prvenstveno o podacima koji će se pohranjivati u bazi podataka te o odnosima između tih podatka. Neki od modela baza podataka su relacijski, nerelacijski, hijerarhijski, mrežni i objektno-orijentirani modeli.

1.1.1. Relacijski model

Relacijski model je najčešće korišteni model baza podataka te model korišten za izradu upravo ove baze podataka. Model se temelji na entitetima, to jest na tablicama koje imaju retke i stupce. Svaki redak u tablici predstavlja jedan zapis, a svaki stupac jedan atribut. Tablice su međusobno povezane relacijama, to jest vezama. Te se veze ostvaruju pomoću primarnih i stranih ključeva.

[1] Manager, R(2003). *Baze podataka*, Zagreb https://sssrozaje.me/wp-content/uploads/2020/03/Baze-podataka-skripta.pdf

Primarni ključ jedinstveno identificira svaki zapis u tablici, dok strani ključ služi za povezivanje s drugim tablicama, osiguravajući referencijalni integritet podataka. Na primjer, u tablici "Student", primarni ključ može biti *studentId*, dok se isti taj ključ može pojaviti kao strani ključ u tablici "Upisi", što omogućuje praćenje koje kolegije svaki student pohađa.

Jedna od ključnih prednosti relacijskog modela je osiguranje integriteta podataka. Integritet podataka se održava kroz razne vrste ograničenja kao što su primarni i strani ključevi, ali i ograničenja poput jedinstvenosti, ne-null vrijednosti i ograničenja provjere (CHECK constraints). Ove značajke pomažu u sprječavanju pogrešaka i nesukladnosti u podacima.

Relacijski model također podržava jednostavnu i složenu manipulaciju podacima pomoću SQL-a (Structured Query Language). SQL omogućuje definiranje, manipulaciju i dohvat podataka na strukturiran i lako razumljiv način. Korištenjem SQL-a, korisnici mogu izvoditi razne operacije nad podacima, poput selekcije, projekcije, spajanja tablica (JOIN), grupiranja (GROUP BY), filtriranja (WHERE) i mnogih drugih. Upravo zbog svoje moćne i fleksibilne podrške za složene upite, relacijski model postao je industrijski standard za baze podataka.

1.1.2. Nerelacijski model

Nerelacijski model baza podataka, poznat i kao NoSQL (Not Only SQL), osmišljen je za rad s nestrukturiranim ili polu-strukturiranim podacima koji se teško uklapaju u tradicionalni relacijski model. Ovaj model postao je popularan zbog svoje sposobnosti da rukuje velikim količinama raznolikih podataka uz visoku fleksibilnost i skalabilnost. Za razliku od relacijskog modela, nerelacijske baze podataka ne koriste tablice sa strogo definiranim stupcima i redcima, već koriste različite strukture podataka, kao što su dokumenti, ključ-vrijednost parovi, grafovi i stupci, prilagođene specifičnim potrebama aplikacija.

Dokumentno-orijentirane baze podataka, poput MongoDB-a, pohranjuju podatke u obliku dokumenata u formatu kao što je JSON, omogućujući lako rukovanje složenim i raznolikim podacima. Baze podataka tipa ključ-vrijednost, kao što su Redis ili Amazon DynamoDB, pohranjuju podatke kao jednostavne parove ključeva i vrijednosti, što ih čini izuzetno brzim za pristup i idealnim za aplikacije koje zahtijevaju visoke performanse za pohranu sesija ili predmemorije. Grafičke baze podataka, poput Neo4j, koriste strukture grafova za pohranu podataka i analizu složenih odnosa među podacima, dok baze podataka sa stupcima, kao što je

Apache Cassandra, omogućuju brzi pristup velikim količinama podataka u analitičkim aplikacijama.

1.1.3. Hijerarhijski model

Hijerarhijski model je model baza podataka koji podatke organizira u obliku stabla, gdje svaki čvor predstavlja jedan zapis, dok grane stabla prikazuju veze između zapisa. Model koristi odnos roditelj-dijete, to jest svaki zapis može imati više podređenih zapisa, ali samo jedan nadređeni. Hijerarhijski model je efikasan za pohranu podataka s jasno definiranim hijerarhijama, kao što su primjerice organizacijske strukture.

1.1.4. Mrežni model

Mrežni model sličan je hijerarhijskom, ali za razliku od hijerarhijskog, omogućuje da zapisi mogu imati više nadređenih zapisa. Tako omogućava stvaranje mreže podataka u kojoj bilo koja dva zapisa mogu biti povezana, neovisno o drugim vezama.

1.1.5. Objektno-orijentirani model

Objektno-orijentirani model temelji se na konceptima objektno-orijentiranog programiranja, gdje su podaci organizirani u obliku objekata. Svaki objekt sadrži podatke (atribute) i metode (funkcionalnosti) koji se odnose na te podatke. Ovaj model omogućuje pohranu složenih podataka i odnosa između njih, uključujući nasljeđivanje i polimorfizam.[2]

1.2. Usporedba relacijskih i nerelacijskih baza podataka

Relacijski i nerelacijski modeli baze podataka predstavljaju dva najzastupljenija pristupa u svijetu baza podataka, svaki s vlastitim prednostima i nedostacima. Relacijski model koristi strogu strukturu podataka organiziranu u tablice sa stupcima i redcima, što osigurava konzistentnost i jasnoću u pohrani podataka. Glavna prednost relacijskih baza podataka leži u moćnom SQL jeziku, koji omogućuje korisnicima izvođenje složenih upita i manipulaciju podacima koji se protežu kroz više tablica. To omogućava naprednu analizu i dohvaćanje podataka, što je od velike važnosti za aplikacije koje zahtijevaju preciznost i točnost, kao što su financijski sustavi ili sustavi za upravljanje zalihama.

[2] Manager, R(2003). *Baze podataka*, Zagreb https://sssrozaje.me/wp-content/uploads/2020/03/Baze-podataka-skripta.pdf

Međutim, relacijski modeli imaju i svojih ograničenja. Njihova stroga struktura, iako osigurava dosljednost podataka, otežava horizontalnu skalabilnost, što znači da je dodavanje dodatnih servera u sustav kako bi se rasporedilo opterećenje izazovno. Relacijske baze podataka obično se oslanjaju na vertikalnu skalabilnost, što podrazumijeva povećanje kapaciteta postojećeg servera umjesto dodavanja novih servera. Ovaj pristup može biti skuplji i manje fleksibilan u usporedbi s horizontalnim skaliranjem. Također, zbog stroge sheme, svaka promjena u strukturi podataka zahtijeva modifikaciju same sheme baze, što može biti složen i rizičan zadatak. Iz tog razloga, u relacijskim bazama često se stavlja veliki naglasak na pažljivo planiranje sheme prilikom izrade baze, a neke baze podataka ostaju nepromijenjene desetljećima kako bi se izbjegli rizici povezani s migracijom ili promjenom sheme.

S druge strane, nerelacijski modeli baza podataka ne zahtijevaju strogu shemu, što omogućuje veću fleksibilnost u modeliranju podataka i lakše prilagodbe promjenama. Ovo ih čini idealnim za aplikacije koje rade s nestrukturiranim ili polu-strukturiranim podacima, kao što su društvene mreže, aplikacije za praćenje korisničkog ponašanja ili IoT aplikacije. Nerelacijske baze su također dizajnirane za jednostavnu horizontalnu skalabilnost, što omogućuje učinkovito dodavanje novih servera za rukovanje većim opterećenjem ili povećanje količine podataka.

