UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Milica Radojičić

AUTOMATIZACIJA PRAVLJENJA RELACIONE BAZE PODATAKA NA OSNOVU TABELARNIH PODATAKA

master rad

Beograd, 2025.

Mentor:

dr Saša Malkov, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Nenad Mitić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Vesna Marinković, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Naslov master rada: Automatizacija pravljenja relacione baze podataka na osnovu tabelarnih podataka

Rezime: U ovom radu razvijen je softver koji za cilj ima da od kolekcije datoteka CSV napravi predlog projekta (dela) baze podataka u obliku SQL/DDL fajla. Predlog projekta baze podataka obuhvata: tabele, kolone i tipove, sugestije primarnih ključeva, sugestije stranih ključeva i sugestije u vezi sa normalnim formama do treće normalne forme.

Ključne reči: relaciona baza podataka, tabelarni podaci, tabele, kolone, tipovi podataka, primarni ključ, strani ključ, normalne forme, CSV, SQL, DDL

Sadržaj

Glava 1

Uvod

1.1 Konvertovanje tabelarnih podataka u baze podataka

Konvertovanje proizvoljnih kolekcija tabelarnih podataka u tabele relacione baze podataka predstavlja osnovni deo procesa integracije podataka i skladištenja podataka. Pretvaranje kolekcije tabelarnih podataka u strukturiranu bazu podataka omogućava lakše i efikasnije pravljenje skladišta podataka, ali i drugih vrsta baza podataka.

Datoteke CSV su popularan i široko korišćen format za čuvanje i razmenu tabelarnih podataka. Ograničavanjem na upotrebu formata CSV ne sužava se mogućnost primene zato što praktično svi drugi tabelarni formati mogu da se konvertuju u format CSV. Automatizovano konvertovanje podataka CSV formata u relacionu bazu podataka omogućilo bi korisnicima da iskoriste velike mogućnosti i fleksibilnost relacionih baza podataka.

U ovom radu je detaljno predstavljen razvijeni softver koji za cilj ima da od kolekcije datoteka CSV napravi predlog projekta (dela) baze podataka u obliku SQL/DDL fajla.

Izvorni kod razvijenog softvera dostupan je na GitHub repozitorijumu.

1.2 Pregled tehnologija i arhitektura softvera

Pregled tehnologija

Za implementaciju softvera odabran je programski jezik Python [2], verzija 3.13, koji pruža bogatu kolekciju biblioteka za obradu i analizu podataka, što je centralno za problematiku automatskog otkrivanja strukture podataka.

Python standardna biblioteka sadrži module za rad sa datotekama CSV (modul `csv` [1]), regularne izraze (modul `re` [3]), i statističku analizu (modul `statistics` [4]), što eliminiše potrebu za spoljašnjim zavisnostima u osnovnoj funkcionalnosti softvera.

Arhitektura softvera

Softver je dizajniran prema modularnoj arhitekturi koja jasno razdvaja različite faze obrade podataka. Ova arhitektura omogućava lakše održavanje, testiranje i proširivanje funkcionalnosti.

Slojevi softvera

Arhitektura se sastoji od četiri glavna sloja:

Sloj za obradu datoteka CSV (`csv_processing`) odgovoran je za učitavanje, parsiranje i osnovnu analizu datoteka CSV. Ovaj sloj sadrži komponente za automatsku detekciju zaglavlja, analizu separatora i osnovnu validaciju strukture datoteka.

Sloj za otkrivanje strukture (`schema_analysis`) predstavlja centralni deo softvera koji implementira algoritme za detekciju tipova podataka, primarnih i stranih ključeva, kao i analizu normalnih formi. Ovaj sloj je podeljen na specijalizovane module za svaki tip analize.

Sloj za generisanje sheme baze podataka (`ddl_generator`) transformiše analiziranu strukturu u konkretan SQL/DDL fajl prilagođen različitim dijalektima baza podataka.

Sloj za konfiguraciju (`config`) omogućava fino podešavanje algoritama kroz konfiguracione parametre i granične vrednosti.

Svaki sloj je organizovan u logičke module koji enkapsuliraju specifičnu funkcionalnost, kao što je prikazano na slici 1.1.

```
csv_to_ddl/
├── main.py (Ulazna tačka aplikacije)
├── argument_parser.py (Parsiranje argumenata komandne linije)
├── csv_to_ddl_converter.py (Glavni orkestrator konverzije)
├── csv_processing/ (Obrada datoteka CSV)
│   ├── csv_analyzer.py (Glavna klasa za CSV analizu)
│   ├── csv_helpers.py (Pomoćne funkcije za pronalaženje i čitanje fajlova CSV)
│   └── header_detection.py (Detekcija zaglavlja fajlove CSV)
├── schema_analysis/ (Analiza strukture)
│   ├── columns_and_types/ (Analiza kolona i tipova)
│   ├── primary_key/ (Detekcija primarnih ključeva)
│   ├── foreign_key/ (Detekcija stranih ključeva)
│   ├── normalization/ (Analiza normalnih formi)
│   └── models/ (Modeli podataka)
├── config/ (Konfiguracija za algoritme detekcije)
│   ├── config_manager.py (Upravljanje konfiguracijama)
│   ├── config_provider.py (Pružalac konfiguracija)
│   ├── default_config.py (Podrazumevana konfiguracija)
│   └── constants/ (Konstante i pragovi)
│       ├── primary_key.py (Konstante za primarne ključeve)
│       ├── foreign_key.py (Konstante za strane ključeve)
│       ├── type_detection.py (Konstante za detekciju tipova)
│       ├── normalization.py (Konstante za normalizaciju)
│       └── csv.py (konstante za CSV)
└── ddl_generator.py (Generisanje SQL/DDL fajla)
```

Slika 1.1: Struktura projekta

Glava 2

Analiziranje podataka

2.1 Koordinacija analize podataka

Analiza podataka predstavlja centralni deo procesa konverzije datoteka CSV u shemu relacione baze podataka. Kao što je prikazano na slici 2.1 ovaj proces obuhvata niz sekvencijalnih koraka: učitavanje i parsiranje datoteka CSV, detekciju zaglavlja, analizu tipova podataka, identifikaciju primarnih i stranih ključeva, kao i proveru normalnih formi.

Svaki od ovih koraka zahteva specifične algoritme i konfiguracione parametre koji određuju preciznost i pouzdanost analize.

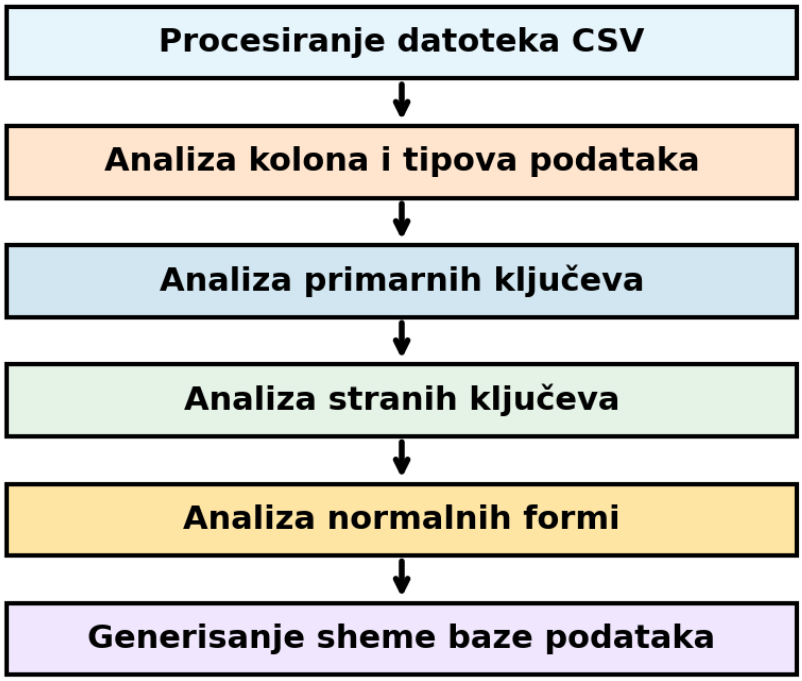
Softver koristi dva ključna projektna obrasca za organizaciju procesa analize podataka: obrazac Fasada za upravljanje složenošću sistema i obrazac Strategija za fleksibilnu konfiguraciju algoritama.

Obrazac Fasada - Glavni interfejs sistema

Klasa `CSVToDDLConverter` implementira obrazac Fasada, pružajući jednostavan interfejs prema složenom sistemu koji koordiniše tri glavne komponente: analizu datoteka CSV, analizu strukture podataka i generisanje sheme baze podataka.

Fasada enkapsulira složenu interakciju između različitih modula sistema:

- **Modul csv_processing** - Rukuje učitavanjem datoteka, detekcijom enkodiranja i parsiranjem sadržaja
- **Modul schema_analysis** - Implementira algoritme za analizu strukture podataka



Slika 2.1: Dijagram toka procesa konverzije datoteka CSV u shemu baze podataka

- **Generator sheme relacione baze podataka** - Transformiše analiziranu strukturu predlog projekta (dela) baze podataka u obliku SQL/DDL fajla

Korisnik poziva `convert()` metodu, dok se koordinacija između komponenti izvršava automatski.

Obrazac Strategija - Fleksibilnost konfiguracije

Klase `ConfigProvider` i `ConfigManager` definišu interfejs za različite načine učitavanja konfiguracije algoritma analize. Ovaj pristup omogućava dodavanje novih izvora konfiguracije bez menjanja postojećeg koda.

