|  | **Univerzitet u Nišu, Elektronski fakultet**  **Katedra za računarstvo** |  |
| --- | --- | --- |

**Mihajlo Veljković**

**Skladištenje poverljivih informacija u PostgreSQL DBMS-u**

**Seminarski rad**

**Niš, 2023**

Sadržaj

[Uvod 3](#_Toc135763148)

[Rukovanje poverljivim informacijama 4](#_Toc135763149)

[Ograničavanje pristupa 6](#_Toc135763150)

[RBAC sistem 6](#_Toc135763151)

[RLS sistem 7](#_Toc135763152)

[Enkriptovanje pomoću pgcrypto modula 8](#_Toc135763153)

[Dodatna enkripcija pomoću pgsodium alata 10](#_Toc135763154)

[Anonimizacija podataka 13](#_Toc135763155)

[Zaključak 15](#_Toc135763156)

[Literatura 16](#_Toc135763157)

# Uvod

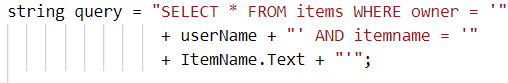
U današnjem svetu, kompanije raznih veličina prikupljaju i čuvaju ogroman broj poverljivih informacija o korisnicima. Ovi podaci mogu biti lični podaci, finansijski podaci, intelektualne svojine i druge poverljive informacije. Potrebno je pronaći adekvatno rešenje za čuvanje i zaštitu ovakvih podataka kako bi sprečili njihovo neovlašćeno korišćenje, modifikaciju ili curenje istih.

PostgreSQL DBMS nudi određena rešenja koja mogu pomoći pri rešavanju ovih problema. Ugrađen sistem kontrole pristupa koji je zasnovan na ulogama (RBAC) omogućava administratoru podešavanje različitih nivoa pristupa nad podacima u zavisnosti od potrebe korisnika. Pored toga, postoji i ugrađena podrška za enkripciju podataka pomoću *pgcrypto* modula.

Međutim, kod pojedinih aplikacija i podataka koji se skladište, ovi moduli se nisu pokazali kao kompletna rešenja. Shodno tome, u ovom radu biće i reči o ekstenzijama *pgsodium* i *postgresql\_anonymizer* koje pružaju viši nivo sigurnosti.

# Rukovanje poverljivim informacijama

Pri projektovanju aplikacija treba uzeti u obzir različite nivoe zaštite i ne vršiti pretpostavke kada su u pitanju poverljive informacije. Klijentu (bilo to delegiran zahtev preko API poziva, ili direktan pristup podacima) nikad ne treba verovati. Najčešći napadi za otkrivanje poverljivih informacija su *SQL Injection* napadi. Ova vrsta napada je izuzetno jednostavna, ali isto tako i laka za odbranu. Pri korišćenju bilo kakvih upita, potrebno je očistiti parametre od sintaksnih znakova SQL jezika ili koristiti upite sa parametrima koji mogu automatski zaštiti potencijalno zlonameran upit od izvršavanja.

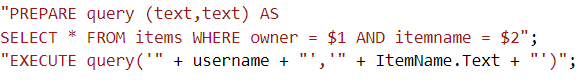


Slika 1.1 Primer konkateniranog upita u C# sa potencijalom za napad

Izvršenje ovog upita može biti promenjeno ako se pošalju zlonamerni parametri. Ako pošaljemo da je *userName*:

Slika 1.2 Zlonamerni parametar

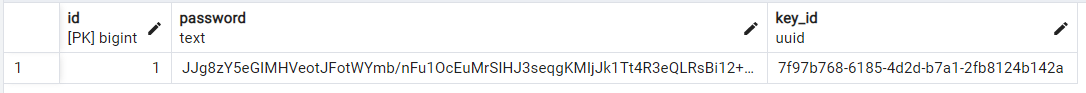
PostgreSQL će evaluirati *owner* varijablu kao „user“, ali pored toga će evaluirati ceo uslov kao tačan zbog dodatog „2=2“. Uz parametre, ovaj upit bi izgledao:



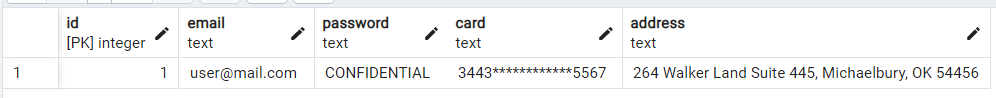
Slika 1.3 Upit sa parametrima

Za ovakav upit baza može bezbedno da prosledi parametre, pa nije moguće izvršiti ovaj tip napada.