Međutim, fleksibilnost nerelacijskih baza podataka dolazi uz određene kompromise. Iako omogućuju lakše prilagodbe i skalabilnost, nerelacijski modeli često imaju ograničene mogućnosti upita u usporedbi s relacijskim bazama. Na primjer, pridruživanje tablica, što je standardna operacija u relacijskim bazama, često nije podržano ili je znatno složenije u nerelacijskim bazama. Također, performanse upita mogu varirati i u nekim slučajevima biti sporije nego u relacijskim bazama podataka, posebno kada se radi o složenim upitima koji zahtijevaju pristup podacima iz više izvora.

Još jedan izazov kod nerelacijskih baza je upravljanje podacima bez stroge sheme. Iako to omogućava fleksibilnost, može dovesti do nekonzistentnih i nepredvidivih struktura podataka, što može otežati upravljanje podacima i održavanje baze. Ovo može postati problematično u situacijama gdje je konzistentnost podataka ključna, jer nedostatak stroge sheme može uzrokovati poteškoće u osiguravanju integriteta podataka.

U konačnici, izbor između relacijskih i nerelacijskih baza podataka ovisi o specifičnim potrebama projekta. Relacijske baze odlične su za sustave koji zahtijevaju strogu konzistentnost i podršku za složene transakcije, dok su nerelacijske baze prikladnije za aplikacije koje zahtijevaju fleksibilnost, brzinu i skalabilnost u radu s velikim i raznolikim skupovima podataka.[3]

1.2.1. Primjer relacijske baze podataka

U relacijskoj bazi podataka, podaci se pohranjuju u tablicama koje su povezane putem ključeva. Svaka tablica ima definiranu strukturu sa stupcima (atributima) i redcima (zapisima), gdje stupci predstavljaju karakteristike podataka, a redci individualne zapise. Na primjer, ako pohranjujemo informacije o korisnicima i njihovim narudžbama, mogli bismo imati dvije tablice: jednu za korisnike i jednu za narudžbe.

	KorisnikID	Ime	Prezime	Email
1	1	Ivan	Ivić	ivan@primjer.com
2	2	Ana	Aničić	ana@primjer.com

	NarudzbaID	KorisnikID	Datum	Iznos
1	1	1	2024-09-01	100.00
2	2	2	2024-09-02	150.00

Slika 1. Tablice Korisnik i Narudžba

U ovom primjeru, tablica "Korisnik" sadrži osnovne informacije o korisnicima, dok tablica "Narudžba" sadrži podatke o narudžbama. Ove dvije tablice su povezane putem *KorisnikID*, gdje *KorisnikID* u tablici "Narudžba" djeluje kao strani ključ koji referencira primarni ključ u tablici "Korisnik". Na ovaj način osiguravamo konzistentnost podataka i možemo lako izvesti upite koji dohvaćaju podatke iz više tablica, kao što su svi korisnici s njihovim narudžbama.

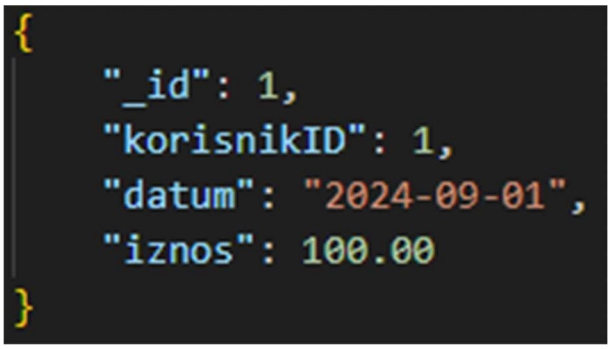
[3] Bačelić, P. (2023). Usporedba relacijskih i nerelacijskih baza podataka na primjeru informatičkog sustava ambulante (Završni rad). Split: Sveučilište u Splitu, Prirodoslovno-matematički fakultet. Preuzeto s https://urn.nsk.hr/urn:nbn:hr:166:038728

1.2.2. Primjer nerelacijske baze podataka

U nerelacijskoj bazi podataka, podaci se mogu pohraniti u fleksibilnijim strukturama, poput dokumenata, često u formatu JSON ili BSON. Ovi dokumenti mogu pohranjivati složene i raznolike podatke unutar jedne strukture, što omogućava veću fleksibilnost pri radu s nestrukturiranim podacima.

Ako uzmemo isti primjer s korisnicima i narudžbama, u dokumentno-orijentiranoj bazi podataka kao što je MongoDB, podaci bi se mogli pohraniti u jedan dokument unutar kolekcije, ili u dvije različite kolekcije, ali bez stroge sheme.

Slika 2. Kolekcija Korisnici



Slika 3. Kolekcija Narudžbe

U prvom primjeru s kolekcijom "Korisnici", sve informacije o korisniku i njegovim narudžbama pohranjene su unutar jednog dokumenta, što omogućava brži pristup i manipulaciju podacima koji su povezani, bez potrebe za pridruživanjem kao u relacijskim bazama. U drugom primjeru, gdje su narudžbe pohranjene odvojeno, podaci se također mogu povezati putem *korisnikID* polja, no pridruživanje je manje formalizirano i često se provodi na razini aplikacijskog koda.

Ovaj pristup omogućuje lakše skaliranje i fleksibilnost u pohrani podataka, no dolazi uz rizik nekonzistentnosti i redundantnosti, budući da nema stroge sheme koja osigurava konzistentnost podataka između različitih dijelova baze.

1.3. Entiteti i atributi

Entitet je osnovni element baze podataka. On predstavlja jedan objekt, jedan pojam koji se pohranjuje u bazi podataka. Entitet se sastoji od jednog ili više atributa. Atribut je svojstvo ili karakteristika entiteta. Ove pojmove najjednostavnije je objasniti primjerom. Entitet nam može biti Kolegij. Kolegij je entitet koji u sebi sadrži informacije o pojedinom kolegiju na sveučilištu. Primjerice, entitet kolegij može sadržavati naziv kolegija i broj ECTS bodova koji kolegij nosi. Ti podaci su atributi entiteta “Kolegij“. Koje informacije će entitet “Kolegij“ pohraniti, to jest koje će atribute imati ovisi o zahtjevima korisnika, odnosno o tome koje podatke o kolegiju korisnik želi imati pohranjene.

1.4. Relacije

Relacije u bazi podataka predstavljaju logičke veze između entiteta, to jest odražavaju stvarne odnose između entiteta. Relacije omogućuju organiziranje i povezivanje podataka iz različitih entiteta pomoću njihovih zajedničkih atributa i time osiguravaju dosljednost podataka.

Tipovi relacija:

1. Jedan-na-jedan(1:1): Relacija jedan-na-jedan označava da je svaki objekt iz jednog entiteta povezan sa točno jednim objektom iz drugog entiteta i obrnuto. Primjer ovoga je relacija između entiteta "Osoba" i "Osobna iskaznica", gdje svaka osoba ima jednu osobnu iskaznicu koja, naravno, pripada isključivo toj osobi. U bazi podatka relacije jedan-na-jedan mogu se izbjeći spajanjem dvije tablice u jednu, ali nekada je bolje ostaviti podatke u dvije tablice, kao kod ovog primjera. Iako se podaci mogu spojiti, kod dohvaćanja podataka o osobi neće uvijek biti potrebni podaci o njenoj osobnoj iskaznici. U ovom slučaju baza podataka bi u većini upita bila sporija spajanjem ovih podataka u jednu tablicu.
2. Jedan-na-više(1:N): Relacije jedan-na-više označava da svaki objekt iz jednog entiteta može biti povezan s više objekata iz drugog entiteta, ali svaki objekt iz drugog entiteta je povezan s maksimalno jednim objektom iz prvog entiteta. Primjer ovoga je relacija između entiteta "Osoba" i "Automobil". Jedna osoba može posjedovati više automobila, ali jedan automobil može imati samo jednog vlasnika.
3. Više-na-više(N:N): Relacija više-na-više označava da svaki objekt iz jednog entiteta može biti povezan s više objekata iz drugog entiteta te obrnuto. Primjer može biti relacija između entiteta "Student" i "Kolegij", gdje jedan student može upisati više kolegija i jedan kolegij može upisati više studenta. U bazi podataka ova relacija razbija se na dvije relacije jedan-na-više te se dodaje dodatna tablica, na primjer "StudentKolegij".