Trenutna implementacija koristi `DefaultConfigProvider` koji čita vrednosti iz konstanti definisanih u modulu `constants`. Svaki algoritam analize pristupa konfiguraciji kroz `ConfigManager` koji pruža sledeće tipove konfiguracija:

- **HeaderConfig** - Parametri za detekciju zaglavlja datoteka CSV
- **TypeConfig** - Pragovi za detekciju tipova podataka

- **KeyConfig** - Konfiguracija algoritma za detekciju primarnih i stranih ključeva
- **NormalizationConfig** - Parametri za analizu normalnih formi

Obrazac Strategija omogućava prilagođavanje ponašanja algoritma različitim tipovima podataka ili domenskim zahtevima kroz izmenu konfiguracionih parametara.

Tok procesa analize

Proces analize podataka se izvršava kroz koordinisane korake koje orkestira klasa `CSVToDDLConverter`:

1. **Učitavanje konfiguracije** - ConfigManager čita sve potrebne parametre
2. **Analiza datoteka CSV** - Detekcija zaglavlja, enkodiranja i separatora
3. **Analiza kolona i tipova** - Statistička analiza i detekcija tipova podataka
4. **Detekcija ključeva** - Identifikacija primarnih i stranih ključeva
5. **Analiza normalizacije** - Provera prve, druge i treće normalne forme
6. **Priprema specifikacija** - Kreiranje objekata `TableSpec` za generisanje sheme baze podataka

Svaki korak koristi rezultate prethodnih analiza, stvarajući progresivno detaljniju sliku strukture podataka koja se finalno transformiše u shemu relacione baze podataka.

2.2 Procesiranje datoteka CSV i detekcija zaglavlja

Format CSV predstavlja jedan od najčešće korišćenih načina za razmenu tabelarnih podataka. Međutim, format CSV nije striktno standardizovan, što dovodi do različitih varijanti u pogledu delimitera, enkodiranja i rukovanja specijalnim karakterima. Ova sekcija opisuje implementaciju analizatora CSV koji rukuje različitim varijantama formata CSV i automatski detektuje postojanje zaglavlja u datotekama CSV.

Arhitektura procesora CSV

Analizator CSV je implementiran kroz klasu `CSVAnalyzer` koja koordiniše čitanje datoteka, detekciju formata i pripremu podataka za dalju analizu. Glavne komponente analizatora fajlova CSV su:

- **Pronalaženje datoteka CSV** - Rekurzivno skeniranje direktorijuma i identifikacija datoteka CSV
- **Detekcija enkodiranja** - Automatsko prepoznavanje enkodiranja teksta
- **Detekcija delimitera** - Identifikacija separatora kolona
- **Detekcija zaglavlja** - Identifikacija naziva kolona ili generičko generisanje naziva kolona

Pronalaženje i učitavanje datoteka CSV

Softver podržava fleksibilno učitavanje datoteka CSV iz različitih izvora. Analizator automatski prepoznaje pojedinačne datoteke ili direktorijume i podržava kompresovane datoteke CSV sa ekstenzijom `.csv.gz`.

Proces pronalaska datoteka koristi rekurzivno pretraživanja poddirektorijuma kroz funkciju `find_csv_files`. Za svaku identifikovanu datoteku, sistem proverava dostupnost i validnost pre pokušaja čitanja. Greške prilikom pristupa datotekama se beleže kroz sistem logovanja bez prekidanja procesa analize drugih datoteka.

Detekcija enkodiranja i delimitera

Jedan od glavnih izazova u procesiranju datoteka CSV je raznovrsnost enkodiranja teksta i delimitera. Analizator implementira sekvencijalnu detekciju koja pokušava nekoliko čestih enkodiranja: UTF-8, Latin-1, CP1252 i ISO-8859-1.

Detekcija delimitera koristi klasu `Sniffer` iz standardne biblioteke Python modula `csv`. Sniffer analizira uzorak sadržaja datoteke i pokušava da identifikuje najčešći separator između kolona. Proces detekcije radi sa ograničenim uzorkom datoteke umesto čitanja celog sadržaja, što poboljšava performanse za velike datoteke.

Algoritam detekcije enkodiranja pokušava da otvori datoteku sa svakim enkodiranjem u definisanom redosledu. Prvi uspešan pokušaj dekodiranja se koristi za dalju analizu. Ako nijedna od standardnih opcija ne uspe, datoteka se preskače sa odgovarajućom porukom o grešci.

Detekcija zaglavlja

Detekcija zaglavlja predstavlja kritičan korak u procesiranju datoteka CSV jer direktno utiče na celokupnu dalju analizu strukture. Datoteke CSV mogu imati zaglavlje u prvom redu ili počinjati direktno podacima bez zaglavlja. Pogrešna identifikacija zaglavlja može dovesti do neispravne analize tipova podataka i generisanja neadekvatne sheme relacione baze podataka.

Kompleksnost problema identifikacije zaglavlja

Algoritam za detekciju zaglavlja mora razlikovati između redova koji sadrže nazive kolona i redova koji sadrže stvarne podatke. Ova distinkcija je složena jer nazivi kolona mogu biti numerički (npr. 2023, ID123), dok podaci mogu biti tekstualni (npr. imena, kategorije).

Klasa `HeaderDetection` implementira algoritam koji kombinuje nekoliko heuristika za detekciju zaglavlja kroz metodu `has_header`. Algoritam analizira prvi red datoteke CSV i računa verovatnoću da taj red predstavlja zaglavlje na osnovu različitih kriterijuma, koristeći uzorak podataka iz narednih redova za poređenje.

Multifaktorski algoritam bodovanja

Proces detekcije se odvija kroz sistematičnu analizu svake ćelije prvog reda, pri čemu se koristi kombinacija tri faktora bodovanja:

Pozitivni indikatori zaglavlja (metoda `_calculate_header_indicators_score`): Algoritam analizira svaku ćeliju potencijalnog zaglavlja i dodeljuje pozitivne poene na osnovu četiri kriterijuma. Prvi kriterijum proverava da li ćelija počinje slovom i sadrži samo alfanumeričke karaktere, podvlake ili razmake kroz regex obrazac. Ako ćelija odgovara ovom obrascu, dobija bonus `header_letter_pattern_bonus`. Drugi kriterijum proverava prisustvo podvlaka ili razmaka u nazivu ćelije, što je česta praksa u nazivima kolona, dodeljujući bonus `header_underscore_space_bonus`. Treći kriterijum testira konzistentnost slova - ako je ceo sadržaj ćelije napisan velikim ili malim slovima (kroz metode `isupper()` i `islower()`), dobija bonus `header_case_consistency_bonus`. Četvrti kriterijum dodeljuje bonus `header_length_bonus` ćelijama čija je dužina manja ili jednaka konfigurisanom pragu `header_max_length_threshold`, jer nazivi kolona obično imaju ograničenu dužinu.

Negativni indikatori - penali (metoda `_calculate_anti_header_penalties`): Ćelije koje sadrže numeričke vrednosti dobijaju penale jer je manja verovatnoća da

predstavljaju nazive kolona. Funkcije `is_integer`, `is_decimal` i `is_float` testiraju numeričke formate, dok `is_date` i `is_datetime` proveravaju formate datuma. Ako pouzdanost detekcije bilo kojeg tipa premašuje konfigurisani prag, ćelija dobija odgovarajući penal kroz parametre `header_numeric_penalty` ili `header_date_penalty`.

Poređenje tipova podataka (metoda `_calculate_type_comparison_score`): Kritičan deo algoritma poredi tipove podataka između potencijalnog zaglavlja i uzorka stvarnih podataka iz narednih redova. Za svaku kolonu se poziva funkcija `detect_column_type` zasebno za header ćeliju i za uzorak podataka. Različiti tipovi između zaglavlja i podataka povećavaju verovatnoću da je prvi red zaista zaglavlje kroz `header_type_difference_bonus`. Posebno se favorizuje situacija gde je zaglavlje tekstualnog tipa a podaci su strukturirani (numerički, datumi, email adrese) kroz `header_text_vs_structured_bonus`.

Globalna prilagođavanja i finalna odluka

Nakon bodovanja individualnih ćelija, algoritam primenjuje globalne korekcije konačne ocene:

Penalizacija duplikata (metoda `_calculate_uniqueness_penalty`): Nazivi kolona treba da budu jedinstveni, tako da algoritam računa odnos broja jedinstvenih naziva prema ukupnom broju kolona. Ova metrika se koristi kao multiplikator finalne ocene - duplikati značajno smanjuju verovatnoću da je red zaglavlje.

Bonus za uobičajene obrasce (metoda `_calculate_common_pattern_boost`): Algoritam prepoznaje česte sufikse za nazive kolona: _id, _key, _date, _time, _at, _count, _total, _name i _code. Procenat kolona koje završavaju sa bilo kojim od ovih sufiksa se dodaje kao dodatni bonus finalne ocene.

Prag odlučivanja: Finalna ocena se poredi sa konfigurisanim pragom `header_confidence_threshold` iz `HeaderConfig`. Samo redovi sa ocenom iznad praga se prihvataju kao validna zaglavlja. Ovaj pristup omogućava prilagođavanje osetljivosti algoritma različitim tipovima podataka kroz konfiguracione parametre.

Rukovanje slučajevima bez zaglavlja

Kada algoritam zaključi da datoteka CSV nema zaglavlje, softver automatski generiše nazive kolona kroz metodu `generate_column_names`. Generišu se nazivi u formatu „column_N” gde je N redni broj kolone počevši od 1.