Potrebno je i pretpostaviti da postoji mogućnost da neki napadač dobije pristup ili kredencijale koje ne sme da ima i obezbediti dovoljno dobar dizajn baze podataka tako da je pristup poverljivim informacijama onemogućen ili da su te informacije enkriptovane ili anonimizirane na takav način da nisu čitljive. U ranoj eri interneta, najčešći način curenja informacija je bio neovlašćeni pristup korisničkim podacima koji su najčešće bili neenkriptovani, pa je bilo lako direktno pročitati sve podatke, a potom to zloupotrebiti.



Slika 1.4 Enkriptovane informacije



Slika 1.5 Anonimizirane informacije

Kao korisnik, ne treba se oslanjati na bezbednosne mere kompanija, pa je potrebno imati određene predostrožnosti pri korišćenju bilo koje aplikacije:

* ne koristiti istu šifru na više različitih aplikacija
* ako je moguće koristiti različite mejlove (ili alijase mejlova)
* ostavljati samo potrebne informacije
* proveravati validnost sertifikata pri pristupu aplikaciji

# Ograničavanje pristupa

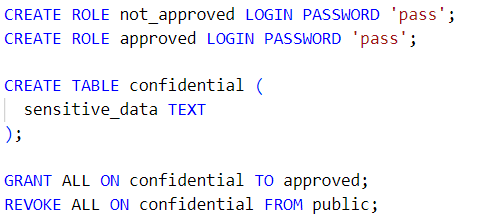
PostgreSQL DBMS poseduje sistem kontrole pristupa zasnovan na ulogama, koji u kombinaciji sa sistemom provere pristupa redovima baza (RLS - *row level security*) može efektivno ograničiti nedozvoljeni pristup podacima.

## RBAC sistem

Svaki objekat (tabela, pogled, šema, funkcija...) u PostgreSQL-u ima vlasnika (najčešće uloga koja je izvršila skriptu za kreiranje). Vlasnik ima sve privilegije nad kreiranim objektom, ali je potrebno eksplicitno dodeliti privilegije pristupa nad objektima.

Moguće privilegije su:

* SELECT (r) – mogućnost čitanja podataka iz kolona, pri čemu je moguće navesti i specifične kolone za pristup
* INSERT (a) – mogućnost upisa podataka, takođe je moguće upis u samo specifičnim kolonama
* UPDATE (w) – mogućnost ažuriranja podataka
* DELETE (d) – mogućnost brisanja podataka
* TRUNCATE (D) – dozvoljava operaciju *TRUNCATE* nad tabelom
* REFERENCES (x) – mogućnost povezivanja stranih ključeva
* TRIGGER (t) – mogućnost povezivanja okidakčih funkcija na tabelu
* CREATE (C) – mogućnost kreiranja objekata unutar baze ili šeme
* CONNECT (c) – dozvoljava konekcije ka bazi
* TEMPORARY (T) – mogućnost kreiranja privremenih tabela
* EXECUTE (X) – dozvoljava korišćenje funkcija i procedura
* USAGE (U) – dozvoljava korišćenje objekata unutar šeme,kreiranih tipova ili sekvenci
* SET (s) – dozvoljava postavljanje varijabli
* ALTER SYSTEM (A) – dozvoljava promenu sistemskih varijabli

Ovakvim sistemom privilegija je moguće dati korisnicima pristup samo neophodnim podacima, ali to nije i dovoljno rešenje. Uzmimo za primer sledeću tabelu:

Slika 2. Primer tabela sa pravilima pristupa

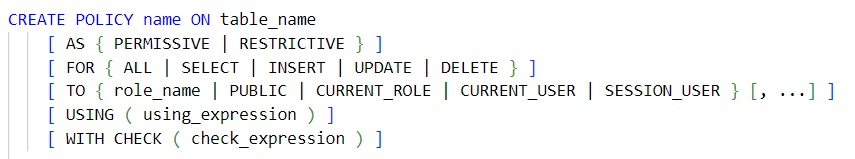
Iako smo ograničili pristup ovoj tabeli korisnicima koji nemaju privilegije, korisnik koji ima privilegije može da čita sve redove unutar te tabele. Potrebno je ograničiti pristup i na nivou reda.