2. Izrada baze podataka

2.1. Opis problema

Za ovaj projekt potrebno je izraditi bazu podataka koja može pohraniti sve relevantne podatke povezane s nastavom na Prirodoslovno-matematičkom fakultetu u Splitu. Trenutno, podaci o različitim studijskim programima i kolegijima fakulteta pohranjeni su u nerelacijskoj bazi podataka u formatu JSON. Ovaj format omogućuje fleksibilnu pohranu nestrukturiranih i polu-strukturiranih podataka, no za potrebe novog sustava, koji zahtijeva precizno modeliranje i analizu podataka, potrebna je relacijska baza podataka.

Relacijska baza podataka treba obuhvatiti informacije o svim studijskim programima i kolegijima koje fakultet trenutno nudi, kao i planira nuditi u budućnosti, uz sve dodatne podatke povezane s tim osnovnim pojmovima. Ključni zahtjevi uključuju podršku za različite razine studija (prijediplomske i diplomske), gdje svaki studij obuhvaća više semestara, a svaki semestar uključuje nekoliko kolegija. Neki kolegiji mogu se izvoditi na više različitih studija, a podaci vezani uz studije i kolegije mogu se mijenjati tijekom akademskih godina, dok su drugi podaci statični. Svaki kolegij može imati jednog ili više nositelja koji se mogu mijenjati iz godine u godinu.

Analizom opisane problematike, jasno je da se pojmovi poput kolegija, semestra i studija mogu definirati kao entiteti, s jasno definiranim relacijama između njih. S obzirom na to da su ovi pojmovi konstantni i njihovi odnosi se nisu mijenjali već desetljećima, relacijski model baze podataka najprikladniji je izbor zbog svoje sposobnosti da precizno modelira i održava ove strukture i odnose.

Kako bi se osigurala uspješna migracija postojećih podataka iz JSON formata u relacijsku bazu, potrebno je razviti pretvarač koji će automatski preuzeti podatke iz nerelacijske baze podataka, mapirati ih na relacijski model i pohraniti ih u novu relacijsku bazu. Ovaj pretvarač mora biti sposoban obraditi sve relevantne podatke, uključujući dinamične informacije koje se mogu mijenjati, te održati integritet podataka tijekom prijenosa.

U konačnici, projekt zahtijeva ne samo dizajn i implementaciju relacijske baze podataka koja zadovoljava specifične potrebe fakulteta, već i razvoj učinkovitog sustava za migraciju podataka iz postojeće nerelacijske baze kako bi se omogućila besprijekorna integracija i konzistentnost podataka.

2.2. Nerelacijska baza podataka

Prije kreiranja relacijske baze podataka te pretvarača potrebno je razumjeti postojeću nerelacijsku bazu podataka, njenu strukturu i podatke koje sadrži. Sama baza podataka se sastoji od jedne JSON datoteke od gotovo 30 000 redaka u kojoj su pohranjene sve informacije o svim kolegijima i studijima na Prirodoslovno-matematičkom fakultetu u Splitu za jednu akademsku godinu.

2.2.1. JSON format

JSON (JavaScript Object Notation) je lagan format za razmjenu podataka koji je jednostavan za čitanje i pisanje ljudima, a također ga je lako parsirati i generirati računalima. Iako je temeljen na sintaksi objekata iz JavaScript jezika, JSON je neovisan o njemu i široko prihvaćen kao standard za razmjenu podataka među različitim sustavima i programskim jezicima. JSON se sastoji od parova ključ-vrijednost, gdje su ključevi stringovi, a vrijednosti mogu biti različiti tipovi podataka poput brojeva, string-ova, objekata ili nizova. Zbog svoje fleksibilnosti i jednostavnosti, JSON se ne koristi samo za pohranu i prijenos konfiguracijskih podataka i rezultata upita iz baza podataka, već se često koristi i u nerelacijskim bazama podataka, kao što su baze podataka bazirane na dokumentima, koje pohranjuju podatke u JSON formatu.

2.2.2. Struktura i podaci

Struktura nerelacijske baze podataka organizirana je u formatu JSON, pri čemu baza prvo sadrži dva glavna para ključ-vrijednost. Prvi par ključ-vrijednost označava prijediplomske studije, dok drugi par predstavlja diplomske studije. Ključ je u oba slučaja nasumični broj iz kojeg se ne može zaključiti što je pohranjeno u vrijednosti, do te informacije može se doći samo analizom svih studija unutar vrijednosti.

Unutar svakog od tih parova ključ-vrijednost, koji označavaju prijediplomske i diplomske studije, pohranjeni su parovi ključ-vrijednost u kojima je ključ akronim studija, a vrijednost svi podaci tog studija.

Vrijednost studija sadrži nekoliko parova ključ-vrijednost. Prvi par ključ-vrijednost unutar vrijednosti sadrži osnovne informacije o svakom studiju poput naziva, razine i akronima te za ključ ima riječ *Info*. Informacija o akronimu već je prethodno zapisana kao ključ studija te je taj podatak dupliciran. Ova baza podataka ima više primjera dupliciranih podataka te je duplikacija podataka jedna od mana nerelacijskih baza podataka. Studij u sebi sadrži i informacije o svim semestrima koje taj studij sadrži.

Semestar za ključ ima broj semestra, a kao vrijednosti ima zapisano različite identifikatore semestra. Unutar semestra također su zapisani svi kolegiji koji se izvode u tom semestru, to jest semestar u sebi ima dva para ključ-vrijednost, jedan s ključem *obvezni* koji sadržava sve obvezne grupe kolegija te jedan s ključem *izborni* koji sadržava sve izborne grupe kolegija.

Unutar, primjerice, obveznih grupa kolegija se nalazi jedna ili više grupa kolegija. Ključ grupe kolegija je njegov jedinstveni identifikator. Svaka grupa kolegija ima unutar sebe zapisano jedan ili više kolegija.

Kolegij kao ključ ima svoj ISVU kod, a unutar vrijednosti sadržava nekoliko identifikatora, te osnovne podatke o kolegiju poput naziva i broja ECTS bodova. Kolegij unutar sebe sadrži i podatke o satnici koji su smješteni unutar para ključ-vrijednosti s ključem *satnica* te vrijednosti koja sadrži ključeve s oznakom vrste nastave i vrijednosti koje imaju pohranjene broj sati nastave.

Unutar kolegija nalaze se i informacije o nositeljima koji se nalaze unutar para ključ-vrijednosti s ključem *nositelji* te vrijednosti koja sadrži informacije o svim nositeljima. Nositelji kao ključ imaju jedinstveni identifikator, a u vrijednosti imaju zapisano osnovne podatke o djelatniku poput imena i titule.