Integracija sa analizom strukture

Procesor CSV priprema podatke u standardizovanom formatu koji omogućava dalju analizu kroz dva glavna izlaza:

Mapiranje zaglavlja tabela: Rečnik `tables_headers` mapira naziv svake tabele na listu naziva kolona. Ovaj rečnik se koristi u svim daljim fazama analize za navigaciju kroz strukturu podataka.

Mapiranje podataka tabela: Rečnik `tables_data` mapira naziv tabele na listu redova podataka, gde je svaki red reprezentovan kao lista stringova. Pošto format CSV čuva sve vrednosti kao stringove, ova struktura očuvava originalan format podataka koji se zatim analizira algoritmima za detekciju tipova radi zaključivanja odgovarajućih SQL tipova.

2.3 Kolone i tipovi

Nakon uspešnog procesiranja datoteka CSV i detekcije zaglavlja, sledeći korak u kreiranju sheme baze podataka predstavlja analizu kolona i automatsku detekciju tipova podataka. Ovaj proces transformiše stringovske vrednosti iz datoteka CSV u odgovarajuće SQL tipove podataka, što direktno utiče na performanse, integritet i funkcionalnost finalne baze podataka.

Arhitektura sistema za detekciju tipova

Sistem za detekciju tipova podataka je implementiran kroz modularan pristup koji omogućava fleksibilnu analizu različitih formata podataka. Glavne komponente sistema su:

- **Detektori tipova** - Specijalizovane funkcije za prepoznavanje specifičnih tipova podataka
- **Analizator tipova kolona** - Koordinator koji upravlja procesom detekcije
- **Kalkulatori veličine tipova** - Određuju optimalne parametre za tipove sa ograničenjima
- **Statistički analizator** - Računa statistike o podacima u kolonama

Algoritam detekcije tipova podataka

Centralni algoritam za detekciju tipova je implementiran u funkciji `detect_column_type` koja koristi višfazni pristup zasnovan na ocenjivanju pouzdanosti detekcije. Algoritam implementira varijaciju obrasca Lanac odgovornosti gde se zahtev za detekciju prosleđuje kroz sekvencijalni niz detektora tipova dok jedan od njih ne pruži zadovoljavajući rezultat. Algoritam analizira sve vrednosti u koloni i pokušava da identifikuje najspecifičniji tip podataka koji odgovara sadržaju.

Procedura detekcije

Proces detekcije se izvršava kroz sistematičan tok operacija:

Filtriranje vrednosti: Algoritam prvo uklanja prazne i null vrednosti iz analize kako bi obezbedio pouzdano prepoznavanje tipova.

Testiranje prema hijerarhiji tipova: Sistem koristi uređenu listu detektora tipova gde se specijalizovani tipovi testiraju pre generalnih. Redosled je kritičan jer sprečava lažno pozitivne rezultate - na primer, vrednost 123 može biti interpretirana kao integer ili kao varchar, ali specifičniji integer tip ima prioritet.

Ocenjivanje pouzdanosti: Svaki detektor vraća odnos pouzdanosti između 0.0 i 1.0 koji predstavlja procenat vrednosti u koloni koje odgovaraju testiranom tipu. Prvi tip čiji odnos pouzdanosti premašuje konfigurisani prag se prihvata kao finalni tip kolone.

Rezervni mehanizam za stringovske tipove: Ako nijedan specijalizovani tip ne dostigne potreban prag pouzdanosti, algoritam analizira dužine stringova i bira između tipova `CHAR`, `VARCHAR` i `TEXT`. Logika izbora je sledeća: ako sve vrednosti u koloni imaju tačno jedan karakter dužine, bira se `CHAR` tip. Ako je maksimalna dužina manja ili jednaka konfigurisanom pragu `max_varchar_length`, bira se `VARCHAR` tip. Za sve ostale slučajeve gde vrednosti prelaze `VARCHAR` ograničenja, koristi se `TEXT` tip.

Hijerarhija testiranja tipova

Sistem implementira preciznu hijerarhiju detektora tipova gde se svaki tip testira sekvencijalno prema opadajućem stepenu specifičnosti.

Boolean tip (funkcija `is_boolean`): Prvi u hijerarhiji zbog visoke specifičnosti. Detektor prepoznaje boolean reprezentacije kroz eksplicitnu listu dozvoljenih vrednosti: 'true', 'false', '1', '0', 'yes', 'no', 'y', 'n', 't', 'f', 'on', 'off', 'enabled', 'disa-

bled'. Prepoznavanje se vrši bez razlikovanja velikih i malih slova. Algoritam broji poklapanja sa dozvoljenim vrednostima i računa odnos sa ukupnim broj vrednosti u koloni.

Integer tip (funkcija `is_integer`): Koristi regex obrazac koji prepoznaje opcioni minus znak praćen jednom ili više cifara. Tokom analize se prate minimalne i maksimalne vrednosti za optimizaciju SQL tipova. Funkcija takođe pokušava konverziju u Python integer kako bi validirala numeričku ispravnost.

Bigint tip (funkcija `is_bigint`): Nadovezuje se na integer detekciju ali dodatno proverava da li brojevi premašuju standardni 32-bitni opseg. Računa odnos vrednosti koje zahtevaju BIGINT skladištenje u odnosu na ukupne integer vrednosti. Ova logika omogućava optimizaciju između INTEGER i BIGINT tipova zavisno od opsega podataka.

Decimal tip (funkcija `is_decimal`): Implementira regex koji striktno traži decimalne brojeve sa fiksnom tačkom. Tokom detekcije se prikupljaju informacije o broju decimalnih mesta što pomaže u određivanju preciznosti i skale za SQL DECIMAL tip.

Float tip (funkcija `is_float`): Koristi Python-ovu ugrađenu `float()` konverziju za prepoznavanje brojeva sa pokretnom tačkom. Dodatno proverava prisustvo decimalne tačke ili naučne notacije ('e' ili 'E') kako bi razlikovao float od integer vrednosti koje mogu biti uspešno konvertovane u float.

UUID tip (funkcija `is_uuid`): Implementira preciznu validaciju UUID formata prema RFC 4122 standardu kroz regex obrazac koji definiše tačan format UUID stringova.

Email tip (funkcija `is_email`): Koristi regex za osnovnu validaciju email adresa. Ovaj regex proverava da li string počinje nizom slova, cifara ili dozvoljenih specijalnih znakova, nakon čega mora da sledi znak @, zatim naziv domena sastavljen od slova, cifara, tačaka ili crtica, i na kraju obavezno završava tačkom posle koje dolazi nastavak domena od najmanje dva slova.

URL tip (funkcija `is_url`): Prepoznaje HTTP i HTTPS adrese kroz regex obrazac. Validacija osigurava da URL počinje sa protokolom i sadrži validne karaktere za web adrese.

Datetime tip (funkcija `is_datetime`): Testira širok spektar datetime formata koristeći Python funkciju `datetime.strptime`. Podržani formati uključuju ISO standard, različite separatore (-, /, .), mikrosekunde i T separator između datuma i vremena. Lista formata pokriva najčešće korišćene reprezentacije datetime vredno-

sti.

Date tip (funkcija `is_date`): Fokusira se isključivo na datumske formate bez vremenske komponente. Koristi sličnu logiku kao datetime ali sa formatima koji ne sadrže vreme. Testiranje date tipa nakon datetime tipa osigurava da se kompletni datetime podaci ne klasifikuju pogrešno kao obični datumi.

Time tip (funkcija `is_time`): Prepoznaje formate vremena kroz regex obrasce koji pokrivaju HH:MM:SS, HH:MM i format sa mikrosekundama.

JSON tip (funkcija `is_json`): Poslednji specijalizovani tip koji testira da li stringovi predstavljaju validne JSON strukture. Proverava da string počinje sa '' ili '[' i završava sa '' ili ']', zatim pokušava parsing kroz standardni Python JSON parser.

Optimizacija uzorkovanja podataka

Algoritam implementira optimizaciju performansi kroz ograničavanje broja vrednosti koje se analiziraju po koloni. Parametar `type_detection_sample_size` definiše maksimalni broj vrednosti koje se testiraju. Uzorkovanje se vrši sa početka kolone.

Konfigurabilni pragovi pouzdanosti

Sistem omogućava podešavanje osetljivosti detekcije kroz parametar `type_detection_confidence_threshold`. Viši pragovi zahtevaju veći procenat poklapanja sa tipom pre prihvatanja, što smanjuje lažno pozitivne rezultate ali može dovesti do klasifikacije podataka kao generičkih stringova.

Ova fleksibilnost omogućava prilagođavanje različitim kvalitetima podataka - podaci sa inkonzistentnostima mogu koristiti niže pragove, dok čisti strukturirani podaci mogu koristiti strože kriterijume.

Analiza veličina tipova podataka

Paralelno sa detekcijom tipova, sistem računa optimalne parametre veličine kroz funkciju `calculate_size_spec`. Različiti tipovi zahtevaju različite pristupe:

String tipovi: Za `VARCHAR` tipove se koristi konfigurisana maksimalna dužina, dok `CHAR` tipovi koriste tačno detektovanu maksimalnu dužinu iz podataka.

Decimalni brojevi: Preciznost i skala se postavljaju na osnovu konfiguracije umesto analize podataka.

Specijalizovani tipovi: UUID tipovi dobijaju standardnu dužinu od 36 karaktera, dok email adrese koriste generičku `VARCHAR` dužinu zbog varijabilnosti email formata.