## RLS sistem

Za dodatnu kontrolu pristupa, možemo koristiti RLS sistem kako bi ograničili i kojim redovima može da pristupa neki korisnik. Kako bi uključili ovaj sistem, potrebno je izvršiti upit nad tabelom u kojoj želimo ovaj sistem:

Slika 2. Komanda za uključivanje RLS sistema

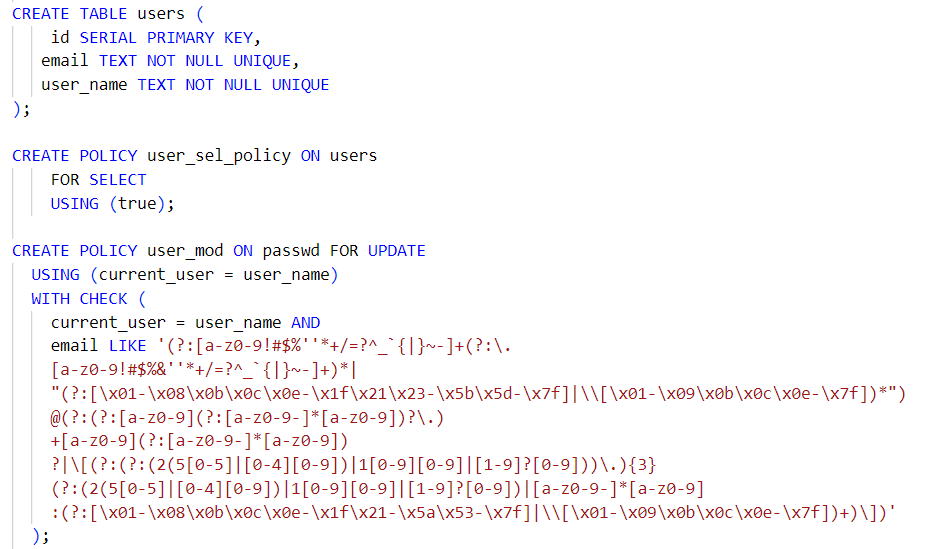
Ovaj upit postavlja i implictno pravilo da nikom nije dozvoljen pristup podacima u tabeli, pa je potrebno napraviti posebna pravila za sve korisnike kojima želimo da dozvolimo korišćenje tabele. Pravilo mora biti takvo da se evaulira kao logički izraz tj. da je rezultat tačno ili netačno. Pravila mogu biti univerzalna (ključna reč *ALL*) ili za specifične upite (*SELECT, INSERT, UPDATE, DELETE*). U slučaju da definišemo pravila koja nisu univerzalna, potrebni su nam izrazi koji se evaluiraju nad postojećim redovima koji se definišu korišćenjem ključne reči *USING*, dok se izrazi za nove redove definišu pomoću ključne reči *WITH CHECK*. Sintaksa pravila je sledeća:



Slika 2. Sintaksa pravila

Pravila ne važe za korisnike koji su *superuser* ili imaju *BYPASSRLS* atribut, a mogu da ih definišu samo vlasnici tabela. Moguće je definisati više od jednog pravila i ona su zasebni objekti, pa je moguće i menjati postojeća pravila (pomoću *ALTER POLICY* komande) ili ih brisati (pomoću *DROP POLICY*).

Uzmimo za primer tabelu korisnika. Želimo da dozvolimo korisnicima čitanje ostalih korisnika, ali modifikovanje samo svojih redova. Takođe, zelimo da validiramo e-mail prilikom upisa podataka. Pravilo *user\_sel\_policy* služi za dozvolu pristupa svim redovima. Pravilo user\_mod\_policy dozvoljava modifikaciju postojećih redova ukoliko se poklapaju korisnička imena i ako je e-mail validan po standardnom regexu za validaciju. Sve ostale operacije u tabeli nisu dozvoljene.



