Kriptiranje i sigurnost relacijskih baza podataka

Vučemilović-Vranjić, Matea

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

KRIPTIRANJE I SIGURNOST RELACIJSKIH BAZA PODATAKA

Matea Vučemilović-Vranjić

Split, rujan 2024.

Temeljna dokumentacijska kartica

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

Završni rad

KRIPTIRANJE I SIGURNOST BAZA PODATAKA

Matea Vučemilović-Vranjić

SAŽETAK

Ovaj rad istražuje ključne sigurnosne aspekte relacijskih baza podataka s naglaskom na primjenu kriptografije, kontrole pristupa i zaštite podataka. Razmatraju se različite metode za osiguranje povjerljivosti, integriteta i dostupnosti podataka te se prikazuje praktična implementacija relacijske baze podataka sa navedenim sigurnosnim mehanizmima kroz Python i SQL.

Ključne riječi: relacijska baza podataka, kriptografija, sigurnost, integritet, kontrola pristupa, Python, SQL

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

Rad sadrži: 49 stranica, 17 grafičkih prikaza, 3 tablica i 14 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: Doc. dr. sc. Goran Zaharija, docent

Ocjenjivači: Doc. dr. sc. Goran Zaharija, docent

Prof. dr. sc. Saša Mladenović, redoviti profesor

Nika Jerković, asistent

Rad prihvaćen:

Basic documentation card

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

B. Sc. Thesis

ENCRYPTION AND SECURITY OF RELATIONAL DATABASES

Matea Vučemilović-Vranjić

ABSTRACT

This paper explores key security aspects of relational databases with a focus on the application of cryptography, access control, and data protection. Various methods for ensuring confidentiality, integrity, and availability of data are examined, and a practical implementation of a relational database with these security mechanisms is demonstrated using Python and SQL.

Keywords: relational database, cryptography, security, integrity, access control, Python, SQL

Thesis deposited in the library of Faculty of Science, University of Split

Thesis consists of: 49 pages, 17 figures, 3 tables and 14 references, original in: Croatian

Mentor: Goran Zaharija, Ph.D. *Assistant Professor*

Reviewers: Goran Zaharija, Ph.D. *Assistant Professor*

Saša Mladenović, Ph.D. *Professor*

Nika Jerković, *assistant*

Thesis accepted:

IZJAVA

o samostalnoj izradi završnog rada

Izjavljujem pod punom materijalnom i moralnom odgovornošću da sam ovaj rad izradio/la samostalno te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu propisano označeni kao citati s navedenim izvorom iz kojeg su preneseni.

U Splitu, rujan 2024.

Matea Vučemilović-Vranjić

(student-ica)

Sadržaj

Uvod

Baze podataka pohranjuju, dijele i organiziraju brojne osjetljive informacije poput osobnih, poslovnih podataka i financijskih transakcija zbog čega je njihova sigurnost od ključne važnosti u današnjem digitalnom svijetu. Hakerski napadi, krađa podataka i neovlašteni pristupi postaju sve učestaliji, uzrokujući značajne gubitke i narušavanje privatnosti. Prijetnje poput *SQL injekcija* i *brute-force* napada predstavljaju ozbiljan rizik, zbog čega organizacije moraju primjenjivati napredne sigurnosne mjere kako bi zaštitile svoje podatke.

Kriptografija je ključna tehnika za zaštitu podataka, osiguravajući njihovu povjerljivost i integritet. Enkripcija omogućava pretvaranje podataka u nečitljiv oblik za neovlaštene korisnike, dok *hashiranje* lozinki i autentifikacija pružaju dodatne slojeve sigurnosti.

Cilj ovoga rada je istražiti i implementirati sigurnosne mehanizme za zaštitu relacijskih baza podataka, s naglaskom na enkripciju, te pružiti osnovni uvid u ključne pojmove vezane uz baze podataka i kriptografiju. Praktična primjena sigurnosnih metoda bit će prikazana kroz izradu sustava relacijske baze koja osigurava povjerljivost i zaštitu osjetljivih informacija.

Ovaj rad podijeljen je u pet glavnih poglavlja. U prvom poglavlju obrađuju se osnovni pojmovi vezani uz baze podataka s naglaskom na relacijske. Sljedeće poglavlje fokusira se na sigurnosne prijetnje, gdje se definiraju sigurnosni ciljevi i analiziraju različite vrste prijetnji koje mogu ugroziti sigurnost podataka. Treće poglavlje bavi se kontrolom pristupa, pri čemu su detaljno opisani procesi autentifikacije, autorizacije i dodjeljivanja korisničkih ovlasti. Pretposljednje poglavlje obrađuje kriptiranje podataka, s naglaskom na podjelu kriptografskih sustava i primjenu različitih metoda kriptiranja u samim bazama podataka. Završno, peto poglavlje donosi praktični rad u jezicima *Python* i *SQLite*, u kojima se kroz izradu relacijske baze podataka prikazuju opisani sigurnosni mehanizmi i njihova primjena u stvarnim sustavima.

1. Općenito o bazama podataka

Baza podataka (engl. *Database*) organizirana je zbirka strukturiranih informacija ili podataka koji se obično pohranjuju elektronički u računalni sustav. Takvi podaci najčešće se modeliraju u recima i stupcima u nizu tablica kako bi obrada i pretraživanje podataka bilo učinkovito. Podacima se zatim može jednostavno upravljati, mijenjati ih, ažurirati, kontrolirati i organizirati (Oracle, n.d.). Većina baza podataka za zapisivanje i postavljanje upita upotrebljava strukturirani jezik za upite (engl. *Structured Query Language*, skraćeno SQL).

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 (Manger, 2011).

Za rad baze potreban je sveobuhvatan softverski program baze podataka poznat kao sustav za upravljanje bazom podataka (engl. *Database Management System*, skraćeno DBMS). ***DBMS*** služi kao sučelje između baze podataka i krajnjih korisnika ili programa, što korisnicima omogućuje dohvat podataka, njihovo ažuriranje i upravljanje načinom organiziranja i optimizacije podataka. *DBMS* olakšava nadzor i kontrolu baza podataka, omogućujući razne administrativne operacije kao što su nadzor izvedbe, podešavanje te sigurnosno kopiranje i vraćanje.

1.1. Modeli podataka

Model podataka je način na koji su podaci u bazi logički organizirani (moraju biti u skladu s modelom koji podržava odabrani sustav). To je skup pravila koja određuju kako može izgledati logička struktura baze te čini osnovu za projektiranje i implementiranje baze. Sustavi za upravljanje bazom podataka podržavaju relacijski, mrežni, hijerarhijski ili objekti model (Manger, 2011).

- **Relacijski model** – zasnovan je na matematičkom pojmu relacije u kojem se podaci i veze među podacima prikazuju u obliku tablica.

- **Mrežni model** – u kojem je baza predočena usmjerenim grafom, gdje su čvorovi tipovi zapisa, a lukovi veze među njima.
- **Hijerarhijski model** – specijalni slučaj mrežnog u kojem je baza zapravo jedno stablo ili skup stabala.
- **Objektni model** – inspiriran je objektno-orijentiranim programiranjem, gdje je baza skup trajno pohranjenih objekata koji se sastoje od svojih internih podataka i metoda za rukovanje tim podacima.

1.1.1. Relacijski model baza podataka

Relacijski model baza podataka (engl. *Relational database model*) temelji se na matematičkom pojmu relacije, odnosno tablice, u kojoj se podaci organiziraju u redove i stupce. Svaka tablica predstavlja **entitet** koji se odnosi na određeni skup podataka, dok su stupci unutar tablice **atributi** koji opisuju karakteristike tog entiteta. Na primjer, tablica *"Klijenti"* može sadržavati atribute kao što su *"Ime", "Prezime", "Adresa"* i slično.

Ključna karakteristika relacijskog modela je njegova sposobnost da jasno definira odnose između različitih entiteta unutar baze podataka. Ti odnosi se ostvaruju kroz primarne i strane ključeve. **Primarni ključ** (engl. *Primary key*, skraćeno PK) je atribut ili skup atributa koji jednoznačno identificira svaki redak unutar tablice, dok je **strani ključ** (engl. *Foreign key*, skraćeno FK) atribut u jednoj tablici koji se odnosi na primarni ključ u drugoj tablici, stvarajući tako vezu između tablica.

Relacije među tablicama:

- **Jedan na jedan** (1:1) – svaki zapis u jednoj tablici odgovara samo jednom zapisu u drugoj tablici. Ova vrsta veze nije česta, ali se koristi kada je potrebno logički podijeliti entitet u dvije odvojene tablice. Na primjer, tablica *"Klijenti"* može biti povezana s tablicom *"Detalji o klijentima"*, gdje su podaci iz obiju tablica strogo vezani na način jedan-prema-jedan.
- **Jedan na više** (1:n) – jedan zapis u prvoj tablici može biti povezan s više zapisa u drugoj tablici. Ovo je najčešća vrsta odnosa. Na primjer, tablica *"Klijenti"* može biti povezana s tablicom *"Računi"*, gdje jedan klijent može imati više bankovnih računa, ali jedan račun pripada samo jednom klijentu.

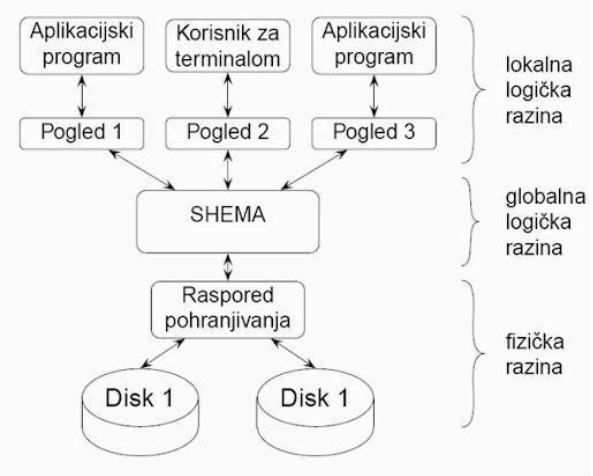
- **Više na više** (n:m) – više zapisa u jednoj tablici može biti povezano s više zapisa u drugoj tablici. Ova vrsta veze zahtjeva dodatnu poveznu tablicu (engl. *Join table*) koja sadrži primarne ključeve iz obje povezane tablice. Na primjer, filmovi mogu biti povezani s više različitih žanrova putem povezne tablice *"Filmovi_Žanrovi"*. Jedan film može imati više žanrova, te jednom žanru također može pripadati više filmova.

Jedan od aspekata relacijskog modela je i **normalizacija**, proces kojim se baza podataka dizajnira tako da se smanji redundancija podataka i osiguraju odnosi među podacima na dosljedan i učinkovit način. Normalizacija obuhvaća razdvajanje podataka u više povezanih tablica kako bi se postigla veća organiziranost i lakša manipulacija. Relacijske baze podataka su najraširenije baze podataka zbog svoje skalabilnosti, performansi i jednostavnosti. Neki od najpoznatijih sustava koji koriste relacijski model uključuju *MySQL*, *PostgreSQL*, *Oracle Database* i *Microsoft SQL Server*.

1.2. Arhitektura baze podataka

Arhitektura baze podataka sastoji se od tri sloja i sučelja među njima što je prikazano slikom (Sl. 1.1). Imamo tri razine apstrakcije.

1. **Fizička razina** – fizički prikaz i raspored podataka na jedinicama vanjske memorije koju vide samo sistemski programeri. Raspored pohranjivanja opisuje kako se elementi logičke definicije baze preslikavaju na fizičke uređaje.
2. **Globalna logička razina** – odnosi se na logičku strukturu cijele baze. To je aspekt kojeg vidi administrator odnosno projektant baze. Ovaj dio naziva se shema. Shema je tekst ili dijagram koji definira logičku strukturu baze koja je u skladu sa zadanim modelom.
3. **Lokalna logička razina** – je aspekt kojeg vidi korisnik ili programer aplikacije, a odnosi se na logičku predodžbu o dijelu baze kojim se koristi pojedina aplikacija. Opis jedne logičke definicije zove se pod-shema (tekst ili dijagram kojim se imenuju i definiraju svi lokalni tipovi podataka i veze među njima, opet u skladu s modelom).



Sl. 1.1 Arhitektura baze podataka

1.3. Jezici za rad s bazama podataka

Komunikacija između korisnika ili aplikacijskog programa i sustava za upravljanje bazama podataka odvija se putem specifičnih jezika, koji se tradicionalno dijele na tri kategorije.

- **Jezik za opis podataka** (engl. *Data Description Language*, skraćeno DDL) koriste projektanti baze podataka ili administratori za definiranje strukture baze, odnosno sheme ili pogleda.
- **Jezik za manipulaciju podacima** (engl. *Data Manipulation Language*, skraćeno DML) omogućuje programerima manipulaciju podacima u bazi, odnosno unos, brisanje, ažuriranje i dohvaćanje podataka.
- **Jezik za postavljanje upita** (engl. *Query Language*, skraćeno QL) koristi se za interaktivno pretraživanje podataka u bazi. Ovaj jezik omogućuje dohvaćanje podataka bez potrebe za definiranjem postupka koji se koristi.

Danas su ti jezici objedinjeni u jedan sveobuhvatni jezik za relacijske baze podataka – ***SQL***. Koristi se za definiranje, manipulaciju i pretraživanje podataka. Može se koristiti interaktivno ili u sklopu aplikacija, čime je postao standard za rad s bazama podataka. Iako prethodno definirani jezici nisu programski jezici, neophodni su za stvaranje i upravljanje bazom podataka, ali nisu dovoljni za razvoj kompleksnih aplikacija koje rade s podacima iz baze. U današnje vrijeme aplikacije se najčešće razvijaju u standardnim objektno orijentiranim programskim jezicima. Zbog uporabe gotovih klasa za interakciju s bazom, takva tehnika je dovoljno produktivna, a razvijeni program se lako izmjenjuje, uklapa u veće sustave ili prenosi s jedne baze na drugu.

2. Sigurnosne prijetnje

Sigurnosne prijetnje baza podataka predstavljaju ozbiljan rizik za povjerljivost, integritet i dostupnost podataka. S obzirom na sve veću količinu osjetljivih informacija pohranjenih u bazama, napadači koriste različite metode kako bi došli do tih podataka ili ih kompromitirali. U nastavku su opisane najčešće prijetnje koje ugrožavaju sigurnost baza podataka. Plan zaštite baze podataka treba sadržavati proceduru procjene rizika koja identificira prijetnje i ranjivosti te uspostavlja prikladne kontrole kako bi se ostvarili sigurnosni ciljevi također opisani u nastavku.