Statistički metapodaci kolona

Sistem generiše detaljne statistike kroz klasu `ColumnStatistics` koje uključuju:

- **Broj null vrednosti** - Omogućava odlučivanje o nullable ograničenjima
- **Broj različitih vrednosti** - Pomaže u identifikaciji potencijalnih ključeva
- **Odnos jedinstvenosti** - Računa se kao broj različitih vrednosti podeljen ukupnim brojem vrednosti
- **Statistike dužine** - Maksimalna i prosečna dužina stringova za optimizaciju tipova i određivanje ključeva

Ove statistike se koriste u kasnijim fazama analize.

Adaptacija na različite dijalekte baza podataka

Iako detekcija tipova koristi standardizovanu enumeraciju `DataType`, sistem je dizajniran da omogući mapiranje na specifične tipove različitih baza podataka. Funkcija `format_type_with_size` transformiše generičke tipove u sintaksu specifičnu za ciljnu bazu podataka.

Na primer, boolean tip se mapira u `BOOLEAN` za PostgreSQL, `TINYINT(1)` za MySQL, `BIT` za SQL Server, `INTEGER` za SQLite.

Integracija sa ostatkom sistema

Rezultati analize tipova se skladište u strukturi `ColumnSpec` koja kombinuje detektovani tip, parametre veličine, statistike i dodatne metapodatke kao što su nullable ograničenja. Ova standardizovana struktura omogućava ostatku sistema da radi nezavisno od specifičnih implementacijskih detalja detekcije tipova.

Podaci o tipovima kolona predstavljaju temelj za sve dalje analize - detekciju ključeva, normalizaciju i finalno generisanje sheme.

2.4 Primarni ključevi

Uvod u primarne ključeve

Primarni ključ predstavlja osnovni koncept relacionih baza podataka koji omogućava jedinstvenu identifikaciju svakog reda u tabeli. U kontekstu automatskog generisanja sheme baze podataka iz datoteka CSV, prepoznavanje i pravilno definisanje primarnih ključeva predstavlja bitan korak koji direktno utiče na integritet i funkcionalnost rezultujuće baze podataka. [9]

Tok implementacije analizatora primarnih ključeva

Implementacija sistema za automatsko prepoznavanje primarnih ključeva sledi hijerarhijski pristup koji se sastoji od tri glavna koraka izvršavanja. Prvi korak predstavlja pokušaj prepoznavanja prirodnog primarnog ključa kroz analizu karakteristika svake kolone u tabeli. Ako ovaj pokušaj ne uspe, sistem prelazi na drugi korak koji testira kombinacije od dve kolone za kompozitni primarni ključ. U slučaju da ni kombinacije kolona ne zadovoljavaju uslove jedinstvenosti, treći korak kreira veštački primarni ključ koji se automatski dodaje u tabelu.

Glavni analizator primarnih ključeva implementiran je u klasi `PrimaryKeyAnalyzer` koja koristi klasu `ConfigManager` za učitavanje parametara analize. Analiza se pokreće pozivom metode `analyze_primary_key` koja prima specifikaciju tabele, zaglavlja kolona i redove podataka kao ulazne parametre.

Prepoznavanje prirodnih primarnih ključeva

Prvi korak algoritma implementiran je u funkciji `detect_single_key` koja prolazi kroz sve kolone u tabeli i testira da li neka od njih može služiti kao prirodni primarni ključ.

Algoritam filtriranja kandidata

Algoritam počinje filtriranjem kolona na osnovu osnovnih uslova koji moraju biti zadovoljeni za primarni ključ. Prva provera eliminiše sve kolone koje dozvoljavaju null vrednosti, jer primarni ključ mora imati vrednost u svakom redu tabele. Kod implementira ovu proveru kroz svojstvo `nullable` objekta `ColumnSpec`.

Druga provera testira jedinstvenost vrednosti u koloni kroz svojstvo `unique_ratio` iz statistika kolone. Kolone čiji odnos jedinstvenih vrednosti prema ukupnom broju

redova nije manji od praga jedinstvenosti (podešen na vrednost 1.0) se eliminišu iz daljeg razmatranja. Ovaj prag znači da svaka vrednost u koloni mora biti jedinstvena.

Sistem bodovanja kolona

Za kolone koje prođu filtriranje, sistem računa kompozitnu ocenu koji se zasniva na tri faktora implementirana u funkciji `calculate_column_pk_score`.

Prvi faktor analizira naziv kolone i dodeljuje bonuse na osnovu naziva koji sugerišu funkciju primarnog ključa. Kolone sa nazivima 'id', 'pk' ili 'key' kao i kolone čiji nazivi se završavaju sa '_id' ili 'id' dobijaju bonus. Analiza naziva se vrši kroz prevođenje u mala slova i direktno poređenje stringova.

Drugi faktor evaluira tip podataka kolone i favorizuje tipove koji su uobičajeni za primarne ključeve. Kolone sa tipovima `INTEGER`, `BIGINT` ili `UUID` dobijaju bonus.

Treći faktor primenjuje kaznu za tekstualne kolone koje imaju veliku prosečnu dužinu. Ako je kolona tipa `TEXT` i njena prosečna dužina prelazi konfigurisani prag, ta kolona dobija kaznu. Ova logika se zasniva na pretpostavci da dugi tekstualni stringovi verovatno predstavljaju opisne podatke a ne identifikatore.

Izbor najboljeg kandidata

Nakon što se ocene izračunaju za sve kandidate, algoritam sortira kolone prema oceni u opadajućem redosledu i bira kolonu sa najvećom ocenom. Funkcija vraća objekat `PrimaryKeySpec` sa nazivom kolone i tipom ključa označenim kao „natural".

Ako nijedna kolona ne prođe početno filtriranje, funkcija vraća `None` što signalizira da prirodni primarni ključ nije pronađen.

Prepoznavanje kompozitnih primarnih ključeva

Kada analiza ključeva koji imaju jednu kolonu ne uspe, sistem pokušava da pronađe kombinaciju dve kolone koje zajedno mogu služiti kao kompozitni primarni ključ. Ovaj pristup je implementiran u funkciji `detect_composite_primary_key`.

Početna provera mogućnosti kompozitnog ključa

Algoritam prvo proverava da li tabela ima dovoljno kolona za formiranje kompozitnog ključa. Minimalan broj kolona potreban za kompozitni ključ je definisan u konfiguraciji kao `pk_composite_size` sa vrednošću 2.

Sledeća provera filtrira kolone koje ne dozvoljavaju null vrednosti, jer nijedna komponenta kompozitnog primarnog ključa ne sme imati null vrednosti. Ako broj kolona koje ne dozvoljavaju null vrednosti nije manji od potrebnog broja za kompozitni ključ, algoritam prekida izvršavanje.

Generisanje i testiranje kombinacija

Sistem koristi funkciju `combinations` iz modula `itertools` za generisanje svih mogućih kombinacija kolona zadate veličine. Za svaku kombinaciju se računa kompozitna ocena kao zbir ocena pojedinačnih kolona.

Ključni deo algoritma je testiranje jedinstvenosti kombinacije kroz funkciju `_test_composite_uniqueness`. Ova funkcija prima nazive kolona, redove podataka i zaglavlja kao parametre i računa odnos jedinstvenih kombinacija prema ukupnom broju kombinacija.

Testiranje jedinstvenosti kombinacije

Funkcija `_test_composite_uniqueness` prvo mapira nazive kolona na njihove indekse u zaglavlju tabele kroz metodu `index` liste. Za svaki red podataka kreira se n-torka vrednosti iz specificiranih kombinacije kolona.

Sve n-torke se čuvaju u listi, a zatim se koristi `set` struktura podataka za računanje broja jedinstvenih kombinacija. Odnos jedinstvenih kombinacija prema ukupnom broju kombinacija predstavlja meru jedinstvenosti koja mora biti najmanje jednaka pragu jedinstvenosti.

Izbor najbolje kombinacije

Algoritam zadržava kombinaciju sa najvećom ocenom koja zadovoljava uslov jedinstvenosti. Finalna kombinacija se vraća kao objekat `PrimaryKeySpec` sa tipom ključa označenim kao „composite”.

Ako nijedna kombinacija ne zadovoljava uslove jedinstvenosti, funkcija vraća `None`.

Generisanje veštačkog primarnog ključa

U slučaju da ni prirodni ni kompozitni primarni ključ nisu pronađeni, sistem automatski kreira veštački primarni ključ implementiran u funkciji `generate_surrogate_primary_key`.

Kreiranje naziva za veštački ključ

Funkcija prvo konstruiše predlog naziva kolone kombinovanjem naziva tabele sa sufiksom „_id". Da bi izbegla konflikte sa postojećim nazivima kolona, algoritam proverava da li predloženi naziv već postoji u tabeli kroz poređenje sa skupom postojećih naziva prevedenih u mala slova.

Ako dođe do konflikta naziva, sistem dodaje numerički sufiks „_1", „_2", itd. sve dok ne pronađe jedinstveni naziv. Ovaj pristup garantuje da veštački ključ neće imati isti naziv kao postojeća kolona.

Definisanje karakteristika veštačkog ključa

Veštački primarni ključ se definiše kao kolona tipa `INTEGER` koja ne dozvoljava null vrednosti i ima svojstvo automatskog uvećavanja (`is_auto_increment = True`). Statistike se postavljaju tako da odražavaju savršenu jedinstvenost - broj jedinstvenih vrednosti je jednak broju redova u tabeli, a odnos jedinstvenosti je 1.0.