Slika 2. Primer pravila

# Enkriptovanje pomoću pgcrypto modula

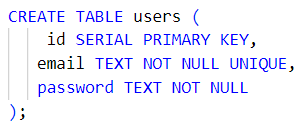
Čak i sa postavljenim ograničenjima pristupa, još uvek postoji mogućnost nedozvoljenog pristupa podacima u slučaju da je napadač dobio neovlašćen pristup bazi, pogotovo ako ima kredencijale nekog administratora. U ovom slučaju, jedino rešenje je enkripcija podataka.

PostgreSQL ima ugrađen modul za enkripciju - *pgcrypto*. Potrebno je uključiti ekstenziju komandom:

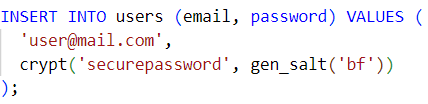
Slika 3. Komanda za uključivanje ekstenzije

Ovaj modul nudi sledeće funkcije:

* digest() – funkcija za izračunavanje hash vrednosti (md5,sha1,sha224,sha256,sha384,sha512 i bilo koji drugi algoritmi koji su podržani od strane OpenSSL biblioteke)
* hmac() – funkcija za izračunavanje *MAC* vrednosti
* funkcije za čuvanje šifri : crypt() i gen\_salt() – čuvanju hash vrednosti šifre kombinovanu sa nasumičnom vrednošću (*salt* vrednost) kako bi bilo teže izračunati hash
* PGP enkripciju
* funkcije za direktnu enkripciju

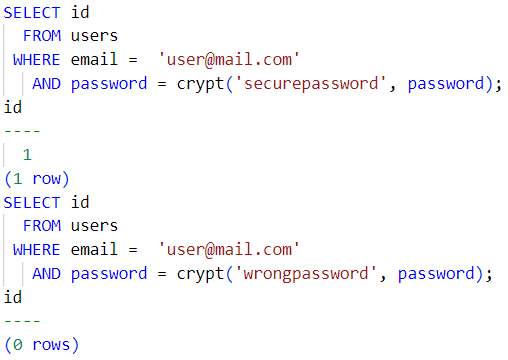
Uzmimo za primer čuvanje šifri korisnika. Ako imamo tabelu korisnika *users* sa ključem, korisničkim imenom i šifrom, definišimo je kao bilo koju tabelu:

Slika 3. Primer tabele



Enkripcija dolazi pri čuvanju podataka odnosno pri *INSERT* upitu:

Slika 3. Enkripcija pri upisu podataka

Koristimo *crypt* i *gen\_salt* funkcije. Argument za generisanje salt vrednosti je „bf“ što označava korišćenje *blowfish* algoritma koji je najbezbedniji u okviru modula. Pri čitanju podatka, imamo poziv samo *crypt* funkcije:

# Dodatna enkripcija pomoću pgsodium alata

Slika 3.4 Čitanje enkriptovanih podataka

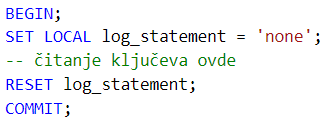
Sa pojavom bržih računara, detaljnom analizom algoritama i najavom kvantnih računara, potrebno je uzeti u obzir da algoritmi i rešenja *pgcrypto* modula se ne mogu smartati kao adekvatna. Pisanje funkcija za enkripciju od nule nije pametno rešenje, a pitanje je kada će PostgreSQL ažurirati svoj modul. Kao privremeno rešenje postoji ekstenzija *pgsodium* koja je zasnovana na *libsodium* biblioteci.

Instalacija ekstenzije može da se obavi na više načina, ali najlakši način za korišćenje je pokretanje PostgreSQL DMBS-a pomoću Docker-a sa nekom slikom koja već ima ugrađenu tu ekstenziju. Za primer ovog rada koristićemo sliku *supabase/postgres*.