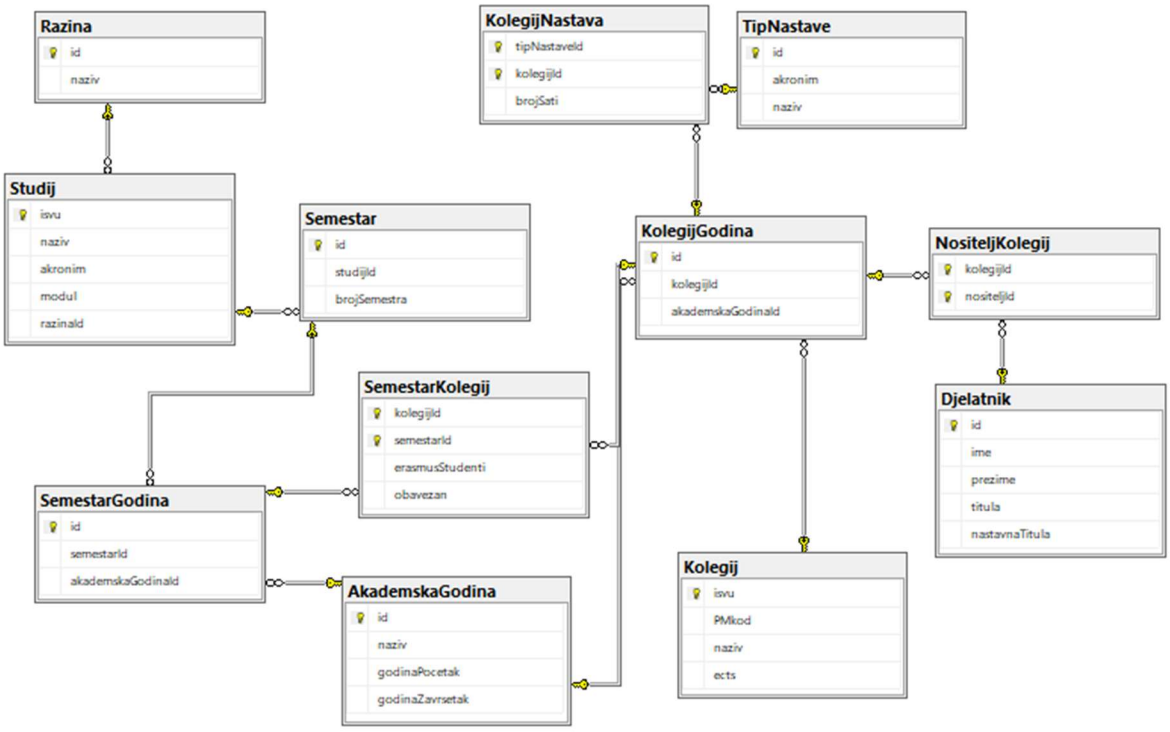
Slika 4. Dio nerelacijske baze podataka

Analizom podataka može se zaključiti da se velik broj podataka duplicira poput identifikatora koji su zapisani i kao ključ unutar para ključ-vrijednosti te naknadno unutar te same vrijednosti. Također, postoje kolegiji koji se izvode na desecima studija te su oni u ovoj bazi podataka zapisani desetak puta sa svim svojim informacijama zbog čega umjesto dvadesetak redaka zauzmu preko dvjesto redaka u bazi podataka.

2.3. Relacijska baza podataka

Baza podataka za ovaj projekt razvijena je korištenjem Microsoft SQL Servera, uz SQL Server Management Studio 20 kao primarni alat za dizajn i upravljanje bazom podataka. Ovaj moćan sustav za upravljanje relacijama baza podataka omogućava učinkovito kreiranje, modifikaciju i održavanje složenih baza podataka s visokom razinom sigurnosti i performansi.

Baza podataka sadrži ukupno 12 tablica, dizajniranih da obuhvate sve ključne aspekte obrazovnog sustava na Prirodoslovno-matematičkom fakultetu u Splitu. Svaka tablica pažljivo je strukturirana kako bi omogućila detaljno praćenje podataka o studijima, semestrima, kolegijima, akademskim godinama i nastavnim aktivnostima. Na Slici 5. nalazi se model baze podataka iz SQL Server Management Studio 20 na kojem su prikazani svi entiteti iz baze podataka te relacije između njih.



Slika 5. Model relacijske baze podataka

2.4. Entiteti

2.4.1. Razina

Entitet “Razina“ predstavlja razinu studija na fakultetu, kao što su prijediplomski ili diplomski studiji. Tablica sadrži dva stupca: jedinstveni identifikator *id*, koji automatski generira vrijednosti pomoću IDENTITY svojstva, i *naziv*, koji pohranjuje naziv razine studija. Ovaj entitet omogućuje jednostavno upravljanje i kategorizaciju studijskih razina unutar sustava baze podataka.

2.4.2. Studij

Entitet “ Studij“ pohranjuje osnovne informacije o svakom studijskom programu, uključujući njegov naziv, akronim, jedinstveni identifikator (ISVU šifra) i pripadajući modul, ukoliko studij ima više modula. Stupac *razinaId* koristi se za povezivanje s entitetom “Razina“, čime se definira na kojoj se razini studij nalazi (npr. prijediplomski). Važno je napomenuti da entitet “Studij“ ne mora nužno odgovarati stvarnom studiju u fizičkom svijetu. Ako studij sadrži više modula, za svaki modul postoji poseban zapis unutar entiteta “Studij“, kao i dodatni zapis za sam studij bez modula. Ovakva struktura omogućava izbjegavanje duplikacije podataka. ISVU šifra koristi se kao jedinstveni identifikator jer je različita, ne samo za svaki studij, nego i za svaki modul unutar tog studija.

2.4.3. Semestar

Entitet “Semestar“ predstavlja jedan semestar unutar određenog studija, primjerice treći semestar prijediplomskog studija Informatike. Tablica sadrži jedinstveni identifikator *id*, koji se automatski generira te *brojSemestra*, koji označava redni broj semestra unutar studija. Stupac *studijId* povezan je s entitetom “Studij“ putem vanjskog ključa, što omogućuje vezu svakog semestra s odgovarajućim studijem. Ova struktura omogućava precizno praćenje semestara unutar različitih studijskih programa.

2.4.4. AkademskaGodina

Entitet “AkademskaGodina“ pohranjuje informacije o akademskim godinama, gdje *godinaPocetak* služi kao primarni ključ i označava godinu početka akademske godine. Ova tablica koristi godinu početka kao jedinstveni identifikator, što omogućuje praćenje i upravljanje različitim akademskim godinama unutar sustava. Godinu početka akademske godine koristimo kao jedinstveni identifikator jer ona mora biti jedinstvena, to jest ne smije se dogoditi da u tablici imamo više zapisa za jednu akademsku godinu.

2.4.5. SemestarGodina

Entitet “SemestarGodina“ služi za upravljanje relacijom više-na-više između entiteta “Semestar“ i “AkademskaGodina“. Ovaj entitet omogućuje povezivanje specifičnih semestara s određenim akademskim godinama. Tablica sadrži jedinstveni identifikator *id*, koji se automatski generira te *semestarId* i *akademskaGodinaId*, koji su vanjski ključevi povezani s entitetima “Semestar“ i “AkademskaGodina“. Ovaj entitet omogućuje praćenje i upravljanje semestrima unutar specifičnih akademskih godina, pružajući fleksibilnost u organizaciji nastave i rasporedu semestara za svaku akademsku godinu.

2.4.6. Kolegij

Entitet “Kolegij“ pohranjuje osnovne informacije o kolegiju, uključujući naziv kolegija, broj ECTS bodova i jedinstveni identifikator. Tablica sadrži *isvu*, koji je primarni ključ i služi kao jedinstveni identifikator kolegija prema ISVU sustavu. Stupac *PMkod* predstavlja specifični identifikator koji koristi Prirodoslovno-matematički fakultet u Splitu za svoje kolegije. Dodatno, stupac *naziv* označava naziv kolegija, dok *ects* prikazuje broj ECTS bodova koje kolegij nosi. Ovaj entitet omogućuje praćenje i upravljanje ključnim informacijama o kolegijima unutar fakultetskog sustava.