Funkcija vraća torku objekata: `PrimaryKeySpec` sa tipom ključa „surrogate" i `ColumnSpec` koji definiše novu kolonu koja će biti dodata u tabelu.

Integracija sa glavnim analizatorom tabela

Analizator primarnih ključeva je integrisan u širi sistem analize tabela kroz klasu `PrimaryKeyAnalyzer`. Glavna metoda `analyze_primary_key` koordiniše izvršavanje tri algoritma u određenom redosledu i vraća rezultat analize zajedno sa eventualnom novom kolonom u slučaju veštačkog ključa.

Sistem koristi mehanizam logovanja za praćenje procesa analize, pri čemu se beležuju informacije o pronađenim kandidatima, njihovim ocenama i konačnom izboru primarnog ključa za svaku tabelu.

2.5 Strani ključevi

Uvod u strane ključeve

Strani ključ predstavlja fundamentalni mehanizam za uspostavljanje i održavanje referencijalnog integriteta u relacionim bazama podataka. U kontekstu automatskog generisanja strukture baze podataka iz datoteka CSV, prepoznavanje stranih ključeva omogućava sistemu da identifikuje veze između tabela i automatski kreira

odgovarajuće ograničenja koja osiguravaju konzistentnost podataka kroz celu bazu. [7]

Tok implementacije analizatora stranih ključeva

Implementacija sistema za automatsko prepoznavanje stranih ključeva sledi složen proces koji kombinuje analizu sadržaja podataka sa heurističkim pravilima za prepoznavanje obrazaca imenovanja. Glavni tok se sastoji od četiri ključne faze: kreiranje mape referentnih ključeva, prepoznavanje stranih ključeva koji imaju jednu kolonu, prepoznavanje kompozitnih stranih ključeva i rešavanje konflikata između potencijalnih veza.

Glavni analizator stranih ključeva implementiran je u klasi `ForeignKeyAnalyzer` koja koristi klasu `ConfigManager` za učitavanje parametara analize. Analiza se pokreće pozivom metode `analyze_foreign_keys` koja prima mape specifikacija tabela i podataka kao ulazne parametre i vraća proširene specifikacije sa identifikovanim stranim ključevima.

Kreiranje mape referentnih ključeva

Pre početka analize stranih ključeva, sistem kreira centralnu mapu svih potencijalnih referentnih ključeva kroz funkciju `build_reference_keys_map`. Ova mapa služi kao osnova za sve dalje analize i omogućava efikasno testiranje preklapanja vrednosti.

Struktura mape referentnih ključeva

Za svaku tabelu u sistemu, mapa sadrži četiri ključna elementa. Prvi element `single_keys` predstavlja mapiranje naziva kolona na skupove jedinstvenih vrednosti za kolone koje mogu služiti kao referentni ključevi. Drugi element `composite_keys` sadrži mapiranje parova naziva kolona na skupove kompozitnih vrednosti za slučajeve kada više kolona zajedno formira referencu. Treći element `header` čuva listu naziva svih kolona u tabeli što omogućava brzo mapiranje između naziva kolona i njihovih pozicija u podacima. Četvrti element `primary_key_columns` sadrži skup naziva kolona koje učestvuju u primarnom ključu tabele, što je važno za dodavanje bonusa pri ocenjivanju potencijalnih veza.

Algoritam popunjavanja mape

Algoritam prvo identifikuje primarne ključeve svake tabele i dodaje ih u mapu kao prioritetne referentne kandidate. Za primarne ključeve koji se sastoje od jedne kolone, sistem izvlači sve jedinstvene vrednosti kroz funkciju `get_single_column_values_from_data` i stavlja ih u mapu pod `single_keys` ključem. Za kompozitne primarne ključeve, koristi se funkcija `get_composite_values_from_data` koja kreira parove kombinovanih vrednosti koje se stavljaju u mapu pod `composite_keys` ključem.

Dodatno, algoritam analizira sve ostale kolone u tabeli i uključuje u mapu pod `single_keys` ključem one čiji jedinstveni odnos vrednosti zadovoljava prag jedinstvenosti definisan u konfiguraciji. Ova logika omogućava prepoznavanje alternativnih ključeva koji mogu služiti kao validne reference čak i kada nisu eksplicitno označeni kao primarni ključevi.

Prepoznavanje stranih ključeva

Prvi korak u prepoznavanju stranih ključeva implementiran je u funkciji `detect_single_column_foreign_keys` koja analizira svaku kolonu u tabeli kao potencijalni strani ključ.

Algoritam analize kolona

Za svaku kolonu u tabeli, algoritam prvo izvlači sve jedinstvene vrednosti iz podataka. Zatim poredi ove vrednosti sa svakom potencijalnom referentnom kolonom u drugim tabelama kroz računanje ocene preklapanja i primenu heurističkih bonusa.

Funkcija `calculate_match_score` implementira logiku ocenjivanja koja kombinuje kvantitativnu analizu preklapanja vrednosti sa kvalitativnom analizom obrazaca imenovanja. Ova funkcija vraća kompozitnu ocenu koji predstavlja verovatnoću da analizirana kolona predstavlja strani ključ koji referencira određenu kolonu u drugoj tabeli.

Validacija odnosa stranih ključeva

Osnovna validacija se vrši kroz funkciju `_is_valid_fk_relationship` koja testira da li je preklapanje vrednosti dovoljno za uspostavljanje valjane veze. Funkcija računa odnos preklapajućih vrednosti prema ukupnom broju jedinstvenih vrednosti u koloni koja se testira.

Ako ovaj odnos prelazi konfigurisani prag preklapanja `FK_OVERLAP_THRESHOLD`, algoritam dodaje bonus za preklapanje i dodatno ocenjuje odnos veličina skupova vrednosti. Kolone čije referentne tabele imaju značajno više jedinstvenih vrednosti dobijaju bonus jer to ukazuje na tipičnu vezu jedan-prema-više koja je karakteristična za strane ključeve.

Heuristike imenovanja kolona

Funkcija `_get_naming_score` implementira sistem za prepoznavanje obrazaca imenovanja koji sugerišu veze stranih ključeva. Sistem razlikuje nekoliko tipova obrazaca sa različitim bonusima.

Najveći bonus `FK_EXACT_MATCH_BONUS` dobijaju kolone koje imaju identične nazive sa referentnim kolonama, što često ukazuje na direktnu vezu. Manji bonus `FK_TABLE_PREFIX_BONUS` se dodeljuje kolonama čiji nazivi slede konvenciju da naziv strane tabele služi kao prefiks naziva kolone, kao što su `user_id` koji referiše kolonu `id` u tabeli `user`.

Dodatni bonusi se dodeljuju za kolone čiji nazivi sadrže naziv referentne tabele (`FK_TABLE_REFERENCE_BONUS`) ili pojedinačne reči iz naziva tabele (`FK_WORD_MATCH_BONUS`). Ova hijerarhija bonusa omogućava sistemu da prepozna različite konvencije imenovanja koje se koriste u praksi.

Bonus za referenciranje primarnih ključeva

Sistem dodatno favorizuje veze koje referišu primarne ključeve drugih tabela kroz funkciju `_get_primary_key_bonus`. Kolone koje referišu primarne ključeve dobijaju dodatni bonus `FK_PRIMARY_KEY_TARGET_BONUS` jer su takve veze najčešće i najvaljanije u relacionim bazama podataka.

Prepoznavanje kompozitnih stranih ključeva

Druga faza analize implementirana je u funkciji `detect_composite_foreign_keys` koja se fokusira na prepoznavanje složenih veza gde više kolona zajedno formira strani ključ koji referiše kompozitni primarni ključ u drugoj tabeli.

Identifikacija kompozitnih obrazaca

Algoritam počinje identifikacijom svih kompozitnih primarnih ključeva u referentnim tabelama kroz funkciju `_get_composite_pk_patterns`. Ova funkcija pro-

lazi kroz mapu referentnih ključeva i izvlači sve kombinacije kolona koje formiraju kompozitne ključeve.

Za svaki identifikovani kompozitni obrazac, funkcija `_find_matching_columns` pokušava da pronađe odgovarajuće kolone u analiziranoj tabeli koje imaju identične nazive sa komponentama kompozitnog ključa. Ovaj pristup se oslanja na konvenciju da strani ključevi imaju iste nazive kao referentni ključevi.

Evaluacija kompozitnih preklapanja

Kada se pronađe potencijalno poklapanje naziva kolona, funkcija `_evaluate_composite_match` testira preklapanje kombinovanih vrednosti. Za razliku od analize ključeva koji se sastoje od jedne kolone, ovde se koriste parovi iz više kolona simultano.

Sistem kreira skupove parova iz kombinovanih vrednosti kolona i poredi ih sa odgovarajućim skupovima iz referentnih tabela. Odnos preklapajućih parova mora zadovoljiti isti prag kao za ključeve koji se sastoje od jedne kolone da bi veza bila smatrana validnom.

Rešavanje konflikata između potencijalnih veza

Poslednja faza analize implementira mehanizam za rešavanje situacija gde više potencijalnih veza konkuriše za istu relaciju između tabela kroz funkciju `_claim_relationship`.

Algoritam upravljanja vezama

Sistem koristi mapu `claimed_relationships` gde svaki par tabela može imati maksimalno jednu aktivnu vezu. Ključ mape predstavlja sortiran par naziva tabela što osigurava da se veza `(A, B)` tretira identično sa vezom `(B, A)`.