Potrebno je uključiti ekstenziju komandom:

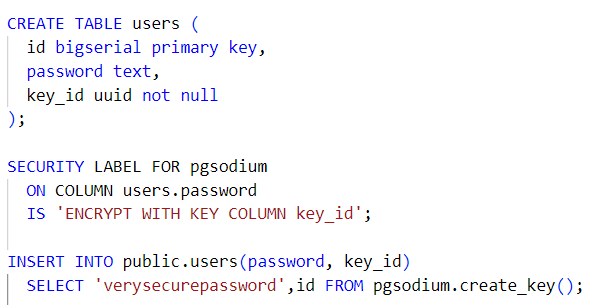
Slika 4.1 Uključivanje ekstenzije

Pomoću ove ekstenzije moguća je transparentna enkripcija kolona, gde su sačuvani podaci uvek enkriptovani, a mogu da se čitaju samo pomoću pogleda. Čak i u slučaju čitanja cele baze, napadač ne bi imao pristup ključevima, a podaci bi bili enkriptovani. Jedini način za dekripciju podataka bi bio ili pokušavanje pronalaženja ključa ručno ili pomoću čitanja ključa koji se ne nalazi unutar DBMS-a.

Enkripcija može da se vrši na nivou kolone gde jedan ključ enkriptuje celu kolonu ili na nivou reda, gde svaki red u tabeli ima poseban ključ. U slučaju da svaki red ima svoj ključ, potrebno je dodati i identifikator ključa u definiciji tabele. Za dodatnu bezbednost unutar upita, potrebno je isključiti logovanje unutar bloka transakcije:

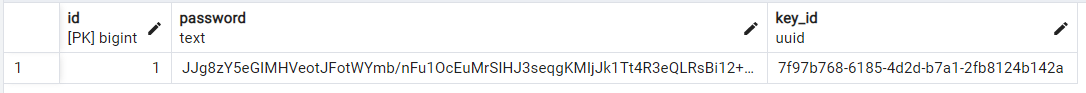
Slika 4.2 Isključivanje logovanja

Primer čuvanja šifre sa *pgsodium* ekstenzijom umesto *pgcrypto* modula:



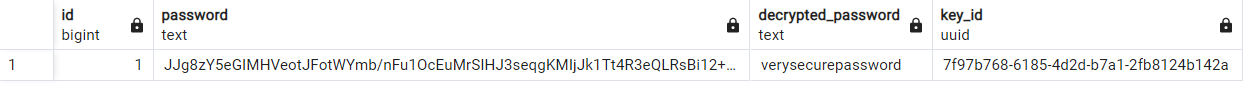
Slika 4.3 Users tabela sa pgcrypto ekstenzijom

Potrebno je čuvati identifikator ključa jer želimo veću sigurnost korisnika. Koristimo bezbednosnu labelu kako bi obavestili ekstenziju da je potrebna enkripcija. Pri čuvanju podataka, potrebno je generisati novi ključ za red pozivom funkcije *pgsodium.create\_key()*. Ako pokušamo da pročitamo ovu tabelu direktno, možemo da vidimo samo enkriptovane podatke:



Slika 4.4 Direktno čitanje tabele

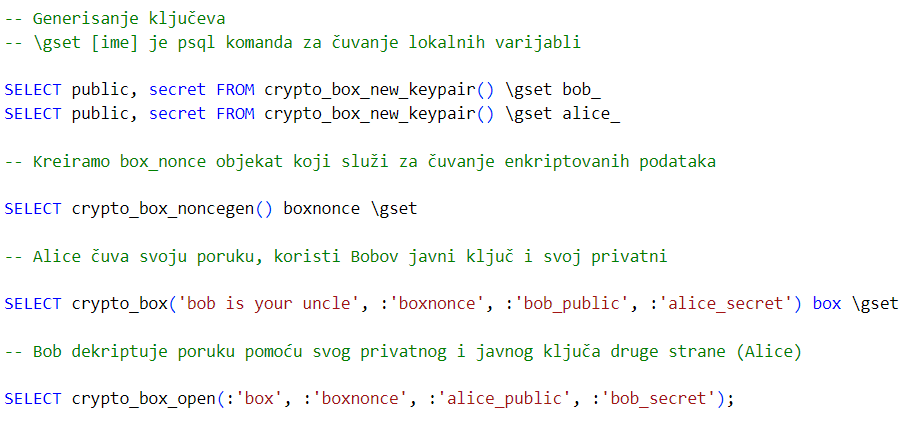
Kako bi uspešno očitali tabelu, potrebno je koristiti pogled iz šeme *pgsodium\_masks*:



Slika 4.5 Čitanje iz pogleda

Čitanjem iz pogleda dobijamo dekriptovanu vrednost kolone, a u kombinaciji sa RLS pravilima možemo osigurati da svaki korisnik može da čita samo svoje redove.