2.4.7. KolegijGodina

Entitet “KolegijGodina“ upravlja relacijom više-na-više između entiteta “Kolegij“ i “AkademskaGodina“, prikazujući instancu kolegija za određenu akademsku godinu. Ovaj entitet je sličan entitetu “SemestarGodina“ u svojoj funkcionalnosti. Tablica sadrži jedinstveni identifikator *id*, koji se automatski generira te *kolegijId* i *akademskaGodinaId*, koji su vanjski ključevi povezani s entitetima “Kolegij“ i “AkademskaGodina“. Stupac *erasmusStudenti* pohranjuje broj Erasmus studenata upisanih u određeni kolegij tijekom te akademske godine. Ovaj entitet omogućuje praćenje specifičnih informacija o Erasmus studentima unutar kolegija za svaku akademsku godinu.

2.4.8. SemestarKolegij

Entitet “SemestarGodina“ upravlja relacijom više-na-više između entiteta “Semestar“ i “AkademskaGodina“, omogućujući praćenje informacija o kolegijima u određenom semestru i akademskoj godini. Tablica sadrži jedinstveni identifikator *id*, koji se automatski generira te *semestarId* i *akademskaGodinaId*, koji su vanjski ključevi povezani s entitetima “Semestar“ i “AkademskaGodina“. Stupac *obvezan* koristi se za označavanje je li kolegij obvezan ili izborni u određenom semestru unutar specifične akademske godine. Ovaj entitet koristi složeni ključ koji kombinira *semestarId* i *akademskaGodinaId*, budući da je kombinacija semestra i akademske godine jedinstvena za svaki zapis.

2.4.9. TipNastave

Entitet “TipNastave“ pohranjuje informacije o različitim vrstama nastave koje postoje na fakultetu. Tablica sadrži jedinstveni identifikator *id*, koji se automatski generira te *akronim*, koji predstavlja skraćeni naziv tipa nastave, i *naziv*, koji označava puni naziv tipa nastave. Ovaj entitet omogućuje sustavno praćenje i kategorizaciju različitih vrsta nastave unutar obrazovnog sustava.

2.4.10. KolegijNastava

Entitet “KolegijNastava“ upravlja relacijom više-na-više između entiteta “KolegijGodina“ i “TipNastave“, omogućujući praćenje različitih vrsta nastave za svaki kolegij unutar određene akademske godine. Tablica sadrži dva vanjska ključa: *tipNastaveId*, koji se odnosi na entitet “TipNastave“, i *kolegijId*, koji se odnosi na entitet “KolegijGodina“. Stupac *brojSati* pohranjuje broj sati koji se izvode prema određenom tipu nastave za kolegij u toj akademskoj godini. Ovaj entitet koristi složeni ključ koji kombinira *tipNastaveId* i *kolegijId*, budući da je kombinacija tipa nastave i kolegija u akademskoj godini jedinstvena za svaki zapis.

2.4.11. Djelatnik

Entitet “Djelatnik“ pohranjuje sve relevantne informacije o svakom djelatniku na fakultetu. Tablica sadrži *id*, koji je primarni ključ i služi kao jedinstveni identifikator djelatnika. Ostali stupci uključuju *ime* i *prezime*, koji predstavljaju osobne podatke djelatnika, *titula*, koja označava akademsku ili profesionalnu titulu te *nastavnaTitula*, koja označava nastavnu titulu, ako postoji. Ovaj entitet omogućuje sveobuhvatno praćenje i upravljanje informacijama o djelatnicima unutar fakultetskog sustava.

2.4.12. NositeljKolegij

Entitet “NositeljKolegij“ upravlja relacijom više-na-više između entiteta “KolegijGodina“ i “Djelatnik“, omogućujući praćenje nositelja kolegija u određenoj akademskoj godini. Ovaj entitet omogućuje svakom kolegiju da ima jednog ili više nositelja. Tablica sadrži dva vanjska ključa: *kolegijId*, koji se odnosi na entitet “KolegijGodina“, i *nositeljId*, koji se odnosi na entitet “Djelatnik“. Kombinacija *kolegijId* i *nositeljId* čini složeni ključ, budući da je svaki zapis jedinstven za određeni kolegij u akademskoj godini i njegovog nositelja. Ovaj entitet omogućuje precizno praćenje koji djelatnici su nositelji na određenim kolegijima u specifičnim akademskim godinama.

2.5. Relacije

2.5.1. Relacije jedan na više

- Entitet “Razina“ povezan je relacijom jedan-na-više s entitetom “Studij“. Svaki studij pripada točno jednoj razini studija, dok jedna razina može obuhvaćati više studija. Na primjer, razina "diplomski" može uključivati brojne studije, dok svaki pojedini studij, poput studija “Biologija“, pripada isključivo jednoj razini.
- Entitet “Studij“ povezan je relacijom jedan-na-više s entitetom “Semestar“. Svaki studij može imati više semestara, dok jedan semestar pripada točno jednom studiju. Drugim riječima, svaki semestar specifičan je za određeni studij, dok svaki studij može obuhvaćati nekoliko semestara.

2.5.2. Relacije više na više

- Entiteti “Semestar“ i “AkademskaGodina“ povezani su relacijom više-na-više. Svaka akademska godina može obuhvaćati više semestara, dok svaki semestar može pripadati više akademskim godinama. Zbog ove složene veze, u bazi podataka dodan je entitet “SemestarGodina“, koji omogućuje precizno praćenje i upravljanje instancama semestara unutar specifičnih akademskih godina.
- Entiteti “Kolegij“ i “AkademskaGodina“ povezani su relacijom više-na-više. Svaka akademska godina može obuhvaćati više kolegija, dok se svaki kolegij može izvoditi u više akademskih godina. Zbog ove složene veze, u bazi podataka je dodan entitet “KolegijGodina“, koji omogućuje praćenje i upravljanje informacijama o kolegijima unutar specifičnih akademskih godina.
- Entiteti “SemestarGodina“ i “KolegijGodina“ povezani su relacijom više-na-više. Svaka inačica semestra unutar akademske godine može biti povezana s više inačica kolegija, dok se svaki kolegij može izvoditi u više inačica semestra unutar te iste akademske godine. Drugim riječima, u svakoj akademskoj godini, jedan semestar može uključivati više kolegija, a svaki od tih kolegija može se izvoditi u više semestara. Zbog ove složene veze, u bazi podataka dodan je entitet “SemestarKolegij“, koji omogućuje praćenje i upravljanje rasporedom kolegija unutar različitih semestara u svakoj akademskoj godini. Na primjer,

kolegij "Objektno orijentirano programiranje" može se izvoditi u više semestara unutar iste akademske godine, a svaki od tih semestara može sadržavati više kolegija.

- Entiteti “KolegijGodina“ i “Djelatnik“ povezani su relacijom više-na-više. Svaka inačica kolegija može imati više nositelja, a svaki djelatnik može biti nositelj više kolegija. Da bi se upravljalo ovom složenom relacijom, u bazi podataka je dodan entitet “NositeljKolegij“. Ovaj entitet omogućuje praćenje i upravljanje informacijama o tome koji djelatnici su nositelji određenih kolegija u specifičnim akademskim godinama.
- Entiteti “KolegijGodina“ i “TipNastave“ povezani su relacijom više-na-više. Svaka inačica kolegija može imati više tipova nastave, dok svaki tip nastave može biti povezan s više kolegija. Da bi se upravljalo ovom složenom relacijom, u bazi podataka dodan je entitet “KolegijNastava“. Ovaj entitet omogućuje praćenje i upravljanje informacijama o vrstama nastave koje se primjenjuju na određene kolegije u specifičnim akademskim godinama. Drugim riječima, “KolegijNastava“ omogućuje evidenciju koliko sati nastave različitih tipova postoji za svaki kolegij u određenoj akademskoj godini.