2.1. Sigurnosni ciljevi

Sigurnosni ciljevi definirani su po ***CIA*** trokutu odnosno modelu (engl. *Confidentiality Integrity and Availability Triad*, skraćeno CIA Triad) prikazanom na slici (Sl. 2.1). *CIA* model pruža sveobuhvatan pristup informacijskoj sigurnosti, pokrivajući najvažnije aspekte zaštite podataka. Kako bi sustavi bili učinkoviti i sigurni, svaki od ovih principa mora biti pravilno implementiran i balansiran, ovisno o specifičnim potrebama organizacije ili sustava. Kršenjem ijedne stavke ovoga modela baza podataka postaje nesigurna (Puharić et al., 2008).

1. **Povjerljivost** baze podataka odnosi se osiguravanje informacija od pristupa neovlaštenih osoba. Cilj povjerljivosti je spriječiti curenje informacija i održavanje privatnosti osjetljivih podataka.
2. **Integritet** baze podataka osigurava da su podaci točni, potpuni i pouzdani. To znači da podaci nisu neovlašteno izmijenjeni, uništeni ili oštećeni. Izmjene podataka uključuju stvaranje, umetanje, ažuriranje, mijenjanje stanja podataka i brisanje. Integritet se gubi ako dođe do neodobrenih izmjena.
3. **Dostupnost** baze podataka se odnosi na pružanje usluge i podataka osobi ili programu koji imaju prava na nju. To znači da sustavi moraju biti dizajnirani i održavani tako da omogućuju pristup podacima i uslugama bez značajnih zastoja ili prekida.

Sl. 2.1 CIA model (Panmore Institute, 2023)

2.2. Tipovi sigurnosnih prijetnji

2.2.1. *SQL* ubacivanje

Napad umetanjem *SQL* koda (engl. *SQL injection*) je vrsta ranjivosti sustava koja se javlja kada napadač može manipulirati ulaznim parametrima *SQL* upita kako bi izvršio neovlaštene radnje ili dohvatio osjetljive informacije iz baze podataka. Ova ranjivost nastaje zbog nepravilnog rukovanja korisničkim unosom od strane aplikacije, dopuštajući da baza podataka ubaci i izvrši zlonamjerne *SQL* izjave (CIS, 2011).

Za početak važno je razumijevanje koncepta ***SQL* upita** (engl. *SQL query*). Oni korisniku omogućuju interakciju s bazama podataka slanjem upita za dohvaćanje, umetanje, ažuriranje ili brisanje podataka. Na primjer, u obrascu za prijavu web aplikacije gdje korisnik upisuje svoje korisničko ime i lozinku, aplikacija može konstruirati upit na sljedeći način:

```
SELECT * FROM users WHERE korisnicko_ime = 'input_username' AND password = 'input_password';
```

U ovom primjeru, 'input_username' i 'input_password' predstavljaju vrijednosti koje je naveo korisnik. Namjera je dohvatiti zapis korisnika iz baze podataka ako se korisničko ime i lozinka podudaraju. Međutim, ako aplikacija ispravno ne provjerava valjanost ili čisti korisnički unos, napadač može manipulirati unosom kako bi ubacio zlonamjerni *SQL* kod. Na primjer, napadač može unijeti sljedeće u polje za korisničko ime:

' ' ili '1'='1'

Rezultirajući *SQL* upit bio bi:

```
SELECT * FROM users WHERE korisnicko ime = '' ILI '1'='1' AND lozinka = 'input_password';
```

U ovom slučaju, umetnuti kod uvijek se procjenjuje na istinito, učinkovito zaobilazeći provjeru lozinke. Kao rezultat toga, upit vraća sve korisničke zapise iz baze podataka, dopuštajući napadaču neovlašteni pristup.

Napadi SQL injekcijom mogu imati teške posljedice, uključujući neovlašteno otkrivanje podataka, manipulaciju podacima, pa čak i potpunu kompromitaciju temeljnog poslužitelja. Kako bi se spriječili ovakvi napadi, potrebno je koristiti **pripremljene *SQL* izjave** (engl. *prepared statements*) koje osiguravaju da se korisnički unos tretira kao podatak, a ne kao dio SQL naredbe.

Običan upit kombinira korisnički unos direktno u *SQL* kod, što omogućuje navedene napade.

```
"SELECT * FROM korisnici WHERE ime = '" + ime + "' AND lozinka = '" + lozinka + "';"
```

Dok se kod pripremljenih izjava unos prosljeđuje kao parametar, a baza podataka zna da je riječ o podatku, a ne o kodu.

```
cursor.execute("SELECT * FROM korisnici WHERE ime = ? AND lozinka = ?", (ime, lozinka))
```

2.2.2. *DoS* napad

Napad uskraćivanja usluga (engl. *Denial of Service*, skraćeno DoS) je pokušaj stvaranja resursa računala nedostupnim ili nekoristivim za legalne korisnike. To je tip sigurnosnog propusta računalnih sustava koji nužno ne rezultira krađom informacija ili bilo kojim drugim materijalnim gubitkom. *DoS* napadi imaju dvije glavne forme. Jedna od njih je prisiljavanje računala žrtve da zauzme svoje resurse tako da više ne može pružati usluge koje je trebalo pružati. Drugi način je narušavanje načina komunikacije između legalnih korisnika i računala žrtve na način da oni više ne mogu komunicirati. Iako najčešće namjerni i zlonamjerni, *DoS* napadi mogu se dogoditi i sasvim slučajno (Antončić, n.d.).

Ne postoji jedan konkretan način kako spriječiti DoS napade ali postoje mnogi načini obrane. Neki detektiraju potencijalne napade, a drugi ih sprječavaju. Jedan od načina je brisanje nepotrebnih funkcija baze podataka, tj. funkcije koje se ne koriste u namijenjenoj

uporabi pojedine baze. Smanjenjem broja protokola, usluga i komponenata koji nisu potrebni ili se uopće ne koriste, povećava se sigurnost. Manji broj elemenata jednostavno znači da je onima koji žele napasti bazu teže pronaći "rupu" u sigurnosti. Uobičajen postupak administratora baza podataka je uvođenje vremenskih ograničenja na razne upite, smanjivanje broja mogućih upita i stavljanja hardverskog limita na upite. Iako neće zasigurno zaustaviti napadača, uvelike će im otežati napad.

2.2.3. *Brute force* napadi

Napadi uzastopnim pokušavanjem (engl. *Brute force*) podrazumijevaju pokušaj neovlaštenog pristupa sustavu putem isprobavanja svih mogućih kombinacija korisničkih imena i lozinki dok se ne pronađe ispravna kombinacija. Ovi napadi su učinkoviti ako sustav nema implementirane zaštitne mehanizme kao što su složene lozinke, vremensko ograničenje između prijava ili ograničenje broja neuspjelih pokušaja prijave. Brute force napadi koriste računalne programe koji automatiziraju proces isprobavanja milijuna različitih lozinki u vrlo kratkom vremenu. Ako lozinke nisu dovoljno složene (npr. sadrže samo slova ili brojeve), napadači ih lako mogu pogoditi. Kako bi se spriječili *brute force* napadi, preporučuje se korištenje složenih lozinki, implementacija ograničenja broja pokušaja prijave te korištenje tehnika poput *CAPTCHA* provjera ili dvofaktorske autentifikacije.

CAPTCHA (engl. *Completely Automated Public Turing test to tell Computers and Humans Apart*) je sigurnosna mjera osmišljena kako bi razlikovala stvarne korisnike (ljude) od automatiziranih programa ili *botova*. Koristi se u mnogim web aplikacijama i sustavima kako bi spriječila zloupotrebu usluga, kao što su masovno slanje obrazaca, stvaranje lažnih korisničkih računa ili upravo *brute force* napadi. *CAPTCHA* koristi zadatke koje je lako riješiti za ljude, ali su teški za računalne programe. U nastavku su navedeni neki od primjera.

- Tekstualna *CAPTCHA* – sastoji se od izobličenog teksta kojeg korisnik mora prepoznati.
- *ReCAPTCHA* – koristi različite zadatke, kao što su prepoznavanje i označavanje slika ili provjeravanje jednostavnog okvira *„Nisam robot"*. Analizira se ponašanje korisnika kako bi odredili je li riječ o čovjeku ili ne, često bez potrebe za izričitim rješavanjem zadataka.

- Matematička *CAPTCHA* – u kojoj korisnici trebaju riješiti jednostavne matematičke zadatke. Cilj je osigurati da zadatak može brzo riješiti ljudski korisnik, ali ne i automatizirani program.
- Audio *CAPTCHA* – je verzija koja prikazuje zvučni zadatak za korisnike koji imaju problema s vidom. Korisnik mora slušati i prepoznati niz izgovorenih brojeva ili slova.

Dvofaktorska autentifikacija (engl. *Two-Factor Authentication*, skraćeno 2FA) je sigurnosna metoda koja zahtijeva od korisnika da potvrdi svoj identitet pomoću dva različita faktora prilikom prijave na sustav. Cilj je dodati dodatni sloj zaštite, čime se smanjuje rizik neovlaštenog pristupa, čak i ako netko probije korisničko ime i lozinku. U nastavku su navedeni neki od primjera.

- *SMS* autentifikacija – nakon unosa lozinke, korisnik prima jednokratni kod putem *SMS-a* kojega je potrebno unijeti da bi dovršio prijavu.
- Autentifikacijska aplikacija - aplikacije poput *Google Authenticatora* ili *Microsoft Authenticatora* generiraju jednokratne kodove koji se ažuriraju svake 30 sekundi.
- *E-mail* autentifikacija – nakon unosa lozinke, korisnik prima jednokratni kod ili link na svoj *e-mail*.
- Biometrijska autentifikacija – sustavi mogu koristiti otisak prsta ili prepoznavanje lica kao drugi faktor, često u kombinaciji s lozinkom.

3. Kontrola pristupa

Najosnovnija metoda zaštite osjetljivih informacija koje se čuvaju u bazi podataka je ograničenje pristupa podacima samo određenoj skupini korisnika. Na ovaj način osigurava se povjerljivost podataka. Kontrola pristupa može se ostvariti na dva načina (CIS, 2012).

1. Autentifikacijom odnosno ovjeravanjem **korisničkog imena, email-a** i **lozinke**.
2. Davanjem posebnih privilegija i **prava** specifičnim podatkovnim objektima i skupovima podataka. Unutar baze podataka to su obično tablice, pregledi, redci i stupci, a prava koja im se dodjeljuju su čitanje, pisanje ili oboje.

Općenito kontrola pristupa definirana je na tri načina:

1. **Obvezna kontrola pristupa** (engl. *Mandatory Access Control*, skraćeno MAC).
2. **Diskretna kontrola pristupa** (engl. *Discretionary Access Control*, skraćeno DAC).
3. **Kontrola pristupa zasnovana na ulogama** (engl. *Role Based Access Control*, skraćeno RBAC).

MAC i *DAC* daju privilegije određenim korisnicima ili grupama koje sadrže korisnike kojima se želi dati pristup. *MAC* pravila se primjenjuju na razini sustava i smatraju se sigurnijima. *DAC* pravila se primjenjuju na razini korisnika, smatraju se dinamičkim, a usmjerena su na sadržaj. To su vrlo moćni alati no *RBAC* je posebno učinkovit u zaštiti *DBMS-a*. Analogan je funkcijama na poslu. Svaka uloga, ima svoja vlastita prava te sadrži ograničenja. Prava uključuju naredbe odabira podataka, modificiranje podataka ili manipuliranje strukture baze podataka (CIS, 2012).

Kontrola pristupa posjeduje i nadogradnju pod nazivom *„odobri/povuci odobrenje“* (engl. *grant/revoke*) koja omogućuje dinamično davanje dozvole pristupa određenom korisniku. Problem koji nastaje pri ovakvom pristupu sigurnosti je mogućnost davanja privilegija korisniku koji zapravo nema dobre namjere te može načiniti štetu na sustavu, a korištenje se dodatno komplicira ako korisnici često moraju mijenjati uloge.

Odobrenje prava pristupa administratori obavljaju putem *SQL-a*. Naredbe na koje se mogu primijeniti prava pristupa su *EXECUTE* (pravo izvršavanja procedura ili skripti u bazi podataka), *INSERT* (pravo dodavanja novih podataka), *SELECT* (pravo čitanja podataka iz

baze), *DELETE* (pravo brisanja podataka) i *UPDATE* (pravo ažuriranja postojećih podataka). Izgled naredbe za odobravanje i oduzimanje prava pristupa je sljedeći:

```
GRANT privileges

[ON relation]

TO users

[WITH GRANTOPTION]
```

U navedenoj relaciji prvo se nalazi osnovna naredba *GRANT* koja daje prava pristupa, zatim *"privileges"* vrsta privilegije koja se daje relaciji (*ON relation*) koja sadrži neku ili sve spomenutih naredbi. Na kraju naredbe navodi se ime korisnika *("TO users"*) kojem se daju prava pristupa te dodatne opcije. Naredba *REVOKE* uklanja prava korisnicima i njena je sintaksa vrlo slična:

```
REVOKE privileges

[ON relation]

FROM users

[WITH GRANTOPTION]
```

3.1. Autentifikacija i lozinke

Autentifikacija je proces provjere identiteta korisnika prije nego što im se omogući pristup bazi podataka. Cilj je osigurati da samo ovlašteni korisnici mogu ući u sustav. Najčešći način autentifikacije je korištenje korisničkog imena i lozinke, pri čemu je važno da lozinke budu složene i sigurne. S obzirom na sve češće napade na lozinke (npr. spomenuti *brute force* napad), sve više sustava koristi naprednije metode autentifikacije, poput dvofaktorske autentifikacije (*2FA*) također spomenute u poglavlju o sigurnosnim prijetnjama. U ovom slučaju, osim korisničkog imena i lozinke, korisnik mora unijeti i dodatni faktor, poput jednokratne lozinke koja se šalje na mobilni uređaj ili se generira u autentifikacijskoj aplikaciji. Na taj način, čak i ako napadač dođe do lozinke, bez drugog faktora ne može pristupiti sustavu. U modernim sustavima autentifikacija može biti integrirana vanjskim uslugama poput *LDAP* (engl. *Lightweight Directory Access Protocol*) ili Otvorene autorizacije (engl. *Open Authorization*, skraćeno OAuth).

LDAP je protokol koji omogućuje pristup i upravljanje informacijama pohranjenim u direktorijskoj službi, kao što je baza podataka s informacijama o korisnicima, uređajima ili

resursima. Često se koristi za autentifikaciju u velikim mrežnim sustavima, gdje postoji potreba za centraliziranim upravljanjem korisničkim računima. U osnovi, *LDAP* omogućuje administraciju korisničkih računa s jedne centralizirane točke. To znači da svaki korisnik ima jedan korisnički račun (jedinstvene vjerodajnice) koji se koristi za pristup različitim sustavima i aplikacijama unutar organizacije. Ovo značajno pojednostavljuje upravljanje korisničkim pravima i ovlastima te smanjuje rizik od neovlaštenog pristupa, jer administrator može jednostavno upravljati svim korisničkim računima iz jedne središnje baze.