Kada sistem identifikuje novu potencijalnu vezu, prvo proverava da li već postoji registrovana veza između istih tabela. Ako ne postoji, nova veza se registruje. Ako postoji, sistem poredi ocene ove dve veze i zadržava onu sa većom ocenom pouzdanosti.

Finalizacija stranih ključeva

Na kraju procesa analize, svi pobednički strani ključevi se dodaju u specifikacije odgovarajućih tabela. Ova finalizacija se vrši kroz iteraciju preko mape

`claimed_relationships` gde se svaki pobednički strani ključ dodaje u listu `foreign_keys` odgovarajuće tabele.

Rukovanje specijalnim slučajevima

Sistem implementira posebnu logiku za rukovanje generičkim nazivima kolona kroz funkciju `_is_generic_column_name` koja prepoznaje automatski generisane nazive kolona kao što su `column_1`, `column_2`, itd.

Za ovakve kolone primenjuju se ublaženi kriterijumi jer njihovi nazivi ne nose semantičku informaciju o potencijalnim vezama. Sistem se tada oslanja isključivo na preklapanje vrednosti i bonuse za referenciranje primarnih ključeva.

Integracija sa glavnim analizatorom tabela

Analizator stranih ključeva je integrisan u širi sistem analize tabela kao poslednji korak nakon određivanja primarnih ključeva. Prepoznavanje stranih ključeva zavisi od prethodno identifikovanih primarnih ključeva koji služe kao referentne tačke.

Sistem koristi mehanizam logavanja za detaljno praćenje procesa analize, uključujući identifikaciju potencijalnih kandidata, računanje ocena pouzdanosti i konačne odluke o uspostavljanju veza između tabela.

2.6 Normalne forme

Uvod u normalne forme

Normalne forme predstavljaju fundamentalne principe dizajna relacionih baza podataka koji osiguravaju smanjenje redundantnosti i eliminisanje anomalija pri unosu, ažuriranju i brisanju podataka. U kontekstu automatskog generisanja sheme baze podataka iz datoteka CSV, analiza normalnih formi omogućava sistemu da identifikuje potencijalne probleme u strukturi podataka i predloži poboljšanja kroz dekompoziciju tabela. [8] [11]

Tok implementacije analizatora normalnih formi

Implementacija sistema za analizu normalnih formi sledi hijerarhijski pristup koji poštuje prirodni redosled provere normalnih formi. Sistem prvo analizira prvu

normalnu formu (1NF), zatim drugu normalnu formu (2NF) i konačno treću normalnu formu (3NF). Ovaj redosled je kritičan jer svaka viša normalna forma zahteva da tabela već zadovoljava sve niže forme.

Glavni analizator normalnih formi implementiran je u klasi `NormalizationAnalyzer` koja koristi klasu `ConfigManager` za učitavanje parametara analize. Analiza se pokreće pozivom metode `analyze_normalization` koja prima naziv tabele, zaglavlja kolona, podatke i specifikaciju tabele kao ulazne parametre.

Obrazac Šablonski metod za normalne forme

Sistem koristi obrazac Šablonski metod za implementaciju različitih normalnih formi kroz apstraktnu osnovnu klasu `NormalForm`. Ovaj pristup omogućava konzistentan interfejs za sve analizatore normalnih formi i olakšava dodavanje novih normalnih formi u budućnosti.

Klasa `NormalForm` definiše apstraktnu metodu `check` koja mora biti implementirana u svim nasleđenim klasama. Ova metoda prima standardni skup parametara: naziv tabele, zaglavlja kolona, redove podataka i specifikaciju tabele, i vraća listu objekata `NormalizationSuggestion`.

Hijerarhijska provera normalnih formi

Analizator implementira logiku prekidanja gde se analiza zaustavlja čim se pronađe narušavanje niže normalne forme. Ako prva normalna forma nije zadovoljena, sistem ne proverava drugu i treću normalnu formu jer one zavise od ispunjenosti 1NF uslova.

Slično, ako su pronađena kršenja druge normalne forme, sistem preskače proveru treće normalne forme jer 3NF zahteva da tabela već bude u 2NF. Ova logika je implementirana kroz eksplicitne provere u glavnoj metodi analize.

Prva normalna forma (1NF)

Analiza prve normalne forme implementirana je u klasi `FirstNormalForm` koja nasleđuje apstraktnu klasu `NormalForm`. Prva normalna forma zahteva da svaka ćelija u tabeli sadrži samo jednu atomsku vrednost.

Algoritam prepoznavanja multivrednosnih atributa

Algoritam implementira sistematično skeniranje svih kolona u tabeli kroz dvostruku iteraciju. Spoljnja iteracija prolazi kroz zaglavlja kolona koristeći `enumerate` funkciju koja obezbeđuje i indeks kolone i naziv. Za svaku kolonu, algoritam inicijalizuje dva brojača: `multi_value_count` za praćenje ćelija sa multivrednosnim sadržajem i `total_count` za ukupan broj nepraznih ćelija.

Unutrašnja iteracija prolazi kroz sve redove podataka i za svaki red testira da li indeks kolone postoji u redu i da li ćelija sadrži podatke. Algoritam primenjuje `str()` konverziju i `strip()` normalizaciju na sadržaj svake ćelije pre analize.

Sistem definiše niz od pet ključnih indikatora multivrednosnog sadržaja: zarez (,) koji ukazuje na liste vrednosti, tačka-zarez (;) koji se koristi za složenije liste, vertikalna crtica (|) koja je česta u izvozima podataka, novi red (\n) koji ukazuje na višelinijski sadržaj i tabulator (\t) koji se javlja u ugnežđenim strukturama.

Za svaku ćeliju sa sadržajem, algoritam iterira kroz sve indikatore koristeći `in` operator za testiranje prisustva separatora u stringu. Čim se pronađe bilo koji separator, algoritam inkrementira `multi_value_count` i prekida unutrašnju petlju kroz `break` naredbu kako bi se izbeglo višestruko brojanje iste ćelije.

Nakon kompletnog skeniranja kolone, algoritam računa odnos multivrednosnih ćelija prema ukupnom broju nepraznih ćelija. Ako ovaj odnos dostiže ili prelazi prag od 0.1 (10%), kolona se označava kao kritična za normalizaciju.

Generisanje predloga za 1NF

Kada algoritam identifikuje narušavanje Prve normalne forme, generiše se detaljni predlog kroz kreiranje objekta `NormalizationSuggestion`. Konstruktor objekta prima četiri ključna parametra: `table_name` koji identifikuje tabelu sa problemom, `suggestion_type` postavljen na „1NF" za kategorizaciju, `description` koji sadrži detaljnu dijagnozu i predlog rešenja, i `confidence` koji kvantifikuje sigurnost preporuke.

Opis predloga uključuje naziv problematične kolone, tačan procenat zahvaćenih redova formatiran sa jednim decimalnim mestom i konkretne preporuke za rešavanje narušavanja normalne forme.

Sistem predlaže dva osnovna pristupa za rešavanje 1NF kršenja. Prvi pristup podrazumeva podelu problematične kolone na više zasebnih kolona kada je broj vrednosti po ćeliji relativno mali i predvidiv. Drugi pristup predlaže kreiranje po-

sebne tabele sa nazivom konstruisanim kombinovanjem originalnog naziva kolone i sufiksa „_values", što je pogodno za složenije slučajeve sa promenljivim brojem vrednosti po entitetu.

Nivo pouzdanosti predloga `confidence` se postavlja direktno na izračunat odnos multivrednosnih ćelija (`multi_value_ratio`), čime se obezbeđuje da predlozi sa višim procentom kršenja imaju veću težinu pri konačnim odlukama o normalizaciji.

Druga normalna forma (2NF)

Analiza druge normalne forme implementirana je u klasi `SecondNormalForm` koja identifikuje parcijalne funkcionalne zavisnosti u tabelama sa kompozitnim primarnim ključevima.

Provera narušavanja 2NF

Funkcija `check` implementira kompletan algoritam za otkrivanje narušavanja druge normalne forme kroz sledeći tok izvršavanja:

Prva faza proverava osnovne preduslove za analizu. Ako tabela nema kompozitni primarni ključ (više od jedne kolone), analiza se preskače jer parcijalne zavisnosti mogu postojati samo kod kompozitnih ključeva. Takođe se proverava da li postoje validni podaci za analizu.

Druga faza mapira kolone primarnog ključa iz naziva u pozicije. Za svaku kolonu kompozitnog primarnog ključa, algoritam traži njenu poziciju u zaglavlju tabele. Ovaj korak je potreban jer kasnija analiza podataka radi sa brojevima kolona umesto njihovih naziva. Ako neka kolona primarnog ključa nije pronađena u podacima, preskače se.

Treća faza traži parcijalne zavisnosti pozivajući funkciju `find_partial_dependencies`. Ova funkcija analizira svaku ne-ključnu kolonu i testira da li zavisi od jednog dela kompozitnog ključa umesto od celog ključa.

Četvrta faza generiše predloge za normalizaciju. Za svaku pronađenu parcijalnu zavisnost, algoritam:

- Predlaže izdvajanje problematične kolone u zasebnu tabelu
- Konstruiše naziv nove tabele na osnovu naziva determinante i zavisne kolone
- Kreira detaljni opis problema sa jačinom zavisnosti

- Postavlja nivo pouzdanosti predloga na osnovu izračunate jačine zavisnosti

Algoritam vraća listu predloga za normalizaciju, gde svaki predlog sadrži konkretnu preporuku za dekompoziciju tabele radi eliminisanja parcijalnih zavisnosti.