3. Prijenos podataka

Nakon kreiranja baze podataka, sljedeći korak je prijenos podataka iz postojećeg JSON formata u novostvorenu bazu podataka. S obzirom na to da JSON podaci sadrže gotovo 30 tisuća redaka, ručno unošenje i pohranjivanje tih podataka bilo bi izrazito dugotrajan i zahtjevan zadatak, s visokim rizikom od ljudskih pogrešaka.

Kako bi se osigurala točnost i učinkovitost u procesu migracije podataka, razvijen je specijalizirani pretvarač koji automatski konvertira podatke iz JSON formata u SQL bazu podataka. Ovaj pretvarač omogućava automatsko mapiranje i pohranu velikih količina podataka bez potrebe za ručnim unosom, čime se značajno smanjuje mogućnost grešaka i ubrzava cijeli postupak. Također, omogućava dosljednost u strukturiranju podataka i osigurava da svi podaci budu ispravno preneseni i pohranjeni prema definicijama u novoj bazi podataka.

3.1. Alati

Za izradu pretvarača koji automatski konvertira podatke iz JSON formata u SQL bazu podataka, korišten je programski jezik Python zbog svoje fleksibilnosti i snažnih mogućnosti za rad s podacima. Python pruža širok spektar biblioteka i modula koji olakšavaju manipulaciju podacima i komunikaciju s bazama podataka, što ga čini idealnim izborom za ovakav zadatak.

Unutar Python okruženja korištene su sljedeće biblioteke:

- *json*: Ovaj modul omogućava jednostavno učitavanje i obradu podataka pohranjenih u JSON formatu. Pomoću funkcija iz ovog modula, podaci su mogli biti pročitani, dekomponirani i pripremljeni za daljnju obradu.
- *pyodbc*: Ova biblioteka je bila ključna za povezivanje Python skripte s Microsoft SQL Server bazom podataka. *pyodbc* pruža jednostavan način za uspostavljanje veze s bazom podataka, izvršavanje SQL upita i pohranjivanje podataka u bazu. Korištenjem *pyodbc*, pretvarač je mogao izravno komunicirati s bazom podataka i vršiti unos podataka prema potrebnim specifikacijama.[4]

[4] Python SQL Driver – pyodbc https://learn.microsoft.com/en-us/sql/connect/python/pyodbc/python-sql-driver-pyodbc?view=sql-server-ver16

3.2. Redoslijed

Kod umetanja podataka vrlo je bitan redoslijed operacija, to jest kojim redom umetati podatke. Prije dodavanje jednog entiteta nužno je prethodno dodati sve entitete koji imaju strani ključ u entitetu koji se dodaje. Ovo može nekada izazvati niz uvjeta, primjerice prije dodavanja entiteta "SemestarKolegij" potrebno je prethodno dodati entitete "SemestarGodina" i "KolegijGodina". Za dodavanje tih entiteta potrebno je prethodno dodati entitete "Semestar", "Kolegij" i "AkademskaGodina". Prije dodavanja entiteta "Semestar" potrebno je dodati entitet "Studij", a prije dodavanja entiteta "Studij" potrebno je dodati entitet "Razina". Ovaj primjer odličan je pokazatelj zašto je vrlo bitno prije izrade pretvarača odrediti kojim redom popunjavati podatke za entitete.

3.3. Dohvaćanje podataka

Za dohvaćanje podataka prvo je potrebno otvoriti JSON datoteku u kojoj se nalaze podaci. Python ima ugrađenu metodu za to. Nakon toga pomoću modula *json* moguće je deserijalizirati dohvaćene JSON podatke. Zatim je potrebno za svaki entitet koji postoji u bazi podataka dohvatiti podatke i pohraniti ih u neku varijablu. Dohvaćanje podataka za neke entitete je jednostavno, dok je za druge entitete složenije. Primjerice, dohvaćanje podataka za entitete "Razina" i "TipNastave" svodi se na pretraživanje svih različitih vrijednosti za jedan ključ.

Zanimljiv primjer je dohvaćanje podataka za entitet "Kolegij". Nakon dohvaćanja podataka o studijima, potrebno je iterirati po svakom studiju, zatim iterirati po svakom semestru unutar tog studija. Nakon iteriranja po svakom semestru, potrebno je iterirati po svakoj grupi kolegija unutar tih studija (jer postoje različite grupe kolegija oviseći o tome radi li se o obveznom ili izbornom kolegiju) te konačno iterirati po svakom kolegiju i pohraniti podatke o njemu. Međutim, zbog toga što se isti kolegiji izvode na različitim studijima, u dohvaćenim podacima postoje brojni duplikati koje je potrebno filtrirati prije konačne pohrane podataka.

```
for studij in studiji:
    zapis_studij = {
        'isvu': studij['Info']['ISVU'],
        'akronim': studij['Info']['akronim'],
        'razina': studij['Info']['razina'],
        'naziv': studij['Info']['studij'],
        'modul': studij['Info']['modul'],
    }
    studijiTablica.append(zapis_studij)

    for sem_key, sem_value in studij.items():
        if sem_key.isdigit():
            zapis_semestar = {
                'studij': studij['Info']['ISVU'],
                'semester': sem_value['semester'],
            }
            semestriTablica.append(zapis_semestar)
            for group_type in ["obvezni", "izborni"]:
                group_data = sem_value.get(group_type, {})
                if isinstance(group_data, dict):
                    for group_key, group_value in group_data.items():
                        groupID = group_value.get("groupID")
                        for predmet_key, predmet_value in group_value.get("predmeti", {}).items():
                            kolegij = {
                                "studij": studij['Info']['ISVU'],
                                "semester": sem_value['semester'],
                                "groupID": groupID,
                                "subjectID": predmet_value.get("subjectID"),
                                "PMkod": predmet_value.get("PMkod"),
                                "ISVU": predmet_value.get("ISVU"),
                                "naziv": predmet_value.get("naziv"),
                                "ECTS": predmet_value.get("ECTS"),
                                "satnica": predmet_value.get("satnica"),
                                "nositelji": predmet_value.get("nositelji"),
                                "erasmus": predmet_value.get("erasmus"),
                                "obvezan": 1 if group_type == "obvezni" else 0
                            }
                            kolegijiTablica.append(kolegij)
```

Slika 6. Dohvaćanje podataka o kolegijima

3.4. Pohrana podataka

Pohrana podataka u pretvaraču postiže se pomoću modula *pyodbc* koji služi za povezivanje s bazom podataka te omogućava izvršavanje SQL naredbi na bazi podataka korištenjem ugrađenih metoda.

Sama pohrana podataka je jednostavna kada su svi podaci dohvaćeni i formatirani te bi se svela na pozivanje kreirane metode za pohranu podataka, vidljivoj na Slici 7.

```python
def save_to_db(cursor, conn, table_name, columns, data):
    cursor.execute(f"SELECT {columns[0]} FROM {table_name}")
    existing_values = [value[0] for value in cursor.fetchall()]

    data = [row for row in data if row[0] not in existing_values]

    if data:
        columns_str = ', '.join(columns)
        values_str = ', '.join(['?'] * len(columns))
        sql_statement = f"INSERT INTO {table_name} ({columns_str}) VALUES ({values_str})"

        cursor.executemany(sql_statement, data)
        conn.commit()
```

Slika 7. Metoda za pohranu podataka

Određeni entiteti sadrže strane ključeve koji su stvoreni od strane baze podataka stoga je vrijednosti tih stranih ključeva potrebno dohvatiti pomoću upita na bazu podataka. U svrhu toga kreirana je funkcija na Slici 8.

```python
def mapirajStraniKljuc(cursor,id, column_name, foreign_table, foreign_column, data_list):
    for item in data_list:
        cursor.execute(f"SELECT {id} FROM {foreign_table} WHERE {foreign_column} = ?", (item[column_name],))
        foreign_id = cursor.fetchone()[0]
        item[column_name] = foreign_id
```

Slika 8. Metoda za dohvat stranih ključeva

3.5. Pokretanje pretvarača

Pretvarač se pokreće pomoću Pythona. Prilikom pokretanja korisnik mora unijeti godinu početka akademske godine za koje unosi podatke jer taj podatak ne postoji unutar same skripte, a nužan je u bazi podataka. Zatim se od korisnika traži potvrda unosa unošenjem riječi “da“ te se nakon toga skripta izvršava bez ikakve dodatne potrebe za unos od korisnika. U nekoliko sekunda u bazu su pohranjeni svi podaci o studijima, kolegijima i djelatnicima za jednu akademsku godinu.

