Predavanje V.

Napredni SQL DDL

BAZE PODATAKA II

doc. dr. sc. Goran Oreški
Fakultet informatike,
Sveučilište Jurja Dobrile, Pula

Sadržaj

- ponavljanje prethodnih predavanja
  - SQL procedure
  - SQL funkcije
  - kursori
  - iznimke
- trigeri (okidači)
  - definicija
  - sintaksa
  - primjer
- materijalizirani pogledi
- autentifikacija
- autorizacija

Ponavljanje

- **korisnički definirane funkcije**
- specifične operacije koje ne postoje u okviru već definiranih funkcija
- mogu biti definirane u proceduralnom SQL jeziku ili nekom od vanjskih jezika
- funkcija može primiti bilo koji broj argumenata (0 i više)
- funkcije **moraju** vraćati vrijednost
- funkcije se sastoje od:
  - zaglavlja
  - tijela funkcije

Ponavljanje

- **pohranjene procedure**
- pohranjene procedure imaju općenitija svojstava bez nekih ograničenja funkcija
- postoje `IN` i `OUT` argumenti
- za poziv procedure koristi se ključna riječ CALL

```
CALL opterecenje_nastavnika(…)
```

- za korištenje procedure potrebno je navesti i varijable za prihvat vrijednosti koje dolaze iz funkcije
  - `@var` deklarira privremenu varijablu vezanu za sesiju

Ponavljanje

- **kursor** je iteracija kroz redove nekog rezultata
  - odnose se na jedan red u rezultatu upita
  - vrijednostima reda se može pristupati pomoću kursora
  - kursor se može pomicati naprijed kroz rezultate
- kursori se mogu koristiti unutar pohranjenih procedura i korisnički definiranih funkcija
- **iznimka** je događaj koji onemogućava normalan nastavak rada programa tj. zahtijeva posebnu obradu
- za iznimke se mogu definirati rukovatelji (engl. handlers)
- kada se signalizira neko stanje, handler se aktivira

Trigeri

- trigeri (okidači) su proceduralni izrazi (pohranjene procedure) koji se izvode automatski kada se baza podataka mijenja
  - najčešće su specificirani u proceduralnom SQL-u, ali i drugi jezici su često podržani
- na primjer: triger se može implementirati da prati promjene podataka u tablici (INSERT, UPDATE, DELETE)
  - npr. stara vrijednost, datum promjene, korisnik koji je učinio promjenu
- zašto aplikacija sama ne evidentira log direktno?
  - ovisno o implementaciji, moguće je zaboraviti evidentirati neku promjenu
  - zlonamjerni developer može ostaviti prostor za izbjegavanje logova, omogućavajući akcije koje nisu logirane

Trigeri

- ako baza podataka upravlja log podacima automatski i nezavisno
  - aplikacijski kod ne postaje složeniji zbog nove funkcionalnosti
  - log postaje pouzdana informacija promjene podataka predmetne tablice
- trigeri se koriste za mnoge druge aktivnosti kao npr:
  - sprječavanje krivih promjena na podacima tablice
  - automatsko osvježavanje timestamp vrijednosti, izvedenih atributa i sl.
  - izvršavanje poslovnih pravila kada se neki podaci promjene
    - naručivanje proizvoda kada zaliha padne ispod određene razine
  - repliciranje promjena na neku drugu tablicu ili bazu podataka
  - čak i upravljanje referencijalnim integritetom

Triger mehanizam

- triger mehanizam mora voditi evidenciju o dvije stvari
- kada će se triger izvršiti?
  - događaj (engl. event) koji uzrokuje triger
  - uvjet (engl. condition) koji mora biti zadovoljen da bi se triger izvršio
- što triger čini jednom kada se izvrši?
  - akcije (engl. actions) koje se izvode kada se triger aktivira
- naziva se event-condition-action model za trigere ili ECA pravilo
  - event-driven computing
  - u kojem akcije pokreću događaji, s obzirom na postojanje specifičnih uvjeta

Kada se triger izvršava

- baza podataka u pravilu podržavaju trigere za insert, update i delete događaje (DML trigeri)
- ne mogu se definirati za select
  - implikacija: trigerima se ne mogu evidentirati logovi ili spriječiti pristup za čitanje baze podataka
- baze podataka u pravilu podržavaju trigere i na mnogim drugim događajima
  - DDL operacije (create/alter/drop …)
  - login/logout korisnika
  - database stratup, shutdown, errors…