Još jedna mogućnost ove ekstenzije je enkripcija pomoću parova privatnih i javnih ključeva. Podaci mogu da se razmenjuju između 2 korisnika, a ključevi se ne čuvaju unutar baze:



Slika 4.6 Primer razmene informacija pomoću parova privatnih i javnih ključeva

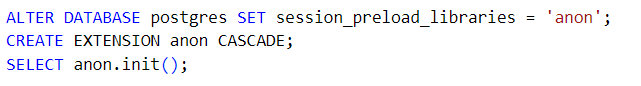
Ova ekstenzija takođe može da radi i sve što radi *pgcrypto* modul, samo sa drugačijim API pozivima i drugačijim algoritmima. *Pgcrypto* je jednostavniji za korišćenje, pa u zavisnosti od potreba aplikacije može se koristiti i jedan i drugi alat.

# Anonimizacija podataka

Pored enkripcije podataka, može biti korisno i vršiti anonimizaciju podataka, bilo to zbog performansi ili zbog saglasnosti sa GDPR regulativama. Ekstenzija koja olakšava ovaj proces je *postgresql\_anonymizer*. Ova funkcija koristi deklarativni način anonimizacije podataka. Svaka tabela ima svoj način maskiranja podatka koji definiše korisnik pri kreiranju tabele pomoću bezbednosnih labela. Pristup podacima je kasnije moguć na sledeće načine:

* anonimni izvozi podataka pomoću *pg\_dump* alata
* statičko maskiranje – modifikacija podataka pomoću pravila
* dinamičko maskiranje – skrivanje podataka za definisane korisnike

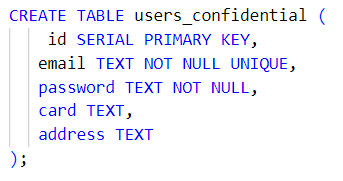
Instalacija može da se vrši na više načina, ali je opet najjednostaviji korišćenje slike koju nude autori ekstenzije (*registry.gitlab.com/dalibo/postgresql\_anonymizer:stable*). Nakon pokretanja DBMS-a potrebno je uključiti ekstenziju pomoću komande:



Slika 5. Uključivanje ekstenzije

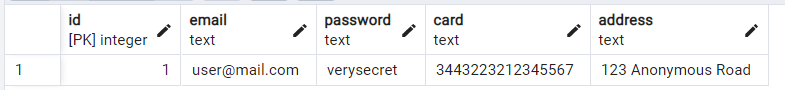
Ugrađene funkcije za anonimizaciju su:

* uništavanje podatka – zamena sa predefisanim vrednostima
* dodavanje šuma – pomeranje originalnih vrednosti za +/- neku vrednost
* nasumično generisanje podataka
* generisanje lažnih podataka
* pseudonimizacija – generisanje lažnih podataka na deterministički način
* generisanje hash-a
* delimično skrivanje podataka
* generalizacija podatka – opisivanje podataka pomoću nekog opsega (npr. Mihajlo ima 25 godina. → Mihajlo ima između 20 i 30 godina.)
* funkcije pisane od strane korisnika

Kao primer anonimizacije, uzećemo tabelu korisnika iz poglavlja 3 i proširiti je dodatkom broja kartice i adrese:

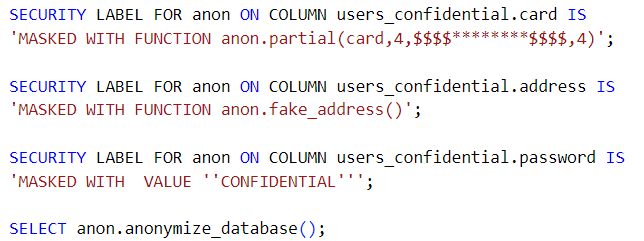
Slika 5. Primer tabele za anonimizaciju

Čitanjem podataka nakon upisa bez ikakvih labela možemo videti podatke:

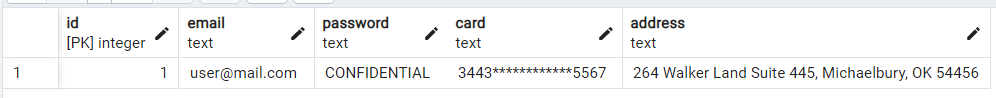


Slika 5. Čitanje podataka bez pravila

Dodatkom pravila za anonimizaciju i komande za statičku anonimizaciju dobijamo sledeće:

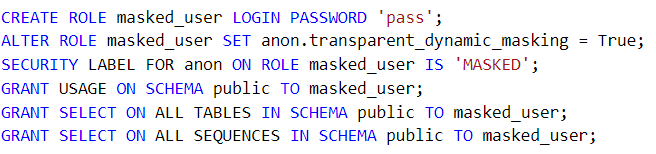


Slika 5. Kreiranje i pokretanje pravila



Slika 5. Čitanje podataka nakon anonimizacije

Za dinamičko maskiranje, potrebno je napraviti novog korisnika za pristup bazi i dodeliti mu labelu:



Slika 5. Dinamičko maskiranje

U ovom slučaju originalni podaci bi ostali sačuvani, ali pristup tabeli od strane bilo kog korisnika koji je obeležen za maskiranje bi dobio identične rezultate kao i statičko maskiranje.

# Zaključak

Čuvanje poverljivih informacija u bilo kom DBMS-u nije trivijalno. Neophodno je imati na umu sve neophodne korake još pri planiraju modela podataka i uz to biti ažuran za sve nove potencijalne napade zlonamernih ljudi. Ne postoji univerzalno i prihvaćeno rešenje, već je potrebno prilagoditi postojeće alate za svaku novu aplikaciju. Potrebno je uspostaviti sistem švajcarskog sira, gde svaki sloj aplikacije ima dovoljno metoda zaštite, da čak i kada napadač uspe da pronađe rupu, postoji nešto drugo što bi sprečilo dalji prolaz. DBMS je među zadnjim slojevima pristupa informacijama, pa je izuzetno bitno imati dovoljne mere zaštite podataka.

Do Maja 2023. godine je bilo čak 20 ogromnih napada koji su uticali na rad vrlo popularnih aplikacija ili aplikacija koje sadrže informacije poput brojeva platnih kartica i ličnih podataka. Ovakvi napadi imaju uticaj na popularnost aplikacija i poverenje korisnika, a potencijalne i pravne posledice. Kao posledica ovih napada, razvijaju se alati koji automatski testiraju i obaveštavaju programere o slabim tačkama u softveru, a korisnike obaveštavaju ukoliko se njihovi podaci pojave na internetu. Ukoliko je aplikacija koristila adekvatne metode za skladištenje podataka, podaci koji budu postavljeni na internetu ne mogu biti iskorišćeni za zlonamerne upotrebe, ali se svakako preporučuje ažuriranje tih podataka (npr. šifri) ukoliko korisnik koristi iste na više različitih aplikacija.

Ovaj rad je ilustrovao samo neke metode zaštite podataka, koje su aktuelne u vreme pisanja rada, a ostaje na čitaocu provera ažurnosti i validnosti metoda.

# Literatura

<https://www.sqlshack.com/using-parameterized-queries-to-avoid-sql-injection/>

<https://www.postgresql.org/docs/current/database-roles.html>

<https://www.postgresql.org/docs/current/role-membership.html>

<https://www.postgresql.org/docs/current/ddl-rowsecurity.html>

<https://luppeng.wordpress.com/2019/11/30/handling-sensitive-data-in-postgresql-row-level-security-column-access-privileges-and-data-masking/>

<https://github.com/michelp/pgsodium>

<https://www.slideshare.net/EnterpriseDB/pgsodiums-features-those-not-provided-by-pgcrypto-and-integration-with-remote-key-management-services-240380390>

<https://www.postgresql.org/docs/current/pgcrypto.html>

<https://postgresql-anonymizer.readthedocs.io/en/latest/>

<https://tech.co/news/data-breaches-updated-list>