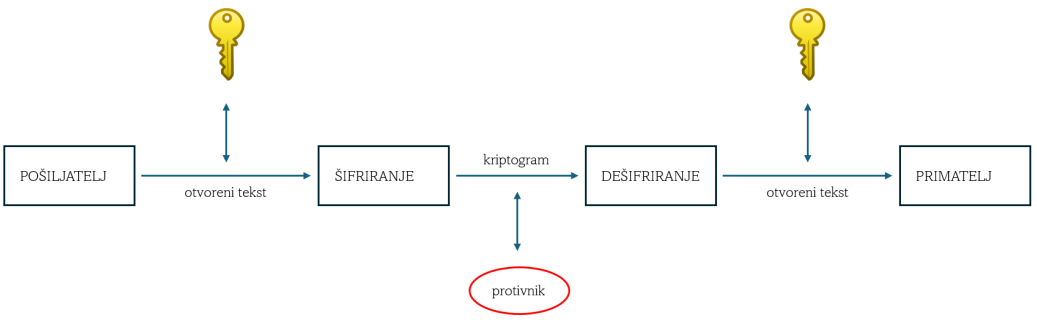
OAuth je protokol za autorizaciju koji omogućuje aplikacijama pristup resursima korisnika na siguran način, bez dijeljenja njihovih vjerodajnica (korisničkog imena i lozinke). Često koristi u situacijama gdje korisnik želi omogućiti jednoj aplikaciji pristup podacima koje ima u drugoj aplikaciji, a da pritom ne otkrije svoje vjerodajnice. Primjer korištenja je kada korisnik želi pristupiti nekoj aplikaciji koristeći svoj *Google*, *Facebook* ili sličan račun. Omogućuje aplikacijama da dobiju token pristupa, koji predstavlja dozvolu za pristup određenim resursima korisnika, ali bez potrebe da korisnik izravno dijeli svoje vjerodajnice. Ovo povećava sigurnost, jer aplikacije ne moraju pohranjivati osjetljive informacije poput lozinki, već koriste sigurnosni token koji ima ograničeni rok trajanja i može biti opozvan u bilo kojem trenutku.

4. Kriptiranje podataka

Kriptiranje podataka predstavlja ključnu metodu zaštite podataka u bazama, omogućujući da podaci postanu nečitljivi neovlaštenim korisnicima bez odgovarajućeg ključa za dešifriranje. Kriptografija se koristi kako bi se osigurala povjerljivost podataka, a u kombinaciji s kontrolom pristupa i integritetom podataka pruža potpunu zaštitu od neovlaštenih pristupa i manipulacija.

Kriptografija (Sl. 4.1) je znanstvena disciplina koja se bavi proučavanjem metoda za slanje poruka u takvom obliku da ih samo onaj kome su namijenjene može pročitati. Sama riječ kriptografija je grčkog podrijetla i mogla bi se doslovno prevesti kao *„tajnopis“* (Dujella et al., 2007).

Osnovni zadatak kriptografije je omogućiti dvjema osobama (**pošiljatelju i primatelju**) komunikaciju preko nekog komunikacijskog kanala kao što je telefonska linija, računalna mreža i slično na način da treća osoba (**protivnik**), koja može nadzirati taj komunikacijski kanal, ne može razumjeti njihove poruke. Poruka koju pošiljatelj šalje zove se **otvoreni tekst** (engl. *plaintext*). Taj tekst transformira se koristeći unaprijed dogovoreni **ključ** (eng. *key*). Ovakav postupak naziva se **šifriranje**, a dobiveni rezultat šifrat ili **kriptogram** (eng. *ciphertext*). Pošiljatelj zapravo šalje šifrat preko nekog komunikacijskog kanala. Protivnik može doznati sadržaj šifrata, ali ne može odrediti otvoreni tekst. Za razliku od njega, primatelj koji zna ključ kojim je šifrirana poruka može dešifrirati šifrat i odrediti otvoreni tekst.



Sl. 4.1 Shema klasične kriptografije

Za razliku od dešifriranja, **kriptoanaliza** ili dekriptiranje (engl. *cryptanalysis*) je znanstvena disciplina koja se bavi proučavanjem postupaka za čitanje skrivenih poruka bez poznavanja ključa. Osnovna pretpostavka kriptoanalize je da kriptoanalitičar zna koji se

kriptosustav koristi. Čak i ukoliko kriptoanalitičar treba provjeriti nekoliko mogućih kriptosustava, time se kompleksnost procedure bitno ne mijenja. Dakle, pretpostavljamo da tajnost šifre u potpunosti leži u ključu. Razlikujemo četiri osnovna nivoa kriptoanalitičkih napada (Dujella et al., 2007).

1. Kriptoanalitičar posjeduje samo šifrat od nekoliko poruka šifriranih pomoću istog algoritma. Njegov je zadatak otkriti otvoreni tekst od što više poruka ili u najboljem slučaju otkriti ključ kojim su poruke šifrirane.
2. Kriptoanalitičar posjeduje šifrat neke poruke, ali i njemu odgovarajući otvoreni tekst. Njegov je zadatak otkriti ključ ili neki algoritam za dešifriranje poruka šifriranih tim ključem.
3. Kriptoanalitičar ima mogućnost odabira teksta koji će biti šifriran, te može dobiti njegov šifrat. Ovaj napad je jači od prethodnog, ali je manje realističan.
4. Kriptoanalitičar je dobio pristup alatu za dešifriranje, pa može odabrati šifrat te dobiti odgovarajući otvoreni tekst. Ovaj napad je tipičan kod kriptosustava s javnim ključem. Tu je zadatak kriptoanalitičara otkriti tajni ključ.

Kriptologija je grana znanosti koja obuhvaća kriptografiju i kriptoanalizu. **Kriptografski algoritam** ili **šifra** je matematička funkcija koja se koristi za šifriranje i dešifriranje. Zapravo, radi se o dvije funkcije, jednoj za šifriranje, a drugoj za dešifriranje. Te funkcije preslikavaju osnovne elemente otvorenog teksta (slova, bitove i slično) u osnovne elemente šifrata, i obratno. Funkcije se biraju iz određene familije funkcija u ovisnosti o ključu. Skup svih mogućih vrijednosti ključeva nazivamo **prostor ključeva**. **Kriptosustav** se sastoji od kriptografskih algoritama, te svih mogućih otvorenih tekstova, šifrata i ključeva (Dujella et al., 2007).

4.1. Podjela kriptosustava

Kriptografski sustavi dijele se prema različitim kriterijima, ovisno o vrsti operacija koje koriste, načinu obrade podataka i načinu upravljanja ključevima. Razumijevanje ovih podjela ključ je za primjenu odgovarajućih kriptografskih tehnika u bazi podataka.

4.1.1. Podjela prema tipu operacija

Ova podjela sastoji se od supstitucijskih šifri u kojima se svaki element otvorenog teksta zamjenjuje s nekim drugim elementom, te transpozicijske šifre u kojima se elementi otvorenog teksta permutiraju. Postoje i kriptosustavi koji kombiniraju ove dvije metode.

Prije same demonstracije rada enkripcijskih algoritama, prikazat će se princip rada jednog od najstarijih algoritama enkripcije, poznatog pod nazivom **Cezarova šifra**. Šifriranje funkcionira na bazi nekog pisma i nekog broja k. Ukoliko je baza engleska abeceda, koja je dugačka 26 slova, tada broj k može biti broj između 0 i 25. U Cezarovoj šifri, originalni se znak mijenja drugim znakom koji je u abecedi k mjesta udaljen od originalnog znaka (Kurose et al., 2010). Taj broj k je pritom vrijednost ključa koji ulazi u algoritam kriptiranja.

Recimo da je ključ k jednak broju 3. Za taj ključ, slovo 'A' postaje 'D', jer je ono udaljeno tri mjesta od 'A'. Slovo 'B' postaje 'E', 'C' postaje 'F', a 'Z' postaje 'C' jer šifra ide u krug što je i prikazano u tablici (Tablica 4.1).

Tablica 4.1 Otvoreni tekst i šifrat za k=3 kod Cezarove šifre

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Cezarova šifra dekriptira poruku istim ključem koji je ulaz u algoritam dekriptiranja, no u obrnutom smjeru, odnosno za svako slovo gleda se tri mjesta unatrag. Tim postupkom od šifrata ponovno dobivamo otvoreni tekst.

Cezarova šifra zapravo je slučaj **supstitucijske šifre** koja svaki znak u izvornom tekst zamjenjuje drugim znakom prema unaprijed definiranim pravilima. Zajedno spadaju u simetrične enkripcijske metode jer koriste isti ključ za enkripciju i dekripciju (o kojima se više informacija nalazi u poglavlju 4.1.3).

Transpozicijske šifre koriste tehniku permutacije slova pa je šifrirani tekst zapravo anagram izvornog teksta. Najjednostavniji primjer je kolonska transpozicija. Uzeta je poruka „SIGURNOST“ koja će se šifrirati koristeći ključ od 3 stupca (Tablica 4.2).

Tablica 4.2 Kolonska transpozicija za riječ SIGURNOST

S	I	G
U	R	N
O	S	T

Iz tablice tekst se čita stupac po stupac pa je time šifrirani tekst „SUOIRSGNT“. Ovdje je poruka šifrirana tako što su slova preuređena na temelju stupaca. Transpozicijske šifre ne mijenjaju učestalost pojavljivanja slova, pa su time ranjive na složenije napade.

4.1.2. Podjela prema načinu obrade teksta

U ovoj podjeli postoje blokove šifre, kod kojih se obrađuje jedan po jedan blok elemenata otvorenog teksta koristeći jedan te isti ključ. Također, postoje i protočne šifre kod kojih se elementi otvorenog teksta obrađuju jedan po jedan koristeći paralelno generirani niz ključeva. **Blok šifra** je metoda kriptiranja koja funkcionira po principu da se poruka koja se šifrira dijeli na blokove određene veličine u bitovima, na primjer dijeli se u 64-bitni blok, i svaki od tih blokova se kriptira nezavisno od drugog bloka (Kurose et al., 2010).

Blok šifre često za dodatnu sigurnost koriste tehniku ulančavanja šifriranih blokova (engl. *Cipher Block Chaining,* skraćeno CBC) koja se temelji na prenošenju jedne nasumične vrijednosti zajedno s prvom porukom kako se prelazi na računanje sljedećeg bloka poruke. Umjesto da se za sljedeći blok uzima novi nasumični *string*, zadnji kodirani blok koristi se kao novi ulaz za kriptiranje sljedećeg bloka. Ako se svaki blok kriptira zasebno, moguće je da dva bloka ispadnu kompletno jednaka, odnosno neki dijelovi bloka mogu biti posve jednaki. Napadač može iskoristiti svako ponavljanje znakova i svaku grešku kako bi odgonetnuo sakrivenu informaciju. *CBC* taj dio rješava tako da je svaki sljedeći blok zasigurno drukčiji od prethodnog.

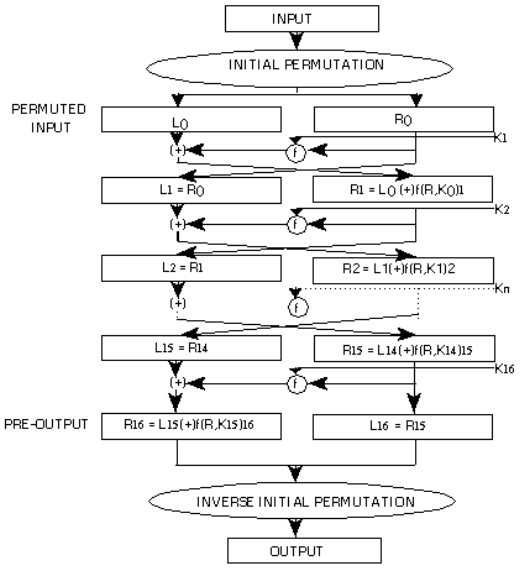
Za razliku od njih, **protočne šifre** kriptiraju podatke bit po bit ili znak po znak u realnom vremenu, generiraju niz pseudo-slučajnih ključeva koji se koriste za enkripciju svakog pojedinačnog bita ili znaka izvornih podataka. Najčešće su korištene za mrežne komunikacije, bežične mreže, šifriranje multimedijskog sadržaja, odnosno aplikacije u kojima je potrebna brza i efikasna enkripcija u realnom vremenu.

4.1.3. Simetrični kriptosustavi

Simetrični kriptografski sustavi koriste isti ključ za kriptiranje i dekriptiranje podataka. Algoritmi nisu tajni, međutim ključ za šifriranje jest. To znači da isti ključ koji se koristi za šifriranje mora biti poznat i pošiljatelju i primatelju poruke kako bi se podaci ispravno dešifrirali. Simetrična kriptografija je brza i učinkovita, osobito za šifriranje velikih količina podataka. Međutim, postoje i neki nedostaci. Prvenstveni problem predstavlja distribucija ključeva. Odnosno, kako sigurno prenijeti ključ između pošiljatelja i primatelja bez rizika da ga neovlaštene osobe presretnu. Također, potrebno je češće mijenjati ključeve jer su kraći pa je samim time veća mogućnost razbijanja šifre.

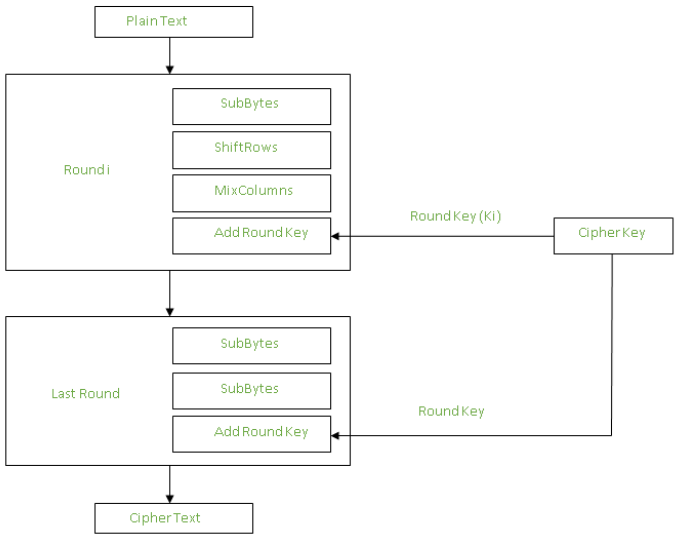
Najpoznatiji simetrični algoritmi korišteni u sustavima za upravljanje bazama podataka su:

- **Standardni algoritam za kriptiranje** (engl. *Data Encryption Standard*, skraćeno DES) je algoritam blok-šifre koji koristi 56-bitne ključeve uz 8 bita za provjeru. *DES* uzima 64-bitni otvoreni tekst i pretvara ga u 64-bitni šifrat uz korištenje istog ključa za kriptiranje i dekriptiranje poruke. Obavlja se prva permutacija, odnosno mijenjanje mjesta bitova po bloku, nad originalnim tekstom. Za sljedeću permutaciju, permutirani blok se dijeli na dvije polovice, i svaka od tih polovici 16 puta ulazi u proces kriptiranja. Na kraju se polovice spajaju i događa se još jedna permutacija nad kombiniranim blokom. Rezultat je 64-bitni šifrirani tekst. Princip rada prikazan je na slici (Sl. 4.2). Zbog kratkog ključa, *DES* je postao ranjiv na *brute force* napade te je s vremenom bila potrebna zamjena.