4. Upiti na bazi

Upiti na bazi su način kako bi se baza koristila za više od same pohrane podataka. Velika prednost relacijskih baza podataka je jednostavnost izrade te brzina izvršavanja upita na bazi podataka.

4.1. Kolegiji po semestrima i studijima

Spajanjem nekoliko tablica moguće je kreirati upit za dohvaćanje svih kolegija u svakom semestru za svaki studij u jednoj akademskoj godini. Također, u rezultatu upita dobijemo i informaciju o broju ECTS bodova koje kolegiji nosi te je li pojedini kolegij u određenom semestru obvezan ili ne. Podaci su sortirani po razini studija, zatim samom studiju, modulu studija, broju semestra te konačno po tome je li kolegij obvezan. Ovim jednim upitom dobiju se sve osnovne informacije o svim kolegijima i studijima koji se izvode u odabranoj akademskoj godini, a upit u SQL se izvršava u manje od jedne sekunde.

```
Select k.naziv,k.ects,s.brojSemestra, sk.obvezan, st.naziv, st.modul, r.naziv from SemestarKolegij sk
 left join SemestarGodina sg on sk.semestarId=sg.id
 left join KolegijGodina kg on kg.id=sk.kolegijId
 left join Semestar s on s.id=sg.semestarId
 left join Kolegij k on k.isvu=kg.kolegijId
 left join Studij st on st.isvu=s.studijId
 left join Razina r on r.id=st.razinaId
 where sg.akademskaGodinaId=2023
 order by r.naziv,st.naziv ,st.modul,s.brojSemestra, sk.obvezan desc
```

Slika 9. Upit za dohvat kolegija po semestrima i studijima

4.2. Popis kolegija za odabrani studij

Popis kolegija za određeni studiji vjerojatno je najtraženija informacija za svakog studenta. Malom preinakom prethodnog upita može se stvoriti upit koji prikazuje sve kolegije za određeni semestar u akademskoj godini, sortirane prvenstveno po semestru te onda po tome je li kolegij obvezan.

```
Select k.naziv,k.ects,s.brojSemestra, sk.obvezan from SemestarKolegij sk
 left join SemestarGodina sg on sk.semestarId=sg.id
 left join KolegijGodina kg on kg.id=sk.kolegijId
 left join Semestar s on s.id=sg.semestarId
 left join Kolegij k on k.isvu=kg.kolegijId
 left join Studij st on st.isvu=s.studijId
 where st.akronim='PD-I' and sg.akademskaGodinaId=2023 order by s.brojSemestra, sk.obvezan desc
```

Slika 10. Upit za dohvat svih kolegija za jedan studij

naziv	ects	brojSemestra	obvezan
Tjelesna i zdravstvena kultura I	0.5	1	1
Praktikum iz internetskih usluga	2.0	1	1
Uvod u racunarstvo	6.0	1	1
Matematika I	7.0	1	1
Programiranje I	6.0	1	1
Osnove elektrotehnike i elektronike	6.0	1	1
Strani jezik u struci I (Engleski)	2.0	1	0
Tjelesna i zdravstvena kultura II	0.5	2	1
Arhitektura racunala	6.0	2	1
Matematika II	7.0	2	1
Programiranje II	6.0	2	1
Praktikum iz arhitekture racunala	4.0	2	1

Slika 11. Rezultat upita za dohvat svih kolegija za jedan studij

4.3. Broj kolegija po profesoru

Spajanjem entiteta Kolegij, Djelatnik i NositeljKolegij može se dobiti broj kolegija u akademskoj godini za koji je svaki djelatnik nositelj. Upit sortira djelatnike po broju kolegija na kojima su nositelji u danoj akademskoj godini.

```
select d.ime,d.prezime, COUNT(d.id) as 'Broj kolegija' from Djelatnik d
left join NositeljKolegij nk on nk.nositeljId=d.id
left join KolegijGodina kg on kg.id=nk.kolegijId
where kg.akademskaGodinaId=2023
group by d.id, d.ime, d.prezime
order by COUNT(d.id) desc
```

Slika 12. Upit za dohvat broja kolegija po profesoru

4.4. Kolegiji na kojima je djelatnik nositelj

Ukoliko nas zanima informacija za nekog određenog djelatnika, to jest na kojim je kolegijima neki djelatnik nositelj, to možemo postići sa sljedećim upitom.

```
select d.ime,d.prezime, k.naziv from Djelatnik d
left join NositeljKolegij nk on nk.nositeljId=d.id
left join KolegijGodina kg on kg.id=nk.kolegijId
left join Kolegij k on k.isvu=kg.kolegijId
where kg.akademskaGodinaId=2023 and d.prezime='Zaharija'
```

Slika 13. Upit za dohvat kolegija jednog djelatnika

4.5. Broj sati po ECTS bodu

Broj sati po ECTS bodu je mjera koliko sati ima određeni kolegiji podijeljeno s brojem ECTS bodova koje taj kolegij nosi. Ovaj upit dohvaća broj sati po ECTS bodu za sve kolegije na jednoj akademskoj godini te ih sortira po broju sati po ECTS bodu. Upit može poslužiti za prikazati značajne razlike pri dodjeli broja sati po ECTS bodu za kolegije, primarno za razlike u kolegijima koji nose jednak broj ECTS bodova.

```
select k.naziv, k.ects, SUM(kn.brojSati) as 'Broj sati', SUM(kn.brojSati)/k.ects as 'Broj sati po ECTS' from KolegijGodina kg
left join Kolegij k on k.isvu=kg.kolegijId
left join KolegijNastava kn on kn.kolegijId=kg.id
where kg.akademskaGodinaId=2023
group by k.naziv, k.ects
order by [Broj sati po ECTS] desc
```

Slika 14. Upit za dohvat broja sati nastave po ECTS bodu

4.6. Usporedba NoSQL i SQL baze podataka

Upiti izvršeni na SQL bazi podataka mogu se izvršiti i na NoSQL bazi podataka. Izvršavanjem upita na obje baze moguće je usporediti kompleksnost stvaranja samih upita te njihovo vrijeme izvršavanja. Takvom usporedbom dobiva se uvid u lakoću korištenja i efikasnost ove dvije baze podataka.

4.6.1. Skripta za usporedbu baza podataka

U svrhe usporedbe dviju baza podataka izrađena je skripta koja izvršava pet prethodno opisanih upita na SQL bazi i na NoSQL bazi te za svaki upit mjeri vrijeme izvršavanja.