Identifikacija parcijalnih zavisnosti

Algoritam testira svaku ne-ključnu kolonu protiv pojedinačnih komponenti kompozitnog primarnog ključa kroz funkciju `find_partial_dependencies`. Za svaku kombinaciju ne-ključne kolone i dela primarnog ključa, sistem računa jačinu determinacije kroz analizu grupa vrednosti.

Funkcija `_build_key_groups` kreira mape gde ključ predstavlja vrednost determinante (dela primarnog ključa), a vrednost predstavlja listu svih zavisnih vrednosti. Parcijalna zavisnost postoji ako svaka vrednost determinante uvek mapira na istu zavisnu vrednost.

Ocenjivanje jačine zavisnosti

Sistem računa dve ocene: jačinu parcijalne zavisnosti (na osnovu dela ključa) i jačinu pune zavisnosti (na osnovu celog ključa) kroz funkciju `_calculate_determination_strength`. Jačina se računa kao odnos determinističkih mapiranja prema ukupnom broju mapiranja.

Funkcija `_evaluate_dependency_strength` poredi ove ocene i određuje da li postoji značajna parcijalna zavisnost koja opravdava dekompoziciju tabele. Parcijalna zavisnost se smatra značajnom ako je njena ocena veća od praga `NF2_PARTIAL_DEPENDENCY_THRESHOLD`.

Predlozi za dekompoziciju tabela

Za svaku identifikovanu parcijalnu zavisnost, sistem generiše predlog za izdvajanje problematične kolone u zasebnu tabelu. Naziv nove tabele se konstruiše na osnovu naziva determinante i zavisne kolone, pri čemu se uklanjaju uobičajeni sufiksi kao što su _id i _name.

Treća normalna forma (3NF)

Analiza treće normalne forme implementirana je u klasi `ThirdNormalForm` koja identifikuje tranzitivne funkcionalne zavisnosti između ne-ključnih kolona.

Prepoznavanje tranzitivnih zavisnosti

Tranzitivna zavisnost postoji kada ne-ključna kolona A određuje ne-ključnu kolonu B, što kreira posrednu zavisnost između primarnog ključa i kolone B kroz kolonu A. Algoritam testira sve parove ne-ključnih kolona tražeći funkcionalne zavisnosti.

Funkcija `find_transitive_dependencies` iterira kroz sve ne-ključne kolone kao potencijalne determinante i testira njihove zavisnosti sa ostalim ne-ključnim kolonama. Za svaki par, algoritam kreira mapu zavisnosti i proverava konzistentnost mapiranja.

Validacija zavisnosti primarnog ključa

Pre identifikovanja tranzitivne zavisnosti, sistem mora potvrditi da determinanta funkcionalno zavisi od primarnog ključa kroz funkciju `_check_pk_dependency`. Ova provera osigurava da identifikovana zavisnost stvarno predstavlja tranzitivnu vezu kroz primarni ključ.

Funkcija testira da li svaka kombinacija vrednosti primarnog ključa jedinstveno određuje vrednost potencijalne determinante. Ako ova veza ne postoji, tranzitivna zavisnost ne može biti validna.

Računanje pouzdanosti zavisnosti

Sistem implementira sistem ocenjivanja kroz funkciju `_calculate_dependency_confidence` koji kombinuje nekoliko faktora: konzistentnost zavisnosti, jedinstvenost determinante i osnovni nivo pouzdanosti.

Konzistentnost se računa kroz funkciju `_calculate_consistency_score` koja testira da li determinanta zaista funkcionalno određuje sve identifikovane zavisne kolone. Svaka nekonzistentnost smanjuje ocenu za kaznu definisanu u konfiguraciji.

Bonus za jedinstvenost se računa kroz funkciju `_calculate_uniqueness_bonus` na osnovu odnosa jedinstvenih vrednosti determinante prema ukupnom broju redova. Veća jedinstvenost ukazuje na jaču funkcionalnu zavisnost.

Integracija sa glavnim sistemom analize

Analizator normalnih formi je integrisan u širi sistem analize tabela kao poslednji korak nakon određivanja primarnih i stranih ključeva. Analiza normalnih formi zavisi od prethodno identifikovanih primarnih ključeva za ispravno funkcionisanje.

Sistem generiše strukturisane predloge normalizacije kroz objekat `Normalization-Suggestion` koji sadrži tip kršenja, opis problema, predlog rešenja i nivo pouzdanosti. Ovi predlozi se mogu koristiti za automatsko generisanje dekomponovanih tabela ili kao smernice za manuelnu optimizaciju strukture baze podataka.

Glava 3

Pravljenje sheme baze podataka

Komponenta `ddl_generator` predstavlja finalni modul u automatskom procesu pravljenja sheme relacione baze podataka na osnovu tabelarnih podataka. Ova komponenta preuzima analizirane specifikacije tabela i generiše skripte CREATE TABLE koje mogu biti direktno izvršene na ciljanim bazama podataka.

Softver implementira klasu `DDLGenerator` koja podržava više različitih dijalekata baza podataka kroz konfigurisano mapiranje tipova i sintaksne varijacije. Generator koristi obrazac Registar za upravljanje dijalektima, što omogućava elegantno rešavanje problema kompatibilnosti između različitih sistema baza podataka. Trenutno je implementirana podrška za četiri dijalekta: PostgreSQL [5], MySQL [12], SQL Server [10] i SQLite [6].

Obrazac Registar za dijalekte baza podataka

Softver koristi obrazac Registar za podršku različitih dijalekata SQL-a, omogućavajući lako dodavanje novih baza podataka kroz konfiguraciju bez strukturnih izmena koda. Ovaj pristup rešava problem heterogenosti između različitih sistema baza podataka koji, uprkos zajedničkom SQL standardu, imaju značajne razlike u sintaksi, tipovima podataka i konvencijama.

Registar dijalekata je implementiran kroz centralnu strukturu podataka `DIALECT_CONFIGS` koja mapira svaki podržan dijalekt na odgovarajući konfiguracioni objekat. Svaki konfiguracioni objekat enkapsulira sve specifičnosti ciljane baze podataka kroz sledeće komponente:

Mapiranje tipova podataka: Svaki dijalekt definiše kako se interni tipovi podataka (INTEGER, VARCHAR, DATETIME, itd.) transformišu u odgovarajuće native tipove ciljane baze podataka. Na primer, tip DATETIME se mapira u TIMESTAMP za PostgreSQL, DATETIME za MySQL, DATETIME2 za SQL Server

i TEXT za SQLite. Ovo omogućava generatoru da proizvede sintaksno ispravanu skriptu bez obzira na ciljanu platformu.

Konvencije citiranja identifikatora: Različite baze podataka koriste različite karaktere za citiranje naziva tabela i kolona. PostgreSQL koristi dvostruke navodnike („table_name”), MySQL može koristiti i dvostruke navodnike i gravis akcente (‘table_name‘), dok SQL Server koristi uglaste zagrade ([table_name]). Registar enkapsulira ove razlike kroz konfiguracioni parametar `quote_char`.

Funkcionalnost auto-increment: Različite baze podataka implementiraju kreiranje surogat ključeva sa automatskim uvećavanjem vrednosti na potpuno različite načine. PostgreSQL koristi SERIAL tipove, MySQL dodaje AUTO_INCREMENT atribut, SQL Server koristi IDENTITY specifikaciju, dok SQLite zahteva AUTOINCREMENT ključnu reč. Registar enkapsulira ove razlike čuvanjem odgovarajuće sintakse za svaki dijalekt.

Ograničenja i validacija: Neki dijalekti imaju specifična ograničenja vezana za nazive objekata, dužinu identifikatora ili podršku za određene tipove podataka. Registar može enkapsulirati ove informacije kroz dodatne konfiguracione parametre.

Dodavanje novog dijalekta se svodi na kreiranje novog konfiguracionog objekta i registrovanje u centralnoj mapi, bez potrebe za menjanjem logike generatora. Ovo čini sistem lako proširivim i održivim, omogućavajući podršku novih baza podataka kroz jednostavne konfiguracione izmene.

3.1 Algoritam generisanja sheme baze podataka

Klasa `DDLGenerator` implementira sveobuhvatan algoritam za transformaciju analiziranih specifikacija tabela u skripte SQL/DDL. Ovaj proces predstavlja završni deo celokupnog procesa analize datoteka CSV, transformišući sve prikupljene informacije o strukturi podataka u izvršive naredbe baze podataka. Centralna metoda `generate_schema_ddl` koordiniše ceo proces kroz sledeće ključne faze:

Inicijalizacija i konfiguracija dijalekta: Generator se inicijalizuje sa specifičnim dijalektom baze podataka, što aktivira slojevitu konfiguraciju koja obuhvata prilagođavanje sintakse, tipova podataka i konvencija imenovanja. Ovaj korak uspostavlja kontekst za sve dalje transformacije podataka.

Kreiranje informativnog zaglavlja: Izlazni fajl počinje strukturiranim komentarima koji pružaju metapodatke o generisanoj skripti. Ovi komentari specificiraju ciljani dijalekt baze podataka i vremensku oznaku kada je skripta generisana.

Sekvencijalno procesiranje specifikacija tabela: Za svaku tabelu iz ulaznih specifikacija poziva se specijalizovana privatna metoda `_generate_table_ddl` koja transformiše objekat `TableSpec` u kompletnu, sintaksno ispravnu naredbu CREATE TABLE. Tabele se procesiraju sekvencijalno u determinističkom redosledu, što obezbeđuje konzistentnost između različitih pokretanja generatora i omogućava predvidljivost u izlaznoj skripti.