Kada se triger izvršava

- triger se tipično može izvršiti prije i nakon DML događaja koji ga aktivira (*engl. before, after*)
  - u pravilu se DDL/korisnički/baze podataka trigeri izvršavaju nakon događaja, što je logično
  - prije-trigeri mogu prekinuti DML operaciju ako je potrebno
- neka baze podataka podržavaju i umjesto-trigere (*engl. instead of*)
  - izvršava triger umjesto izvođenja triger događaja
- trigeri se mogu izvršavati na razini reda ili naredbe
  - row-level trigger se izvršava za svaki red koji je modificiran naredbom
    - ako zadovoljavaju uvjet ukoliko je postavljen
  - statement-level trigger se izvršavaju jednom za cijelu naredbu

Triger podaci

- row-level trigger može pristupiti staroj i novoj verziji podataka reda kada isti dostupni
  - insert trigeri mogu dohvatiti samo nove podatke reda
  - update trigeri mogu dohvatiti stare i nove verzije podataka
  - delete trigeri mogu dohvatiti samo stare podatke
- trigeri mogu pristupiti i modificirati podatke drugih tablica
  - na primjer dohvat neke vrijednosti tijekom izvođenja

Triger sintaksa

- SQL:99 specificira sintaksu trigera
  - [DSC] poglavlje 5.3
- varijacije ovisno o izdavaču baze podataka
  - Oracle i DB2 su slični standardu ali nisu identični
    - trigeri uključuju značajke specifične za pojedinog izdavača
  - SQLServer, PostgreSQL i MySQL imaju različite značajke
  - ograničenja na mogućnosti trigera također jako variraju i ovise od izdavača do izdavača
  - razlika opisana [DSC] *NONSTANDARD TRIGGER SYNTAX* 184 stranica
- analizirat ćemo različite sintakse trigera i funkcionalnosti

Primjer trigera: prekoračenje računa

- želimo upravljati s prekoračenjima po tekućem računu
- ako promjena na računu dovede do negativnog salda:
  - stvoriti novi kredit s istim ID-em kao i račun
  - postaviti iznos kredita na negativnu vrijednost stanja računa
    - saldo računa je negativan
  - potrebno je ažurirati tablicu dužnik
- potrebno je definirati row-level trigger, koji se izvršava prije ili poslije *update* naredbe na tablici *racun*
  - ukoliko baza podataka podržava postavljanje uvjeta na trigere, postaviti triger na update ako je saldo računa manji do 0

SQL99 / Oracle triger sintaksa

```
CREATE TRIGGER trg_prekoracenje AFTER UPDATE ON racun
REFERENCING NEW ROW AS nrow
FOR EACH ROW WHEN nrow.saldo < 0
BEGIN ATOMIC
   INSERT INTO kredit VALUES (nrow.broj_racuna ,
                              nrow.poslovnica,
                              -nrow.saldo);
   INSERT INTO duznik
   ( SELECT klijent_naziv, broj_racuna
     FROM depozitor AS u
     WHERE nrow.broj_racuna = u.broj_racuna);

   UPDATE racun AS r SET saldo = 0
     WHERE r.broj_racuna = nrow.broj_racuna;
END
```

MySQL triger sintaksa

- MySQL ima ograničene triger sposobnosti u donosu na Oracle
  - izvršavanje trigera je uvjetovano isključivo događajima ne uvjetima
    - rješenje, provođenje uvjeta unutar tijela trigera
  - stari i novi redovi imaju fiksan naziv: OLD i NEW
- promijenimo pravila prekoračenja po računu
  - dodajmo naknadu za prekoračenje
- što će se dogoditi ukoliko je račun već prekoračen?
  - tablica kredit će već imati zapis za prekoračeni račun
  - tablica dužnik će također već imati zapis za kredit
  - zadnja verzija trigera bi prouzročila *duplicate key* grešku