Skripta ima osnovnu funkciju koja prima metodu te mjeri i zapisuje njeno vrijeme izvršavanja. Nakon toga slijedi pozivanje pet upita na SQL bazu kojima se mjeri vrijeme te odgovarajućih 5 upita na NoSQL bazu kojima se mjeri vrijeme izvršavanja.

```python
def measureTime(name):
    def decorator(func):
        def wrapper(*args, **kwargs):
            start_time = time.time()
            result = func(*args, **kwargs)
            end_time = time.time()
            execution_time = end_time - start_time
            print("Naziv upita: ", name)
            print("Vrijeme izvršavanja upita: ", execution_time)
            execution_times[name] = execution_time
            return result
        return wrapper
    return decorator
```

Slika 15. Funkcija za mjerenje vremena izvršavanja

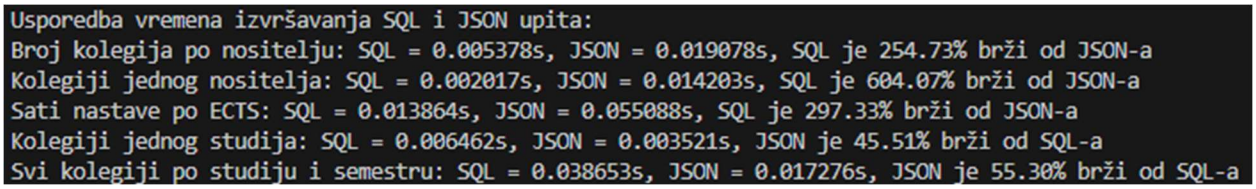
Kao konačni rezultat, skripta uspoređuje vrijeme izvršavanja svakog SQL upita s njegovim odgovarajućim upitom. Skripta ispisuje vremena izvršavanja svakog upita te koji je upit brži i za koliko posto je brži.

4.6.2. Rezultati usporedbe

Cilj ove usporedbe bio je prikazati razlike u stvaranju i izvršavanju upita između postojeće NoSQL baze podataka i novonastale SQL baze podataka.

Usporedbu stvaranja upita zahtjevno je kvantitativno izraziti, no očita je razlika u duljini koda i jednostavnosti izraze upita. Za svih pet upita bilo je znatno jednostavnije i kraće izraziti upit koristeći SQL bazu podataka. SQL omogućuje kraće i jednostavnije upite zbog svoje strukturirane prirode i deklarativnog pristupa. Standardizirane naredbe poput SELECT i JOIN olakšavaju dohvaćanje podataka iz više tablica, dok u nerelacijskim bazama podataka, navigacija kroz hijerarhiju često zahtijeva složeniji i dulji kod.

Vrijeme izvršavanja upita nije nužno bolje kod jedne vrste baza podataka te se to jasno može iščitati mjerenjem vremena izvršavanja prethodno opisanih upita. Upiti koji zahtijevaju samo prolaz kroz JSON podatke bez ikakve manipulacije dohvaćenih podataka biti će brži od SQL upita, no kada su u pitanju složeniji upiti postoji značajna razlika u vremenu izvršavanja u korist SQL upita.



Slika 16. Usporedba vremena izvršavanja SQL i NoSQL upita

Uzimajući sve u obzir, može se zaključiti kako je SQL baza podataka znatno jednostavnija za dohvaćanje podataka, osobito kada se radi o složenim upitima. Zbog jasno definirane strukture i relacija među tablicama, SQL omogućuje jednostavno pisanje upita koji uključuju višestruka spajanja, grupiranja i filtriranja podataka. Ovakvi složeni upiti, koji bi u nerelacijskim bazama zahtijevali opsežniji kod i više koraka, u SQL-u se mogu izraziti precizno i koncizno, koristeći standardne naredbe. Osim toga, SQL baze su optimizirane za rad s velikim količinama podataka i često pružaju bolje performanse prilikom obrade složenih zahtjeva, čineći ih efikasnijim rješenjem u situacijama gdje je potrebna napredna analiza podataka ili upravljanje složenim odnosima između entiteta.

Zaključak

Cilj ovog rada bio je osmisliti i kreirati bazu podataka za pohranu podataka o fakultetu, to jest njegovim studijima te razviti skriptu koja će u novonastalu SQL bazu podataka pohraniti podatke trenutno dostupne u JSON datoteci. Podaci su izvorno pohranjeni u NoSQL formatu, što omogućava veću fleksibilnost, ali se u ovom projektu nastojalo iskoristiti prednosti relacijskih baza za bolje strukturiranje i organizaciju podataka. Kao prvi korak ovog rada napravljena je analiza koje podatke je potrebno pohraniti u bazu podataka, na osnovu čega je izabran relacijski model baze podataka. Nakon izbora modela, kreirana je shema baze podataka, a zatim i sama baza podataka. U sljedećem koraku, razvijena je skripta koja uzima podatke iz JSON datoteke, formatira ih i sprema u kreiranu SQL bazu podataka. Osim toga, izrađena je skripta koja uspoređuje vrijeme izvršavanja pet različitih upita na NoSQL bazi podataka u JSON formatu i na SQL bazi podataka, kako bi se analizirala razlika u performansama između ta dva modela.

Baza podataka u kombinaciji s JSON u SQL pretvaračem omogućava jednostavnu pohranu podataka pri čemu daje korisniku kronološki uvid u razne studije i kolegije koji se izvode na Prirodoslovno-matematičkom fakultetu u Splitu. Baza podataka pruža osnovu i mogućnost za nadogradnju, kao što je kreiranje aplikacije koja bi pregledno prikazivala sve što fakultet nudi te bi mogla biti vrlo korisna studentima, pogotovo ako bi se baza proširila s informacijama o studentima i rasporedom izvođenja nastave. JSON u SQL pretvarač omogućuje jednostavnu i brzu pohranu podataka, kako za trenutnu akademsku godinu, tako i za prošle i nadolazeće akademske godine. Uzimajući sve u obzir, ovaj rad pruža jednostavan i siguran način pohrane podataka za potrebe fakulteta i ujedno može služiti kao podloga za daljnje nadogradnje.

Literatura

1. Šarić, D. (2018). *Baza podataka i mrežna aplikacija za uljare* (Završni rad). Split: Sveučilište u Splitu, Prirodoslovno-matematički fakultet. Preuzeto s https://urn.nsk.hr/urn:nbn:hr:166:243991
2. Tivanovac, M. (2016). *Modeliranje ne relacijskih baza podataka* (Završni rad). Osijek: Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. Preuzeto s https://urn.nsk.hr/urn:nbn:hr:200:558093
3. Bačelić, P. (2023). *Usporedba relacijskih i nerelacijskih baza podataka na primjeru informatičkog sustava ambulante* (Završni rad). Split: Sveučilište u Splitu, Prirodoslovno-matematički fakultet. Preuzeto s https://urn.nsk.hr/urn:nbn:hr:166:038728
4. Hudinec, L. (2023). *Usporedba relacijske i nerelacijske baze podataka* (Završni rad). Čakovec: Međimursko veleučilište u Čakovcu. Preuzeto s https://urn.nsk.hr/urn:nbn:hr:110:420287
5. D. Pereira, P. Oliveira and F. Rodrigues, "Data warehouses in MongoDB vs SQL Server: A comparative analysis of the querie performance," *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, Aveiro, Portugal, 2015, pp. 1-7, doi: 10.1109/CISTI.2015.7170400.
6. How to Convert a JSON String into an SQL Query https://www.sitepoint.com/convert-json-sql-query/
7. Python SQL Driver – pyodbc https://learn.microsoft.com/en-us/sql/connect/python/pyodbc/python-sql-driver-pyodbc?view=sql-server-ver16
8. Read JSON file using Python https://www.geeksforgeeks.org/read-json-file-using-python/
9. Manager, R(2003). *Baze podataka*, Zagreb https://sssrozaje.me/wp-content/uploads/2020/03/Baze-podataka-skripta.pdf