Integracija analitičkih predloga za normalizaciju: Algoritam automatski uključuje sve predloge za normalizaciju kao strukturirane komentare u fajlu, kreirajući direktnu vezu između funkcionalne sheme i analitičkih uvida. Svaki predlog sadrži detaljne informacije: tip normalne forme koja je narušena (1NF, 2NF, ili 3NF), precizno objašnjenje identifikovanog problema sa konkretnim imenima kolona i tabela, kao i numeričku ocenu pouzdanosti analize koja pomaže da se proceni vrednost predloga.

Generisanje definicije tabele: Metoda `_generate_table_ddl` konstruiše naredbu CREATE TABLE kroz koordinisan pristup koji obuhvata precizno definisanje svih kolona, dodavanje primarnih ključeva kao formalna ograničenja na nivou tabele i kreiranje referenci stranih ključeva sa potpunim specifikacijama ciljanih tabela i kolona. Svaka kolona se individualno transformiše pozivom specijalizovane metode `_generate_column_ddl` koja koristi formatiranje tipova podataka specifično za ciljani dijalekt, osiguravajući da se svaki interni tip podataka mapira u optimalan tip ciljane platforme.

Specifikacije kolona: Za svaku kolonu se generiše definicija koja uključuje ispravno citirani naziv kolone (prema konvencijama ciljane baze podataka), optimalno mapiran tip podataka sa izračunatom specifikacijom veličine baziranoj na stvarnim karakteristikama podataka, dijalekt-specifičnu sintaksu auto-increment funkcionalnosti (ako je potrebna i podržana), i ispravno ograničenje NOT NULL za kolone koje ne dozvoljavaju nedostajuće vrednosti.

Kreiranje ograničenja: Primarni ključevi se definišu kao formalna ograničenja na nivou tabele kroz standardizovanu klauzulu PRIMARY KEY koja eksplicitno navodi sve kolone koje čine složeni ili jednostavan ključ. Strani ključevi se implementiraju kroz klauzulu FOREIGN KEY koja specificira kako kolone u trenutnoj tabeli, tako i kompletnu referentnu tabelu sa odgovarajućim referentnim kolonama, omogućavajući bazi podataka da automatski osigurava referencijalni integritet. Ova sintaksa je standardizovana kroz različite dijalekte baza podataka.

Citiranje identifikatora: Metoda `_quote_identifier` osigurava da se svi na-

zivi tabela i kolona ispravno citiraju prema konvencijama ciljane baze podataka.

Rezultujuća skripta može se direktno izvršiti na ciljanoj bazi podataka, kreirajući kompletnu shemu koja odražava strukturu, tipove podataka, ograničenja i relacije originalnih datoteka CSV.

3.2 Primer izlaza softvera

Listing 3.1: Primer izlaza softvera

```
-- Generated DDL for postgresql
-- Generated at: 2025-09-25T14:30:45.123456

CREATE TABLE "student_course" (
    "StudentID" VARCHAR(255) NOT NULL,
    "CourseID" VARCHAR(255) NOT NULL,
    "StudentName" VARCHAR(255) NOT NULL,
    "CourseName" VARCHAR(255) NOT NULL,
    "Grade" CHAR(2) NOT NULL,
    "Credits" INTEGER NOT NULL,
    PRIMARY KEY ("StudentID", "CourseID")
);

-- NORMALIZATION SUGGESTIONS:
-- [2NF] student_course: Column 'StudentName' has partial
   dependency on 'StudentID'
--   (part of composite key ['StudentID', 'CourseID']) instead of
   depending on the full key
--   (strength: 100.0%). Consider extracting to a separate '
   students' table with columns:
--   StudentID, StudentName.
--   Confidence: 100.0%
-- [2NF] student_course: Column 'CourseName' has partial dependency
    on 'CourseID'
--   (part of composite key ['StudentID', 'CourseID']) instead of
   depending on the full key
--   (strength: 100.0%). Consider extracting to a separate 'courses
   ' table with columns:
--   CourseID, CourseName.
--   Confidence: 100.0%
-- [2NF] student_course: Column 'Credits' has partial dependency on
    'CourseID'
```

```
--   (part of composite key ['StudentID', 'CourseID']) instead of
   depending on the full key
--   (strength: 100.0%). Consider extracting to a separate '
   course_credits' table with
--   columns: CourseID, Credits.
--   Confidence: 100.0%
```

Glava 4

Diskusija i zaključak

4.1 Diskusija

Datoteke CSV su široko rasprostranjeni format za čuvanje i razmenu tabelarnih podataka u različitim domenima, od poslovnih aplikacija do naučnih istraživanja. Automatsko pretvaranje ovih datoteka u strukturirane relacione baze podataka predstavlja ključni korak u procesu upravljanja podacima. Razvoj softvera koji može pouzdano da generiše predlog projekta baze podataka iz proizvoljnih datoteka CSV zahteva kombinovanje znanja iz oblasti baza podataka, statistike i algoritamskih pristupa. Tokom razvoja ovog softvera korišćen je iterativni pristup testiranja na različitim tipovima datoteka CSV, što je omogućilo postepeno poboljšavanje preciznosti svih implementiranih algoritma.

Rezime urađenog posla

U ovom radu je razvijen celokupni softver za automatsko pretvaranje datoteka CSV u predlog projekat (dela) baze podataka. Softver rešava problem automatskog prepoznavanja strukture podataka kombinovanjem različitih pristupa analize podataka, statističkih metoda i heurističkih tehnika.

Glavni doprinosi rada uključuju algoritme za automatsko prepoznavanje zaglavlja fajlova CSV, prepoznavanje tipova podataka, otkrivanje primarnih ključeva (prirodni, kompozitni, surogat) i stranih ključeva, kao i implementaciju algoritma za normalizaciju koji otkriva narušavanja 1NF, 2NF i 3NF.

Razvijeni softver generiše SQL/DDL fajl za ulazne fajlove CSV sa podrškom za više SQL dijalekata (PostgreSQL, MySQL, SQL Server, SQLite). Modularna

arhitektura omogućava lako proširivanje i održavanje softvera.

Moguća dalja unapređenja softvera

Buduća unapređenja softvera mogu ići u nekoliko pravaca:

- **Detekcija zaglavlja** mogla bi biti poboljšana kroz implementaciju rečnika naziva kolona koji bi povećao verovatnoću prepoznavanja naziva kolona koje se ponavljaju kroz više tabela, kao i automatsko imenovanje kolona na osnovu sadržaja
- **Veličina kompozitnih ključeva** može biti promenjena da uzima u obzir i više od dve kolone
- **Primena mašinskog učenja** mogla bi poboljšati preciznost detekcije ključeva i tipova podataka kroz zamenu trenutnih heurističkih pristupa naprednim algoritmima klasifikacije
- **Implementacija obrade u tokovima** omogućila bi rad sa velikim datotekama koje se ne mogu učitati u memoriju odjednom
- **Automatska optimizacija sheme baze podataka** za bolje performanse kroz kreiranje odgovarajućih indeksa i strategija podele podataka predstavlja prirodni pravac za dalje razvijanje softvera
- **Grafički korisnički interfejs** olakšao bi korišćenje softvera krajnjim korisnicima koji nisu tehnički stručnjaci
- **Proširivanje podrške na dodatne SQL dijalekte** povećalo bi praktičnu primenu softvera

4.2 Zaključak

Automatska konverzija CSV datoteka u relacionu bazu podataka predstavlja složen zadatak koji zahteva kombinovanje različitih pristupa analize podataka. Razvijeni softver uspešno rešava ovaj problem kroz modularnu arhitekturu koja omogućava preciznu detekciju strukture podataka i generisanje validnih SQL/DDL skripti.

Implementirani algoritmi za detekciju zaglavlja, tipova podataka, primarnih i stranih ključeva, kao i analizu normalizacije, omogućavaju automatsko kreiranje

funkcionalne sheme baze podataka iz proizvoljnih fajlova CSV formata. Podrška za više SQL dijalekata čini softver prilagodljivim različitim sistemima baza podataka.

Bibliografija

[1] Python Software Foundation. csv biblioteka, 2025. on-line at: https://docs.python.org/3/library/csv.html.

[2] Python Software Foundation. Python, 2025. on-line at: https://www.python.org/.

[3] Python Software Foundation. re biblioteka, 2025. on-line at: https://docs.python.org/3/library/re.html.

[4] Python Software Foundation. statistics biblioteka, 2025. on-line at: https://docs.python.org/3/library/statistics.html.

[5] The PostgreSQL Global Development Group. PostgreSQL, 2025. on-line at: https://www.postgresql.org/.

[6] D. Richard Hipp. SQLite, 2025. on-line at: https://www.sqlite.org/.

[7] IBM. Foreign Key Constraint, 2025. on-line at: https://www.ibm.com/docs/en/db2/12.1.0?topic=constraints-foreign-key-referential.

[8] IBM. Normal Forms, 2025. on-line at: https://www.ibm.com/docs/en/tpfdf/1.1.3?topic=database-normalization.

[9] IBM. Primary Key Constraint, 2025. on-line at: https://www.ibm.com/think/topics/primary-key.

[10] Microsoft. SQL Server, 2025. on-line at: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17.

[11] Leah Nguyen. Normal Forms Examples, 2025. on-line at: https://medium.com/@ndleah/a-brief-guide-to-database-normalization-5ac59f093161.

[12] Oracle. MySQL, 2025. on-line at: https://www.mysql.com/.