Sl. 4.2 DES algoritam (Engineering Ebook, n.d.)

- Godine 2002. pojavljuje se **napredni standard šifriranja** (engl. *Advanced Encryption Standard*, skraćeno AES). *AES* je blokovna šifra koja šifrira podatke u fiksnim blokovima od 128 bitova. To znači da svaki blok podataka ima fiksnu duljinu od 128 bitova, bez obzira na duljinu izvornog teksta. Ako je tekst kraći od 128 bitova, koristi se tehnika popunjavanja za popunjavanje praznog prostora. Princip rada prikazan je slikom (Sl. 4.3).



Sl. 4.3 AES algoritam (Geeks for Geeks, 2024)

4.1.4. Asimetrični kriptosustavi

Asimetrični kriptografski sustavi koriste dva ključa: javni ključ za enkripciju i privatni ključ za dekripciju. Javni ključ može biti slobodno dostupan svima, dok privatni ključ mora ostati tajan i poznat samo vlasniku. Ova metoda uklanja potrebu za sigurnim dijeljenjem ključa između pošiljatelja i primatelja koji postoji kod simetričnih kriptosustava. Međutim, asimetrični algoritmi sporiji su zbog veće računarske složenosti i zahtijevaju više resursa i složeniju infrastrukturu za upravljanje ključevima.

Najpoznatiji asimetrični algoritam je ***RSA*** (*Rivest-Shamir-Adleman*). Za kreiranje javnog i privatnog ključa biraju se dva velika prosta (neka su to p i q) broja nad kojima se obavlja niz operacija. Množenjem tih dvaju brojeva dobije se broj modul n. Broj n postaje dio javnog i privatnog ključa. Veličina tog broja zapravo određuje snagu ključa. Nakon toga računa se Eulerova funkcija.

$$\varphi(n) = (p - 1) * (q - 1)$$

Ova vrijednost koristi se za generiranje javnog i privatnog ključa. Zatim se odabire javni ključ (e) koji je relativno prost **s** φ(n), te se izračunava privatni ključ (d) koji zadovoljava sljedeći matematički uvjet:

$$d * e \equiv 1 \pmod{\varphi(n)}$$

Javni ključ je tada par (e, n), dok je privatni ključ par (d, n). Kriptiranje podataka obavlja se funkcijom

$$c = m^e \bmod n$$

pri čemu je c kriptirana poruka, a m originalna poruka. Nadalje, kako bi se poruka dekriptirala, potreban je privatni ključ koji je u obliku para (n, d). Funkcija dekriptiranja je

$$m = c^d \bmod n$$

4.1.5. Ostale metode kriptiranja

Sustavi za upravljanje bazama podataka osim kriptiranja simetričnim i asimetričnim enkripcijskim algoritmima nude i kriptiranje na temelju javnih certifikata i *hash* metoda.

Digitalni potpis je elektronički potpisana izjava koja veže vrijednost javnog ključa s osobom, odnosno tijelom, koje drži korespondirajući, matematički ovisni privatni ključ. Bitna primjena digitalnih potpisa je **javni certifikat**, koji označava pripadnost javnog ključa specifičnom entitetu (Kurose et al., 2010). Tijelo koje izdaje certifikate i koje ih potpisuje naziva se Certifikacijsko tijelo (engl. *Certification Authority*). Jako važno svojstvo digitalnih potpisa, osim autentičnosti i integriteta, jest neporecivost. Potpisnik ne može kasnije poreći da je potpisao dokument budući da je samo on imao pristup svom privatnom ključu.

Hashing je tehnika koja na temelju ulazne vrijednosti pomoću određenih algoritama računa neku *hash* vrijednost. Uzima se tekst proizvoljne duljine nad kojim primijenimo takozvanu *hash* funkciju. Ključni je princip u tehnici da će algoritam uvijek na kraju dati istu *hash* vrijednost za isti ulaz, bez obzira na broj prolaza kroz funkciju i računalo s kojeg se postupak događa. Prilikom spremanja vrijednosti u bazu podataka, ne spremaju se originalni znakovi, već sama *hash* vrijednost, i prilikom usporedbe dovoljno je izračunati *hash* vrijednost nekog niza i usporediti tu vrijednost s onom u bazi (Maleković et al., 2016). Ukoliko su vrijednosti jednake, radi se o jednakim podacima. Ono što tehniku čini sigurnom jest da je *hashing* jednosmjerna funkcija. Vrijednosti se pomoću te tehnike mogu

izračunati, no *hash* se ne može iskoristiti kako bi se reproducirao originalni sadržaj i stoga služi striktno uspoređivanju.

4.2. Kriptiranje podataka spremljenih u bazi

U bazama se podataka razlikuje kriptiranje podataka koji putuju mrežom i kriptiranje podataka koji su spremljeni u bazi. U ovom dijelu razmotrit će se kriptiranje podataka na razini baze zbog praktične implementacije iste.

Kriptiranje podataka na razini baze podataka omogućuje tzv. **selektivno kriptiranje**, pri čemu se odabiru dijelovi baze koji će biti kriptirani i metode koje će se za to koristiti. Ovaj oblik kriptiranja temelji se na konceptu zrnatosti (engl. *granularity*), što označava razinu detalja na kojoj se kriptiranje provodi (Coles et al., 2009).

Razine kriptiranja mogu varirati, ovisno o sustavu za upravljanje bazama podataka, a u literaturi se obično spominju tri do četiri razine kriptiranja.

1. **Kriptiranje na razini polja** (engl. *cell-level encryption*) je najniža razina kriptiranja, gdje se svaka vrijednost atributa šifrira pojedinačno, često uz korištenje jedinstvenih ključeva za svaku vrijednost.
2. **Kriptiranje na razini reda** (engl. *row-level encryption*) je razina kriptiranja koja šifrira svaki red zasebno, pri čemu svaki red može imati vlastiti ključ. Ova metoda omogućuje dohvaćanje samo specifičnih redova.
3. **Kriptiranje na razini stupca** (engl. *column-level encryption*). Ovdje se šifriraju specifični stupci koji sadrže osjetljive podatke, pri čemu svi redovi u stupcu dijele isti ključ za šifriranje što omogućuje ciljanje samo osjetljivih informacija.
4. **Kriptiranje na razini tabličnog prostora** (engl. *table space-level encryption*) je razina koja podrazumijeva kriptiranje cijelih tablica i njihovog sadržaja koristeći jedan ključ po tablici.

Preporučuje se kriptiranje samo onih dijelova baze koji sadrže poslovno osjetljive ili povjerljive podatke.

Jedan od izazova pri implementaciji kriptiranja na razini baze podataka jest mogućnost promjene strukture baze, što može utjecati na performanse sustava. Na primjer, atributi poput primarnih i stranih ključeva koji su kriptirani moraju se dešifrirati prilikom izvršavanja *JOIN* i *WHERE* upita, što može uzrokovati dodatno opterećenje baze podataka.

Ovi upiti također zahtijevaju dešifriranje podataka kako bi se moglo provesti sortiranje ili filtriranje rezultata, što povećava potrošnju procesora.

Također, pri implementaciji kriptiranja može doći do potrebe za promjenom tipova podataka unutar baze kako bi se podržalo pohranjivanje kriptiranih podataka. Tipovi podataka kao što su *BINARY, VARBINARY* i *CHAR* često se koriste za pohranjivanje kriptiranih vrijednosti, budući da omogućuju pohranu znakova i binarnih podataka u šifriranom obliku. Međutim, implementacija kriptiranja može zahtijevati značajne promjene u strukturi baze podataka, što može utjecati na kompatibilnost s postojećim aplikacijama i povećati složenost upravljanja sustavom. Kriptiranje stupaca omogućuje šifriranje osjetljivih stupaca bez potrebe za šifriranjem cijele tablice, što može značajno smanjiti utjecaj na performanse sustava. Kriptiranje na razini reda ili stupca može smanjiti potrošnju procesora jer se šifriraju samo osjetljivi podaci, dok ostali podaci ostaju nekriptirani. Međutim, kako se podaci kriptirani na ovaj način koriste u upitima poput *JOIN* i *WHERE*, potrebno je dekriptirati cijeli stupac ili red prije obrade upita.

U konačnici, implementacija kriptiranja unutar baza podataka predstavlja ključan aspekt zaštite osjetljivih podataka, no zahtijeva pažljivu razmatranje kako bi se održala ravnoteža između sigurnosti i performansi sustava.

5. Praktični rad

Praktični dio ovoga rada sastoji se od izrade relacijske baze podataka za bankovne sustave i razvoja desktop aplikacije koja omogućuje upravljanje i pregled podataka baze (što od strane bankara tj. administratora, što od strane samoga korisnika banke). Projekt uključuje implementaciju različitih sigurnosnih mjera kao što je kriptiranje osjetljivih podataka, autentifikacija korisnika, hashiranje lozinki i kontrola pristupa.

5.1. Korištene tehnologije

Prilikom razvoja baze podataka i aplikacije korištene su razne tehnologije koje su odigrale ključnu ulogu u osiguravanju funkcionalnosti, sigurnosti i učinkovitosti sustava. Svaka tehnologija ima specifičnu svrhu i omogućuje optimizaciju određenih aspekata rada aplikacije.

- ***SQLite*** – lagani, ugrađeni sustav za upravljanje bazama podataka (*DBMS*) koji omogućuje lokalno pohranjivanje podataka bez potrebe za postavljanjem poslužitelja. Korišten je za kreiranje i upravljanje tablicama unutar baze podataka. Prikladan je za desktop aplikacije jer omogućuje brz i jednostavan pristup podacima bez dodatne složenosti te podržava standardne *SQL* upite, što olakšava manipulaciju podacima i njihovo dohvaćanje.
- ***Python*** – programski jezik visoke razine korišten kao glavni alat u izradi aplikacije. Njegove brojne biblioteke čine ga idealnim za razvoj poslovnih aplikacija. U ovome projektu on omogućava interakciju s bazom podataka, manipulaciju podacima te implementaciju ostale logike kao što je autentifikacija, kriptiranje i izrada korisničkog sučelja. Korištenjem modula *sqlite3* omogućeno je povezivanje aplikacije s *SQLite* bazom podataka.
- ***Tkinter*** **i** ***CustomTkinter*** – standardna *Python* biblioteka za izradu grafičkih korisničkih sučelja (engl. *Graphical user interface*, skraćeno GUI). Korišten je za izradu svih komponenti aplikacije poput prozora aplikacije, formi za unos podataka, botuna i slično. *CustomTkinter* je nadogradnja na *Tkinter* koja omogućuje naprednije stiliziranje i moderniji dizajn aplikacija. Kombinacija ovih alata

omogućuje razvoj jednostavnog i intuitivnog korisničkog sučelja prilagođenog krajnjim korisnicima.

- ***PIL (Python Imaging Library)*** – biblioteka za rad sa slikama koja je korištena za generiranje *CAPTCHA* slika. Također omogućava manipulaciju slikama, crtanje teksta i kreiranje nasumičnih sigurnosnih kodova koji se prikazuju na slikama.
- ***Bcrypt*** – biblioteka koja omogućuje sigurnost korisničkih lozinki. Koristi se za sigurno hashiranje lozinki prije nego što se one pohrane u bazu podataka.
- ***Cryptography (AES)*** – biblioteka koja kriptira podataka uz pomoć simetričnog kriptografskog algoritma AES. Koristi se za kriptiranje podataka kao što su OIB, adrese korisnika, brojevi kartica i slično. Osigurano je da osjetljivi podaci ne budu dostupni u svom originalnom obliku čak i ako se baza kompromitira.
- ***PyOTP*** – *Python* biblioteka za generiranje jednokratnih kodova (engl. *One Time Password*, skraćeno *OTP*) kod implementacije dvofaktorske autentifikacije. Konkretno u ovome projektu osigurava da samo ovlašteni zaposlenici, koji imaju pristup odgovarajućoj aplikaciji, mogu završiti prijavu.
- ***Qrcode*** – biblioteka korištena za generiranje *QR* kodova prilikom implementacije dvofaktorske autentifikacije koji olakšavaju korisnicima skeniranje tajnih ključeva u aplikacijama za generiranje *OTP-ova*, što značajno pojednostavljuje proces povezivanja korisnika s aplikacijom za autentifikaciju.

5.2. Dizajn baze podataka

Projektiranje baze podataka ključno je za osiguravanje njezine učinkovitosti, skalabilnosti i sigurnosti. Proces dizajna započinje definiranjem konceptualne sheme, koja predstavlja apstraktnu strukturu podataka bez detalja implementacije, te prelazi na logičku razinu, gdje se definiraju detalji poput relacija, tipova podataka i ključnih ograničenja. Zadnja faza je kreiranje fizičkog modela u relacijskoj bazi podataka. Svaka od ovih faza ima određene značajke opisane tablicom (Tablica 5.1).

Tablica 5.1 Značajke razina projektiranja baze podataka

Značajka	Model		
	Konceptualni	Logički	Fizički
Imena entiteta	x	x	
Veze	x	x	
Atributi		x	
Primarni ključevi		x	x
Strani ključevi		x	x
Imena tablica			x
Imena stupaca			x
Tipovi podataka			x

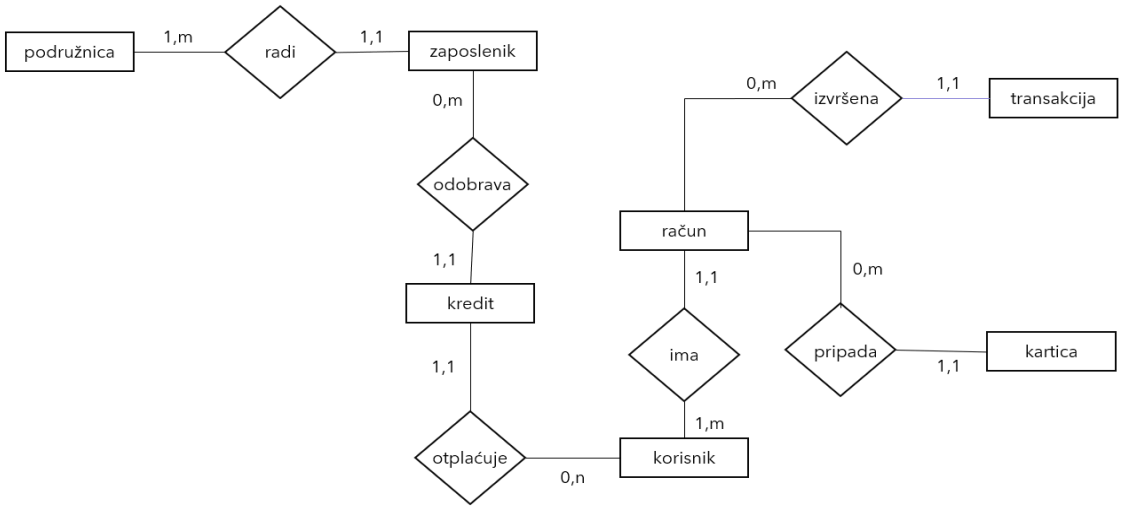
Glavni cilj **konceptualne faze** jest stvoriti **konceptualnu shemu** baze, sastavljenu samo od entiteta i veza među njima. Konceptualna shema daje jezgrovit prikaz baze koji je oslobođen tehničkih detalja. Važno svojstvo konceptualne sheme je da je ona razumljiva ljudima svih struka te da može slutiti kao sredstvo za komunikaciju projektanata i korisnika (Manger, 2011).

Nakon analize potrebnih elemenata aplikacije za bankovne sustave, konceptualna shema izgleda na sljedeći način.

- **Tipovi entiteta** su: KORISNIK, ZAPOSLENIK, PODRUŽNICA, RAČUN, KARTICA, TRANSAKCIJA i KREDIT
- **Veze** su (svaka veza je oblika jedan naprama više):
  - IMA između KORISNIK i RAČUN (jedan korisnik može imati jedan ili više računa, dok jedan račun pripada jednom korisniku)
  - PRIPADA između RAČUN i KARTICA (jedan račun može imati nijednu, jednu ili više kartica, međutim jedna kartica pripada jednom računu)
  - IZVRŠENA između RAČUN i TRANSAKCIJA (jedan račun može imati nijednu, jednu ili više transakcija, dok jedna transakcija pripada jednom računu)

- OTPLAĆUJE između KORISNIK i KREDIT (jedan korisnik može otplaćivati nijedan, jedan ili više kredita, dok jedan kredit otplaćuje jedan korisnik)
- RADI između PODRUŽNICA i ZAPOSLENIK (u jednoj podružnici rade jedan ili više zaposlenika, ali jedan zaposlenik radi samo u jednoj podružnici)
- ODOBRAVA između ZAPOSLENIK i KREDIT (jedan zaposlenik može odobriti nijedan, jedan ili više kredita, ali jedan kredit je odobren od strane samo jednog zaposlenika)

Nakon određivanja entiteta i veza među njima, idući korak je prikaz ovih elemenata u obliku dijagrama. U ovom slučaju koristit će se reducirani **Chenov dijagram** (Sl. 5.1), u kojem su tipovi entiteta nacrtani kao pravokutnici, a veze kao rombovi. Imena entiteta i veza upisana su u odgovarajuće pravokutnike odnosno rombove. Da bi se znalo između kojih je tipova entiteta uspostavljena određena veza, odgovarajući romb povezan je spojnicama s odgovarajućim pravokutnicima. Uz svaku vezu utvrđujemo i dvije **kardinalnosti**, za jedan i drugi smjer. Kardinalnost je nemoguće točno izraziti pa se navodi interval u obliku donje i gornje granice.



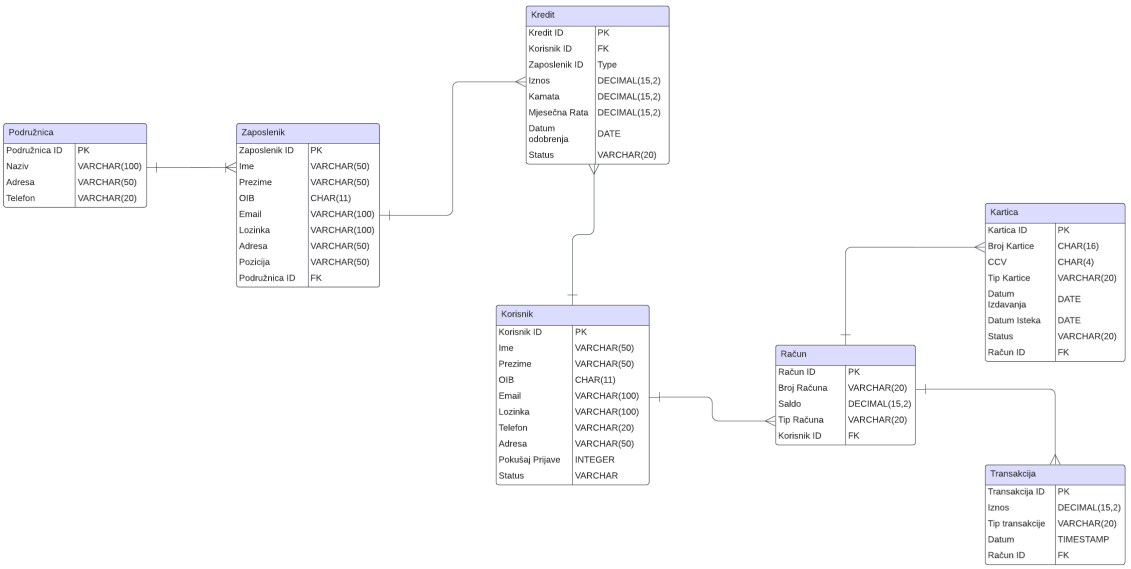
Sl. 5.1 Chenov dijagram za banku

Nakon izrade konceptualne sheme pristupa se **logičkom modeliranju**. Time se u potpunosti, pomoću atributa, opisuju svi postojeći entiteti te potrebe za podacima. Logički podatkovni model tek je proširenje konceptualnog podatkovnog modela, a koriste ga administratori i arhitekti baza podataka pri razvoju fizičkog podatkovnog modela (Kovačević, 2022).

Za početak se definiraju atributi za svaki entitet koji je naveden u konceptualnoj shemi:

- **Atributi** su:
  - KORISNIK ima atribute IME, PREZIME, OIB, EMAIL, LOZINKA, TELEFON, ADRESA, POKUŠAJ PRIJAVE (koliko puta se pogrešnom lozinkom pokušao prijaviti), STATUS (o aktivnosti računa na aplikaciji)
  - ZAPOSLENIK ima atribute IME, PREZIME, OIB, EMAIL, LOZINKA, ADRESA, POZICIJA (radno mjesto), PODRUŽNICA (u kojoj radi)
  - RAČUN ima atribute BROJ RAČUNA, SALDO, TIP RAČUNA, KORISNIK RAČUNA
  - KARTICA ima atribute BROJ KARTICE, CCV, TIP KARTICE, DATUM IZDAVANJA, DATUM ISTEKA, STATUS (je li kartica aktivna), RAČUN (kojem pripada)
  - TRANSAKCIJA ima atribute IZNOS, TIP TRANSAKCIJE, DATUM, RAČUN (s kojega ili na koji je izvršena transakcija)
  - PODRUŽNICA ima atribute NAZIV, ADRESA, TELEFON
  - KREDIT ima atribute KORISNIK (koji otplaćuje kredit), ZAPOSLENIK (koji je odobrio kredit), IZNOS, KAMATA, MJESEČNA RATA, DATUM ODOBRENJA, STATUS (je li kredit otplaćen)

Osim atributa, potrebno je definirati primarne i strane ključeve (kao što je prikazano u tablici (Tablica 5.1)). Na logičkoj razini entiteti, atributi i ključevi za banku biti će prikazani pomoću alata *Lucidchart* (Sl. 5.2). Izrada ove razine još ne ovisi o nikakvom *DBMS-u*. Entiteti imaju svoja imena i navedene sve atribute te njihove tipove podataka. Također, definirani su primarni ključevi koji će se kasnije koristiti za identifikaciju, te strani ključevi koji zapravo stvaraju vezu među entitetima. Prikazane su i veze (svaka veza je jedan naprama više).



Sl. 5.2 Logičko modeliranje baze podataka u alatu *Lucidchart*

Zadnja faza dizajna baze podataka predstavlja kreiranje **fizičkog modela** koji se implementira u relacijskoj bazi podataka. Entiteti iz logičkog modela zamjenjuju se tablicama, a njihovi atributi stupcima koji su sada, ovisno o odabranom načinu implementacije i sustavu, opisani konkretnim, a ne generičkim tipovima podataka (Kovačević, 2022).

Veze među tablicama određuju se na osnovi veza među entitetima u prethodna dva modela. Fizički model predstavlja konkretan način pohrane podataka na disku, koristeći relacijsku bazu podataka. Pretvara logički model u konkretne *SQL* naredbe koje se izvršavaju unutar odabranog sustava za upravljanje bazama podataka. Za potrebe ovog projekta odabran je *SQLite* (integriran pomoću Pythona koristeći sqlite3 biblioteku) zbog svoje jednostavnosti i mogućnosti lokalne implementacije.

Cilj fizičkog modela jest definirati kako će se podaci pohranjivati na disku, a to uključuje definiranje specifičnih tipova podataka, pravila normalizacije, primarne i strane ključeve te ostale parametre koji osiguravaju integritet podataka.

Za početak kreira se funkcija koja stvara vezu s bazom podataka *'MojaBanka.db'* (ako ta baza ne postoji stvorit će novu). U funkciji *stvori_vezu()* nalazi se i opcija *PRAGMA foreign_keys = ON* koja će koristit kasnije za provjeru postojanja stranih ključeva.

```
import sqlite3

def stvori_vezu():
    conn = sqlite3.connect('MojaBanka.db')
    cursor = conn.cursor()
    cursor.execute('PRAGMA foreign_keys = ON;')
    return conn
```

Nakon stvorene veze kreiraju se tablice u bazi po već spomenutom modelu. U funkciji *kreiraj_tablice()* prvo se poziva funkcija za stvaranje veze sa bazom, inicijalizira se *cursor* pomoću kojeg se obavljaju sve funkcije nad bazom i pomoću metode *cursor.execute()* te *CREATE TABLE* opcije dodaju se tablice u bazu. Na kraju potvrđujemo sve promjene napravljene u bazi i zatvaramo konekciju.

```
def kreiraj_tablice():
      conn = stvori_vezu()
      cursor = conn.cursor()
      cursor.execute(' ' '
      CREATE TABLE IF NOT EXISTS korisnici (
            korisnik_id INTEGER PRIMARY KEY AUTOINCREMENT,
            ime VARCHAR(50) NOT NULL,
            prezime VARCHAR(50) NOT NULL,
            oib CHAR(11) UNIQUE NOT NULL,
            email VARCHAR(100) UNIQUE NOT NULL,
            lozinka VARCHAR(100) NOT NULL,
            telefon VARCHAR(20),
            adresa VARCHAR(50),
            pokusaj_prijave INTEGER DEFAULT 0,
            status VARCHAR(20) DEFAULT 'aktivan')
            ' ' ')
      cursor.execute('''
      CREATE TABLE IF NOT EXISTS racuni (
            racun_id INTEGER PRIMARY KEY AUTOINCREMENT,
            broj_racuna VARCHAR(20) UNIQUE NOT NULL,
            saldo DECIMAL(15,2) NOT NULL,
            tip_racuna VARCHAR(20) NOT NULL,
```

```
        korisnik_id INTEGER, FOREIGN KEY (korisnik_id) REFERENCES
        korisnici(korisnik_id))
        ''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS kartice (
        kartica_id INTEGER PRIMARY KEY AUTOINCREMENT,
        broj_kartice CHAR(16) UNIQUE NOT NULL,
        ccv CHAR(4) NOT NULL,
        tip_kartice VARCHAR(20) NOT NULL,
        datum_izdavanja DATE NOT NULL,
        datum_isteka DATE NOT NULL,
        status VARCHAR(20) DEFAULT 'aktivna',
        racun_id INTEGER, FOREIGN KEY (racun_id) REFERENCES
        racuni(racun_id))
        ''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS transakcije (
        transakcija_id INTEGER PRIMARY KEY AUTOINCREMENT,
        iznos DECIMAL(15,2) NOT NULL,
        tip_transakcije VARCHAR(20) NOT NULL,
        datum TIMESTAMP NOT NULL,
        racun_id INTEGER, FOREIGN KEY (racun_id) REFERENCES
        racuni(racun_id))
        ''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS podruznice (
        podruznica_id INTEGER PRIMARY KEY AUTOINCREMENT,
        naziv VARCHAR(100) NOT NULL,
        adresa VARCHAR(50) NOT NULL,
        telefon VARCHAR(20) NOT NULL)
        ''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS zaposlenici (
        zaposlenik_id INTEGER PRIMARY KEY AUTOINCREMENT,
        ime VARCHAR(50) NOT NULL,
```

```
        prezime VARCHAR(50) NOT NULL,
        oib CHAR(11) UNIQUE NOT NULL,
        email VARCHAR(100) UNIQUE NOT NULL,
        lozinka VARCHAR(100) NOT NULL,
        adresa VARCHAR(50),
        pozicija VARCHAR(50),
        podruznica_id INTEGER, FOREIGN KEY (podruznica_id) REFERENCES
        podruznice(podruznica_id))
        ''')
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS krediti (
        kredit_id INTEGER PRIMARY KEY AUTOINCREMENT,
        iznos DECIMAL(15,2) NOT NULL,
        kamata DECIMAL(5,2) NOT NULL,
        mjesecna_rata DECIMAL(15,2) NOT NULL,
        datum_odobrenja DATE NOT NULL,
        status VARCHAR(20) DEFAULT 'aktivan',
        korisnik_id INTEGER, FOREIGN KEY (korisnik_id) REFERENCES
        korisnici(korisnik_id),
        zaposlenik_id INTEGER, FOREIGN KEY (zaposlenik_id) REFERENCES
        zaposlenici(zaposlenik_id))
        ''')
    conn.commit()
    conn.close()
```

Odabrani tipovi podataka usklađeni su s karakteristikama koje najbolje odgovaraju svrsi svakog atributa. Primjerice, za broj računa se koristi tip *VARCHAR(20)* kako bi se omogućilo pohranjivanje različitih formata brojeva računa koji imaju maksimalno 20 znakova. Za polje saldo koristi se *DECIMAL(15,2)* kako bi se precizno pohranili iznosi novčanih transakcija. Slično tome, za datum odobrenja kredita koristi se *DATE*, dok se za vrijeme izvršenja transakcija koristi tip *TIMESTAMP*.

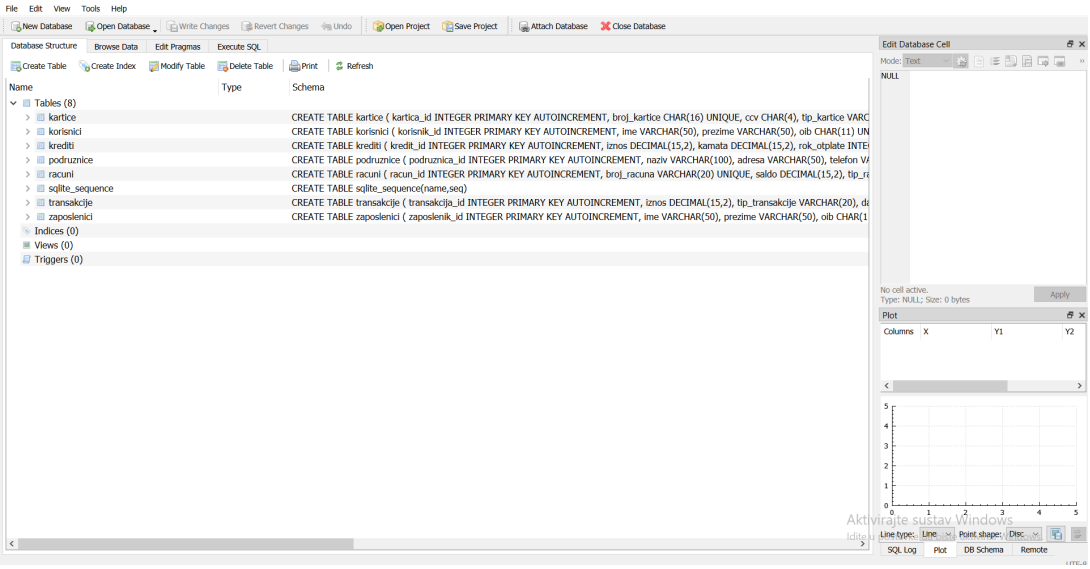
Svaka tablica ima svoj primarni ključ koji osigurava jedinstvenu identifikaciju svakog zapisa. Primarni ključ najčešće je stupac koji koristi *AUTOINCREMENT* opciju, omogućujući automatsko generiranje jedinstvenih vrijednosti. Strani ključevi povezani su s primarnim ključevima iz drugih tablica kako bi se održala konzistentnost i referencijalni

integritet podataka. Na primjer, atribut *korisnik_id* u tablici *racuni* referencira *korisnik_id* iz tablice *korisnici* kako bi se definirala veza između korisnika i računa.

Može se vidjeti i *UNIQUE* ograničenje (engl. *constraint*) koje osigurava da vrijednosti u određenom stupci budu jedinstvene, što znači da nijedan redak ne može imati istu vrijednost u tom stupcu (koristi se za oib, email, broj računa i broj kartice). Za razliku od *PK-a*, *UNIQUE* omogućuje jedinstvenost na više stupaca i može imati *NULL* vrijednost. Dodano je i *NOT NULL* ograničenje koje osigurava da vrijednost stupca ne smije biti prazna. Kroz ovaj projekt koristi se za sve atribute koji su ključni za funkcionalnost baze i ne bi smjeli biti prazni.

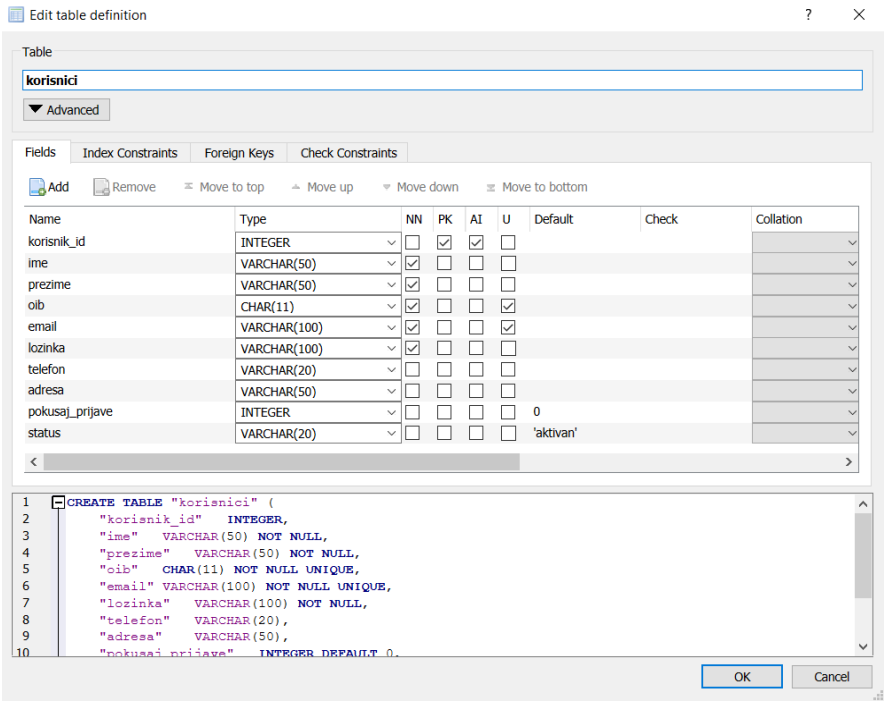
Također korištene su i zadane vrijednosti (engl. *default*) za pojedine atribute. One omogućuju da stupac dobije zadanu vrijednost ako se ne specificira vrijednost prilikom unosa novog retka u tablicu. Na taj način, ako korisnik ne unese podatke za određeno polje, baza će automatski postaviti unaprijed definiranu vrijednost za to polje.

Nakon implementiranja *SQL* skripte za kreiranje tablica, fizički model baze podataka dodan je u *SQLite* sustav što se može provjeriti kroz *DB Browser for SQLite* aplikaciju (Sl. 5.3). Fizičko modeliranje predstavlja zadnju fazu u procesu projektiranja baze podataka. Na ovoj razini dolazi do konkretne implementacije modela unutar odabranog *DBMS-a* te se omogućuje pristup podacima putem *SQL* upita i operacija.



Sl. 5.3 Prikaz tablica u aplikaciji *DB Browser for SQLite* nakon kreiranja

Desnim klikom na bilo koju tablicu i odabirom opcije *Modify table* dostupan je pregled svih stupaca odabrane tablice, njihovih tipova podataka i dodatnih informacija koji se također tu mogu manualno mijenjati. Dostupan je i uvid u primarne i strane, ključeve, ograničenja, zadane vrijednosti i sam kod kojim je tablica kreirana (Sl. 5.4).



Sl. 5.4 Opcija *Modify table* za tablicu *korisnici*

5.3. Unos podataka u bazu

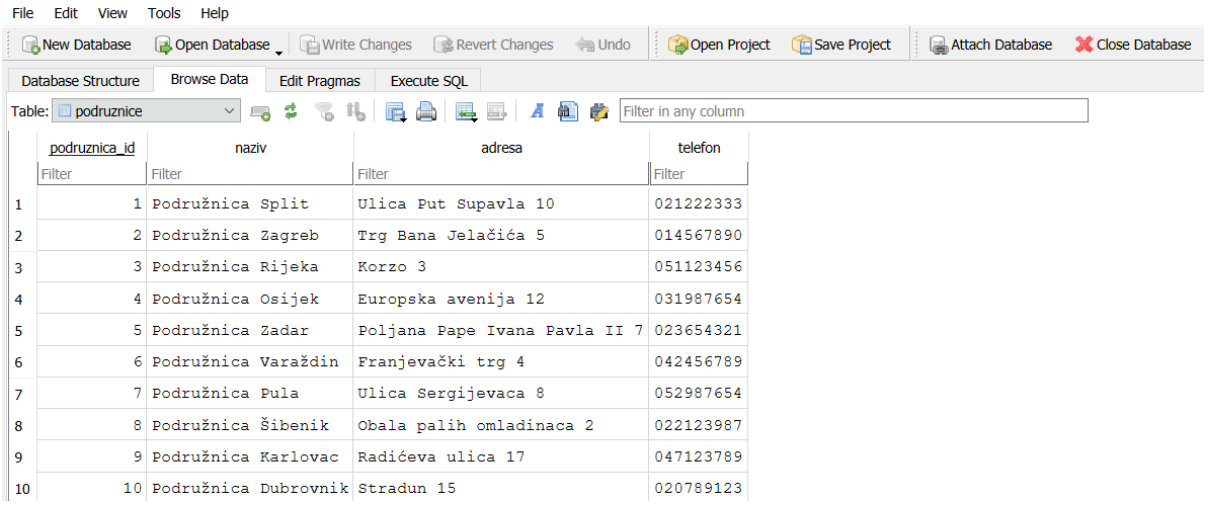
U ovoj fazi razvoja aplikacije, nakon definiranja strukture cijele baze podataka, slijedi unos podataka u bazu. Unos podataka uključuje osnovne informacije o korisnicima, zaposlenicima, računima, karticama, transakcijama, kreditima i podružnicama. No, s obzirom na osjetljivost određenih podataka, kao što su lozinke i informacije o karticama, potrebno je primijeniti tehnike sigurnosne zaštite poput *hashiranja* i kriptiranja kako bi se podaci zaštitili od neovlaštenog pristupa. Također, prilikom unosa podataka u bazu potrebno je implementirati i dodatne provjere valjanosti podataka kako bi se osiguralo da su uneseni podaci ispravni i u skladu s određenim pravilima.

Za početak unose se sve **podružnice**. Unosi se naziv, adresa i telefon. Iako su u bazi već implementirana neka ograničenja, ona se rade i na razini aplikacije kako bi postojala veća kontrola nad aplikacijom i prilagodljive poruke grešaka. Od provjera potrebno je implementirati da telefon sadrži samo znamenke te da niti jedno od polja ne smije biti

prazno. Nakon uvođenja provjera aplikacija se povezuje sa bazom te unutar metode *unesi_podruznicu()* uz opciju *INSERT INTO* unosimo podatke u bazu.

```
def provjeri_telefon(telefon):
    return telefon.isdigit()
def unesi_podruznicu(naziv, adresa, telefon):
    if not naziv or not adresa or not telefon:
        print(f"Greška: Sva polja moraju biti ispunjena.")
        return False
    if not provjeri_telefon(telefon):
        print("Broj telefona nije ispravan!")
        return False
    try:
        conn = stvori_vezu()
        cursor = conn.cursor()
        cursor.execute('''
        INSERT INTO podruznice (naziv, adresa, telefon)
        VALUES (?, ?, ?)
        ''', (naziv, adresa, telefon))
        conn.commit()
        print(f"Podružnica '{naziv}' uspješno unesena.")
    except sqlite3.Error as e:
        print(f"Pogreška prilikom unosa podružnice: {e}")
    finally:
        conn.close()
unesi_podruznicu("Podružnica Zagreb", "Trg Bana Jelačića 5", "014567890")
```

Nakon dodavanja više podružnica u bazu (kako bismo u konačnici imali veće mogućnosti potrebno je dodati više podataka), u *DB Browseru* možemo provjeriti uspješnost unosa podataka klikom na opciju *Browse data* (Sl. 5.5). Također, podaci se mogu filtrirati, prikazivati u kojem poretku korisnik baze želi i slično.



	podruznica_id	naziv	adresa	telefon
	Filter	Filter	Filter	Filter
1	1	Podružnica Split	Ulica Put Supavla 10	021222333
2	2	Podružnica Zagreb	Trg Bana Jelačića 5	014567890
3	3	Podružnica Rijeka	Korzo 3	051123456
4	4	Podružnica Osijek	Europska avenija 12	031987654
5	5	Podružnica Zadar	Poljana Pape Ivana Pavla II 7	023654321
6	6	Podružnica Varaždin	Franjevački trg 4	042456789
7	7	Podružnica Pula	Ulica Sergijevaca 8	052987654
8	8	Podružnica Šibenik	Obala palih omladinaca 2	022123987
9	9	Podružnica Karlovac	Radićeva ulica 17	047123789
10	10	Podružnica Dubrovnik	Stradun 15	020789123

Sl. 5.5 Pregled podataka tablice *podružnice*

Nakon podružnica slijedi unos **zaposlenika**. Za zaposlenike je potrebno unijeti ime, prezime, oib, email, lozinku, adresu, poziciju i podružnicu u kojoj rade. Prije unosa potrebno je napraviti sljedeće provjere:

- određena polja ne smiju biti prazna

```
def provjeri_prazna_polja(ime, prezime, oib, email, lozinka, podruznica_id):
    if not ime or not prezime or not oib or not email or not lozinka or not podruznica_id:
        return False
    return True
```

- oib mora imati 11 znamenki

```
def provjeri_oib(oib):
    if len(oib) != 11 or not oib.isdigit():
    return False
return True
```

- email mora biti ispravnog formata

```
def provjeri_email(email):
    email_regex =
    r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if not re.match(email_regex, email):
        return False
    return True
```

- lozinka mora biti dovoljno sigurna (imati više od 8 znakova, mala i velika slova te jedan poseban znak)

```
def provjeri_lozinku(lozinka):
      if len(lozinka) < 8:
            return False
has_digit = any(char.isdigit() for char in lozinka)
has_upper = any(char.isupper() for char in lozinka)
has_special = any(char in "!@#$%^&*()_+" for char in lozinka)
      return has_digit and has_upper and has_special
```

- pozicija zaposlenika mora biti jedna od postojećih pozicija banke koje su navedene u listi i provjeravaju se

```
dopuštene_pozicije = ['Manadžer', 'Blagajnik', 'Financijski
savjetnik', 'Administrator', 'IT Podrška']
```

Sljedeće jako važno svojstvo prije spremanja zaposlenika u bazu jest ***hashiranje* lozinke**. Koristi se već spomenuta *Python* biblioteka koja omogućava sigurno *hashiranje* lozinki uz dodatak *Salt* koji osigurava da svaka *hashirana* lozinka bude jedinstvena, čak i ako su lozinke identične.

```
import bcrypt
def hashiraj_lozinku(lozinka):
      salt = bcrypt.gensalt()
      hashed_lozinka = bcrypt.hashpw(lozinka.encode('utf-8'), salt) return
      hashed_lozinka
```

Na kraju potrebno je implementirati jako važno svojstvo sigurnosti baze podataka, a to je **kriptiranje osjetljivih podataka**. Što se tiče podataka o zaposlenicima kriptirat će se oib i adresa zbog moguće povrede identiteta zaposlenika u slučaju neovlaštenog pristupa bazi. Radi se o selektivnom kriptiranju baze, budući da nije potrebna kriptografija cijele baze niti tablice. Za kriptiranje koristi se *Fernet* kriptografski sustav iz *Python* biblioteke *cryptography*. To je simetrični sustav kriptiranja, što znači da se isti ključ koristi za šifriranje i dešifriranje podataka. Prvi korak je generiranje ključa koji mora biti sigurno pohranjen i zaštićen.

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
with open('secret.key', 'wb') as key_file:
    key_file.write(key)
```

Nakon toga, pri svakom kriptiranju podatka potrebno je učitati već generirani ključ te pomoću funkcije *encrypt()* šifrirati podatke koji su prethodno pretvoreni u bajtove.

```
with open('secret.key', 'rb') as key_file:
    key = key_file.read()
cipher = Fernet(key)
kriptirani_oib = cipher.encrypt(oib.encode())
kriptirana_adresa = cipher.encrypt(adresa.encode())
```

Sada je moguće dodavanje zaposlenika u bazu. Nakon dodavanja, u bazi možemo vidjeti način na koji su prikazane *hashirane* lozinke (Sl. 5.6) te oib kao jedan od kriptiranih stupaca ove tablice (Sl. 5.7).

```
def dodaj_zaposlenika(ime, prezime, oib, email, lozinka, adresa, pozicija, podruznica_id):
      if not provjeri_prazna_polja(ime, prezime, oib, email, lozinka, adresa, pozicija, podruznica_id):
       print(f"Greška: Sva polja moraju biti ispunjena.")
            return
       if not provjeri_email(email):
            print(f"Greška: E-mail adresa '{email}' nije ispravna.")
            return
       if not provjeri_lozinku(lozinka):
            print(f"Greška: Lozinka mora imati minimalno 8 znakova i sadržavati barem jedno veliko slovo, broj i poseban znak.")
            return
    if not provjeri_oib(oib):
            print(f"Greška: OIB '{oib}' nije ispravan. ")
            return
    if pozicija not in dopuštene_pozicije:
            print(f"Greška: Pozicija '{pozicija}' nije dozvoljena. Dozvoljene pozicije su: {', '.join(dopuštene_pozicije)}.")
            return
    conn = stvori_vezu()
    cursor = conn.cursor()
    hashed_password = hashiraj_lozinku(lozinka)
    kriptirani_oib = cipher.encrypt(oib.encode())
```

```
    kriptirana_adresa = cipher.encrypt(adresa.encode())
    cursor.execute('''
      INSERT INTO zaposlenici (ime, prezime, oib, email, lozinka, adresa, pozicija, podruznica_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?)
''', (ime, prezime, oib, email, hashed_password, adresa, pozicija, podruznica_id))
    conn.commit()
    conn.close()
    print(f"Zaposlenik {ime} {prezime} uspješno dodan.")
dodaj_zaposlenika("Ivan", "Matić", "12345678901", "ivan.matic@gmail.com", "LozinkaIvan1!", "Istarska 14", "Menadžer", 1)
```

lozinka
Filter
$2b$12$hHn2JSYLKooOPI4K2Zpeie9nebf/t0P29wW4FVjWCThBYnrf9t5vK
$2b$12$VG6.SmKGShDh2o4zNDfKH.QTc2OCCjxRQzCVRqNIj45z4dBdpHf9G
$2b$12$zuZTM5/s.FGCfM2b2FFZEOEuXBvkyV9N.e02u/qXMRXMjL0hk6MEK
$2b$12$0eu6/DJ7XIZBcCnB8KypwOWZl24Ioux29h/JjmzHI/6IlehTKI.DW
$2b$12$oR/tnkjj7D70J/SfUsfOSO9iQRvaKi2n9PkGowI9W1uUesGTl8Zpq
$2b$12$a7cOHNVWryi9EaXDZQaDCOTy0HoIaGI0n8sMW5WRWv4biwa7fMLE.
$2b$12$7Golkeo8QLp3AX//Q2JMyu13J9w.IEhf555JdALp8I/y4VEtQGb2i
$2b$12$SmkexNXD4Ldxvrez0MRw1.oPZjFF.RZADnrRlBS8THEqR8.Zz4Ro.
$2b$12$QDclmMXh20JSccl0g15Ivulu.JWG7qIbKBgCoByGXwcEYUThQYwGy
$2b$12$cG7XJ7GCHG/PlOdlcGQMiu9KF/GRsS/696cqh7D3VUWS2VpCaRlQG

Sl. 5.6 Stupac sa *hashiranim* lozinkama zaposlenika

oib
Filter
gAAAAABm7w7p1ltU9JLqkSYeCf6Td2d7eaLxwZz_JhCD2yXXdByR21m6t_1nLT4fK3exP0DOT4fQKIAyQpIbldS88YTCwwllWg==
gAAAAABm7w7pphC5w7p5d4ew7HyhcFX7fQYuxXJgP4wAzZdHI5kH3wk78M5eU_2XTUEBSCKRiLU5mHk0hUtdnj9iXehWDQODAQ==
gAAAAABm7w7qVrzUYOnxEgEF49a8qW03i2dGuCGGJrJC9gR1JeJizXkpG6armCJcyu2ffiSOAIFyhWOOMZ-J7Y1HqQ-zF2smGw==
gAAAAABm7w7qppEPrG7Av_DJ4_BeYimN0LSL7jwgY7i3obZvcD3V3qJG-I91M2aTwlcl5xMpcaAYjv1ah5Iy472l9hY9XsGWNg==
gAAAAABm7w7qMV6lcMbG1uUR-HXQV4ZPb6ZGHAEXlHYEpTdH2YYjfcJxL6OSftFQwlWMWvA_DTG5NwRqMvkRo1U1oSViHSqB4Q==
gAAAAABm7w7rIpZi4n6_YuTfTF-7oANn-THVW-kGd2_2NsY_PtjNXxkuQA_XpuUBMxwSPMiFLuIR9fQ-kxvneB14cQOoWe2Bcg==
gAAAAABm7w7rD6L8_9eBIbsJtuulShf74QCUgGqcEXx4anwSgLzuR_CyuUEA6Z69ibui4MJSOWRyCnqV5wE8IdVVG0KeKNp7Vg==
gAAAAABm7w7r5XNywrAzA0kNf86rTHP_U_ecMicBh8Sg2bMjGmb7INzseDv710_vcNtN5FS8f1dC30VuvuzuV9-3vspm3SkR7A==
gAAAAABm7w7sOkJkOf5rDWDjY0HK23j1AKrC7OL9wsrkJ2Q7Q65kaT-kTZ3Mi_jVd-yilXaaMxVfVCIf_GktN9Exgv1efrsMFA==
gAAAAABm7w7s-tVj-FoWo2uGh8iBp1UN-lSnw39PaEekeuRrnZ2AGS39dNdvtUeefmyY-VReY39LEsj3Ay0WHaBfsWJUrmzwrA==

Sl. 5.7 Stupac sa kriptiranim oib-om zaposlenika

Na vrlo sličan način unose se i podaci o **korisnicima** u bazu. Sadrže iste podatke kao i zaposlenici: ime, prezime, oib, email, lozinku, telefon i adresu te su samim time provjere unosa podataka identične. Osim njih, tablica korisnici sadrži stupce pokušaj prijave i status njihovog profila. Pokušaj prijave ima zadanu vrijednost 0 koja se povećava svaki puta kada korisnik pogrešno unese lozinku, što će detaljno biti objašnjeno u izradi desktop aplikacije. Status profila može biti *'aktivan'*, što je također zadana vrijednost, te *'blokiran'* kada korisnik dostigne određen broj pogrešnih prijava (što je implementirano kao sigurnosna mjera pri izradi aplikacije).

Nakon toga dodaju se **krediti** koji sadrže sljedeće podatke: iznos kredita, kamata, mjesečna rata, datum odobrenja, status je li kredit aktivan ili otplaćen te id korisnika i id zaposlenika koji je odobrio kredit. Prije unosa podataka u ovu tablicu potrebno je samo provjeriti da je kredit odobrio samo onaj zaposlenik čija je pozicija *'Financijski savjetnik'*.

```
def provjeri_financijskog_savjetnika(zaposlenik_id):
    conn = stvori_vezu()
    cursor = conn.cursor()
    cursor.execute('
      SELECT pozicija FROM zaposlenici WHERE zaposlenik_id = ?',
      (zaposlenik_id,))
    zaposlenik = cursor.fetchone()
    conn.close()
    if zaposlenik[0] != 'Financijski savjetnik':
            print(f"Greška: Zaposlenik s ID-om {zaposlenik_id} nije
            'Financijski savjetnik'.")
            return False
    return True
```

Za ovu tablicu nisu potrebne dodatne provjere niti šifriranje bilo kakvih podataka. Sljedeće, dodaju se bankovni **računi** korisnika koji sadrže: broj računa, saldo, tip računa te id korisnika koji je vlasnik toga računa. Implementirane su sljedeće provjere:

- broj računa počinje sa *'HR'* i uz to ima jos 18 znamenki, broj računa biti će kriptirani podatak u tablici

```
def generiraj_broj_racuna():
    broj_racuna='HR' + ''.join(random.choices(string.digits, k=18))
    kriptirani_broj_racuna=cipher.encrypt(broj_racuna.encode('utf-
    8'))
    return kriptirani_broj_racuna
```

- tip računa mora biti jedan od ponuđenih iz liste

```
def validiraj_tip_racuna(tip_racuna):
    dozvoljeni_tipovi=['Ziro','Tekuci','Stedni',
    'Investicijski', 'Devizni']
    return tip_racuna in dozvoljeni_tipovi
```

Pretposljednja tablica u koju dodajemo podatke jest **kartice**. Podatke koje sadrži su: broj kartice, ccv, datum izdavanja, datum isteka te status je li kartica aktivna ili blokirana. Unaprijed određena vrijednost za status kartice jest *'aktivna'*.

- broj kartice mora imati 18 znamenki koje će biti kriptirane

```
def generiraj_broj_kartice():
    broj_kartice = ''.join(random.choices(string.digits, k=16))
    kriptirani_broj_kartice=cipher.encrypt(broj_kartice.encode('utf
    -8'))
    return kriptirani_broj_kartice
```

- ccv ima 3 ili 4 znamenke koje su također kriptirane

```
def generiraj_ccv():
    ccv = ''.join(random.choices(string.digits, k=random.choice([3,
    4])))
    kriptirani_ccv=cipher.encrypt(ccv.encode('utf-8'))
    return kriptirani_ccv
```

Posljednje na redu su **transakcije**. Tablica sa transakcijama sadrži sljedeće stupce čije podatke je potrebno unijeti: iznos, tip transakcije, datum kada je transakcija izvršena te id računa sa kojeg ili na koji se transakcija izvršila. Iznos ne smije biti manji ili jednak 0 te tip transakcije mora biti jedan od ponuđenih iz liste. Oboje je implementirano kroz samu funkciju dodavanja transakcije u bazu.

```
def dodaj_transakciju(racun_id, iznos, tip_transakcije, datum):
    conn = sqlite3.connect('MojaBanka.db')
    cursor = conn.cursor()
    if tip_transakcije not in ['Uplata', 'Isplata', 'Prijenos']:
        print("Neispravan tip transakcije. Mora biti Uplata, Isplata ili
        Prijenos.")
        return False
    if iznos <= 0:
        print("Iznos ne smije biti 0 niti negativan.")
        return False
    cursor.execute('''
      INSERT INTO transakcije (iznos, tip_transakcije, datum_transakcije,
      racun_id)
      VALUES (?, ?, ?, ?)''', (iznos, tip_transakcije, datum, racun_id))
    conn.commit()
    conn.close()
```

Sada je u bazi moguće vidjeti i tablicu *sql_sequence* koja za svaku tablicu prikazuje trenutno najveću vrijednost primarnog ključa generiranog u toj tablici (Sl. 5.8).

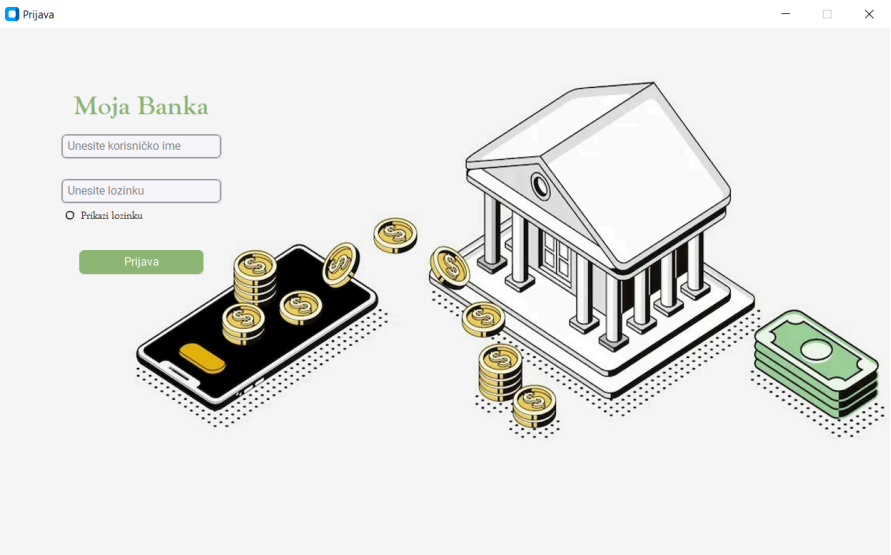
Table: sqlite_sequence

	name	seq
	Filter	Filter
1	podruznice	10
2	zaposlenici	10
3	korisnici	10
4	krediti	15
5	racuni	20
6	kartice	9
7	transakcije	6

Sl. 5.8 Tablica *sql_sequence*

5.4. Izrada desktop aplikacije

U ovome dijelu rada detaljno je opisana izrada desktop aplikacije za bankovni sustav. Aplikacija je implementirana koristeći već spomenute *Python* biblioteke *Tkinter* i *Custom Tkinter* za izradu grafičkog sučelja. Za početak, fokus će biti na sigurnosti prilikom prijave korisnika i zaposlenika kako bi se osigurao siguran pristup osjetljivim podacima pohranjenima u bazi. Izgled prozora za prijavu prikazan je slikom (Sl. 5.9).



Sl. 5.9 Grafičko sučelje za prijavu korisnika i zaposlenika banke

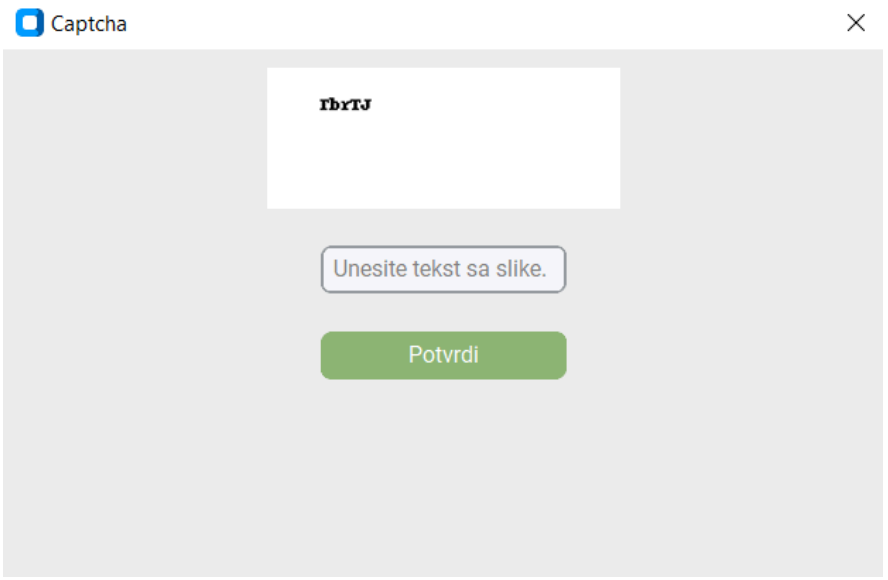
- **Prijava korisnika**

Korisnici sustava imaju standardnu proceduru prijave uz pomoć email-a i lozinke. Potrebno je unijeti oboje, u protivnom se ispisuje poruka greške. Također, ako korisnik tri puta krivo unese lozinku blokira se njegov račun zbog sigurnosti. Za ponovno otključavanje profila potrebno je unijeti ručno status njegovog profila u samoj bazi podataka od strane zaposlenika banke. Provjera točnosti lozinke obavlja se usporedbom unesene lozinke s *hashiranom* lozinkom pohranjenom u bazi podataka. Umjesto jednostavne usporedbe unesene lozinke s pohranjenim *hashom*, koristi se *bcrypt* koji *hashira* unesenu lozinku koristeći istu *"salt"* vrijednost koja je korištena prilikom prvotnog *hashiranja* pohranjene lozinke.

```
def provjeri_lozinku(unos_lozinke, hash_lozinka):
    return bcrypt.checkpw(unos_lozinke.encode('utf-8'), hash_lozinka)
```

Uz to, dodana je i dodatna sigurnosna mjera *CAPTCHA* zaštite kako bi se osiguralo da je korisnik koji pristupa sustavu stvarna osoba, a ne automatizirani softver ili *bot*. Generira se nasumično 5 slova koje korisnik mora unijeti (Sl. 5.10).

```
def generiraj_captcha():
    slova = string.ascii_letters + string.digits
    return ''.join(random.choices(slova, k=5))
def kreiraj_captcha_sliku(tekst):
    img = Image.new('RGB', (200, 80), color=(255, 255, 255))
    font = ImageFont.load_default()
    draw = ImageDraw.Draw(img)
    draw.text((30, 15), tekst, font=font, fill=(0, 0, 0))
    return img
captcha_tekst = ""
def nova_captcha(captcha_label):
    global captcha_tekst, captcha_img
    captcha_tekst = generiraj_captcha()
    captcha_slika = kreiraj_captcha_sliku(captcha_tekst)
    captcha_img = CTkImage(light_image=captcha_slika, size=(200, 80))
    captcha_label.configure(image=captcha_img)
```



Sl. 5.10 *CAPTCHA*

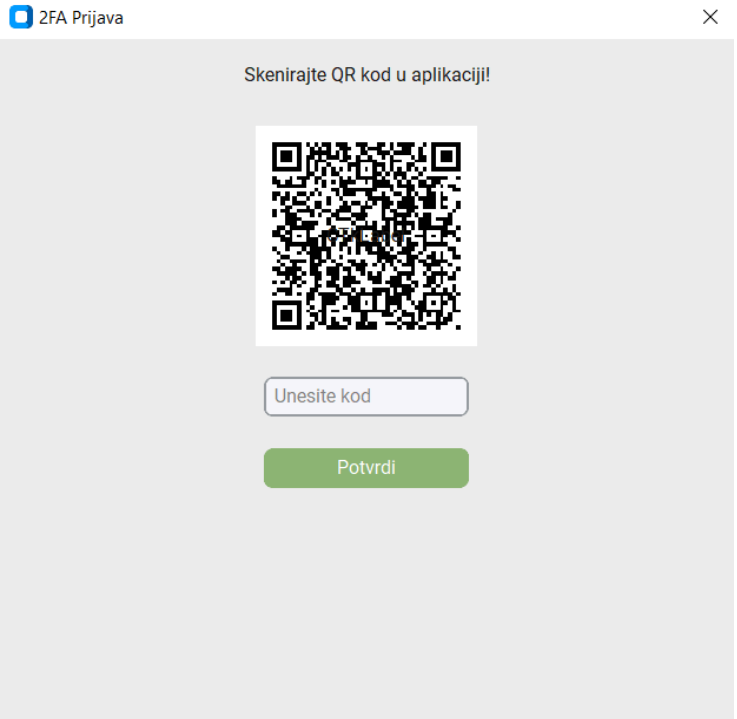
Ako korisnik ispravno unese tekst sa slike, pristup mu je omogućen. Ako unese netočnu *CAPTCHA-u*, generira se nova i korisnik mora ponovno pokušati prijavu. Tek nakon što je svaki od navedenih koraka uspješno unesen, korisniku je omogućen pristup aplikaciji i uvidu u vlastite podatke (kao što su kartice, krediti, računi i transakcije).

- **Prijava zaposlenika**

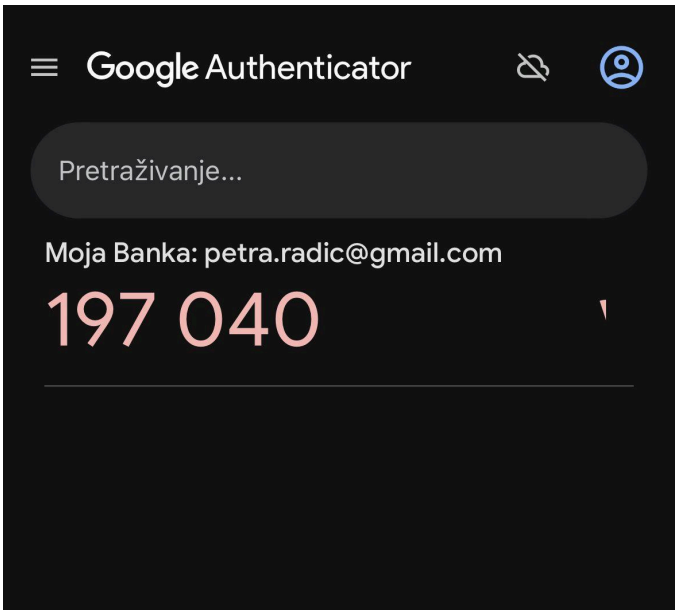
Što se tiče prijave zaposlenika banke u sustav. Osim standardne prijave, koja je ista kao kod korisnika banke, ona uključuje i dodatan sloj sigurnosti u obliku dvofaktorske autentifikacije, što je standardna sigurnosna praksa u modernim aplikacijama. Ono omogućava da prijava nije moguća samo pomoću lozinke, već zaposlenik mora potvrditi svoj identitet unoseći jednokratni kod generiran putem aplikacije za autentifikaciju, poput *Google Authenticator-a*. Funkcionira na način da se generira tajni ključ koji zaposlenik mora dodati u svoju aplikaciju za autentifikaciju prilikom prve prijave (u sklopu ovoga projekta to će biti putem *QR koda*). Aplikacija generira jednokratne kodove koji vrijede samo 30 sekundi. Ako *2FA* kod nije ispravan, prijava se odbija i zaposlenik mora unijeti novi.

```
import pyotp

secret = pyotp.random_base32()

def auth(email):
    def provjeri_2fa():
        uneseni_kod = entry_2fa.get()
        totp = pyotp.TOTP(secret)
        if totp.verify(uneseni_kod):
            messagebox.showinfo("Uspjeh", "Prijava uspješna!")
            auth_window.destroy()
            root.destroy()
            otvori_bankar_page()
        else:
            messagebox.showerror("Greška", "Pogrešan 2FA kod!")
    auth_window = CTkToplevel(root)
```

```
    auth_window.geometry('500x500')
    auth_window.title('2FA Prijava')
    auth_window.transient(root)
    auth_window.grab_set()
    totp = pyotp.TOTP(secret)
    qr_data = totp.provisioning_uri(name=email, issuer_name='Moja Banka')
    qr_img = qrcode.make(qr_data).resize((150, 150))
    qr_img_tk = CTkImage(light_image=qr_img, size=(150, 150))
       qr_label = CTkLabel(auth_window, text="Skenirajte QR kod u
aplikaciji!")
    qr_label.pack(pady=10)
    qr_image_label = CTkLabel(auth_window, image=qr_img_tk)
    qr_image_label.pack(pady=10)
    entry_2fa = CTkEntry(auth_window, placeholder_text="Unesite kod")
    entry_2fa.pack(pady=10)
    potvrdi_button = CTkButton(
        auth_window,
        text="Potvrdi",
        command=provjeri_2fa,
        fg_color='#8FB573',
        text_color='white',
        hover_color='#7A9F63',
        cursor='hand2'
    )
    potvrdi_button.pack(pady=10)
```



Sl. 5.11 Dvofaktorska autentifikacija u aplikaciji



Sl. 5.12 Prikaz jednokratnog koda u *Google Authenticator-u*

Nakon potpuno uspješne prijave, zaposlenika se preusmjerava na dio aplikacije u kojem ima potpuni pristup bazi podataka u kojoj može brisati, izmjenjivati i dodavati podatke.

Zaključak

Rad na temu kriptiranja i sigurnosti baza podataka temeljito je istražio ključne aspekte zaštite podataka unutar modernih sustava. Baze podataka, kao srž mnogih digitalnih sustava, zahtijevaju napredne mjere zaštite kako bi se osigurala povjerljivost, integritet i dostupnost podataka. Kroz analizu različitih sigurnosnih prijetnji, poput *SQL* ubacivanja, napada uskraćivanjem usluge i *brute force* napada, istaknuta je nužnost implementacije sigurnosnih mjera u sustavima za upravljanje bazama podataka.

Kriptiranje predstavlja ključni mehanizam zaštite podataka, omogućujući šifriranje podataka na različitim razinama, od pojedinačnih polja do cjelokupne baze. Primjenom simetričnih i asimetričnih kriptografskih algoritama postiže se visok stupanj sigurnosti, no njihova implementacija zahtijeva pažljivu prilagodbu infrastrukture baze podataka kako bi se održala optimalna učinkovitost sustava.

Praktični dio rada obuhvatio je simulaciju sigurnosnih mehanizama u relacijskoj bazi podataka, čime su potvrđene teorijske postavke. Primijenjeni su mehanizmi *hashiranja* lozinki i kriptiranja osobnih podataka, što je dodatno osiguralo podatke od neovlaštenih pristupa. Osim toga, kroz razvoj desktop aplikacije pomoću *Pythona* i *SQLite-a*, pokazano je kako je moguće stvoriti siguran, skalabilan i učinkovit sustav za upravljanje osjetljivim podacima u bankovnom okruženju. Ovaj rad pruža sveobuhvatan pregled važnosti kriptiranja u zaštiti podataka unutar baza podataka, naglašavajući potrebu za balansiranjem između sigurnosnih zahtjeva i performansi sustava, te demonstrira kako implementacija sigurnosnih mjera može značajno povećati sigurnost i pouzdanost cjelokupnog sustava.

Literatura

[1] Oracle, *What is a database?*, https://www.oracle.com/hr/database/what-is-database/ (pristupljeno 12.9.2024.)

[2] Manger R., Baze podataka, 2011.

[3] Puharić P., Budimlić M., Informacijska sigurnost (europski standardi), 2008.

[4] Panmore Institute, *The CIA Triad: Confidentiality, Integrity, Availability*, https://panmore.com/the-cia-triad-confidentiality-integrity-availability (pristupljeno 18.9.2024.)

[5] Antončić V., Napadi s uskraćivanjem usluge (DoS napadi), http://sigurnost.zemris.fer.hr/ns/2007_Antoncic/, 2007.

[6] CIS, Napadi umetanjem SQL koda, https://www.cis.hr/files/dokumenti/CIS-DOC-2011-09-025.pdf, 2011.

[7] CIS, Zaštita baza podataka, https://www.cis.hr/files/dokumenti/CIS-DOC-2012-08-059.pdf, 2012.

[8] Dujella A., Maretić M., Kriptografija, Element, Zagreb, 2007.

[9] Kurose J., Ross K, Computer Networking: a top-down approach, Pearson, New Jersey, 2010.

[10] Engineering Ebook, *Data Encryption Standard DES, Advantages Disadvantages algorithm*, http://engineeringfourum.blogspot.com/2014/04/data-encryption-standard-des.html (pristupljeno 17.9.2024.)

[11] Geeks for geeks, *Advanced Encryption Standard (AES)*, https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ (pristupljeno 17.9.2024.)

[12] Maleković M., Rabuzin K., Uvod u baze podataka, Fakultet organizacije i informatike, Varaždin, 2016.

[13] Coles M., Landrum R., Expert SQL Server 2008 Encryption, Apress Academic, 2009.

[14] Kovačević Ž., Modeliranje, implementacija i administracija baza podataka, Tehničko veleučilište u Zagrebu, Zagreb, 2022.

Skraćenice

SQL	*Structured Query Language*	strukturirani jezik za upite
DBMS	*Database Management System*	sustav za upravljanje bazom podataka
PK	*Primary Key*	primarni ključ
FK	*Foreign Key*	strani ključ
DDL	*Data Description Language*	jezik za opis podataka
DML	*Data Manipulation Language*	jezik za manipulaciju podacima
QL	*Query Language*	jezik za upite
CIA	*Confidentiality, Integrity and Availability*	povjerljivost, integritet i dostupnost
DoS	*Denial of Service*	uskraćivanje usluga
2FA	*Two-Factor Authentication*	dvofaktorska autentifikacija
MAC	*Mandatory Access Control*	obvezna kontrola pristupa
DAC	*Discretionary Access Control*	diskretna kontrola pristupa
LDAP	*Lightweight Directory Access Protocol*	lagani protokol za pristup direktoriju
OAuth	*Open Authorization*	otvorena autorizacija
CBC	*Cipher Block Chaining*	ulančavanje šifriranih blokova
DES	*Data Encryption Standard*	standardni algoritam za kriptiranje
AES	*Advanced Encryption Standard*	napredni standard šifriranja
RSA	*Rivest-Shamir-Adleman*	
OTP	*One Time Password*	jednokratni kod