MySQL INSERT dodatak

- MySQL ima nekoliko "poboljšanja" INSERT naredbe
  - ostale baze podatka imaju slične mogućnosti
- pokušaj umetnuti red, ukoliko su atributi ključa isti kao već neki uneseni, nemoj izvršiti insert

```
INSERT IGNORE INTO tbl ...;
```

- pokušaj umetnuti red, ukoliko su atributi ključa isti kao već neki uneseni, napravi update reda

```
INSERT INTO tbl ... ON DUPLICATE KEY
   UPDATE attr1 = value1, ...;
```

- pokušaj umetnuti red, ukoliko su atributi ključa isti kao već neki uneseni, zamjeni stari red s novim

```
INSERT REPLACE INTO tbl ...;
```

MySQL triger sintaksa

```
CREATE TRIGGER trg_prekoracenje BEFORE UPDATE ON racun FOR EACH ROW
BEGIN
     DECLARE prekoracenje_nkd NUMERIC(12, 2) DEFAULT 300;
     DECLARE prekoracenje_izn NUMERIC(12, 2);
-- Ako se prekoračenje dogodilo potrebno je dodati ili ažurirati kredit
     IF NEW.saldo < 0 THEN
-- NEW.saldo je negativan!
       SET prekoracenje_izn = prekoracenje_nkd - NEW.saldo;

       INSERT INTO kredit (broj_racuna, poslovnica, saldo)
         VALUES (NEW.broj_racuna, NEW.poslovnica, prekoracenje_izn)
         ON DUPLICATE KEY UPDATE saldo = saldo + prekoracenje_izn;

       INSERT IGNORE INTO duznik (klijent_naziv, broj_racuna)
         SELECT klijent_naziv, broj_racuna FROM duznik
         WHERE duznik.broj_racuna = NEW.broj_racuna;

       SET NEW.saldo = 0;
     END IF;
END;
```

Triger zamke

- trigeri se mogu ili ne moraju izvršavati kada očekujete
  - MySQL insert-trigeri se izvršavaju kada se podaci grupno dodaju u bazu podataka iz backup datoteke
    - baze podataka u pravilu omogućavaju privremeno isključivanje trigera
- ukoliko se triger za neki uobičajeni događaj izvodi sporo utjecat će na performanse baze podataka
- ukoliko triger ima grešku može izazvati probleme i poništiti događaj u neočekivano vrijeme
  - pravi uzrok može biti teško otkriti
- trigeri mogu koristiti druge tablice koje same mogu imati trigere
  - rezultat može biti beskonačan niz triger događaja

Triger alternative

- trigeri mogu biti implementirani za različite složene zadatke
- primjer: može se implementirati referencijalni integritet koristeći trigere
  - za sve insert i update događaje u referencirajućoj tablici potrebno je provjeriti postoji li ista vrijednost ključnog atributa u referenciranoj tablici
    - ako ne, operacija se prekida
  - za sve update i delete događaje u referenciranoj tablici potrebno je provjeriti postoje li zapisi u referencirajućim tablicama s tom vrijednosti
    - ukoliko da, operacija se prekida ili se akcija kaskadno propagira i sl.
- ovaj pristup je sporiji od standardnog mehanizma

Triger alternative

- mogu li se kao zamjene koristiti pohranjene procedure?
  - pohranjene procedure imaju u pravilu manje ograničenja od trigera
    - p. procedure mogu primati raznovrsnije argumente, vraćati vrijednosti da bi označili uspjeh/neuspjeh, sadržavati out-parametre
    - mogu izvoditi sofisticiraniju obradu transakcija
  - podrška za trigere je specifična za svakog vendora, svaki izbor implementacije ima svoja ograničenja
- u pravilu se trigeri koriste na vrlo ograničenom skupu akcija
  - popunjavanje "verzije reda" ili "timestamp zadnje modifikacije" vrijednosti u mijenjanom redu
  - jednostavne operacije koje ne zahtijevaju puno logike
  - ponekad za replikaciju baze podataka

Triger i sumarne tablice (podaci)

- trigeri se mogu koristiti za računanje sumarnih tablica kada se zapisi u detaljima promjene
- npr. tablica poslovnica koja sadrži sumarne vrijednosti

  poslovnica(poslovnica_naziv, broj_kredita, broj_depozita)
- ideja za takvo vođenje sumarnih podataka ima smisla kada se ti podaci jako često koriste i želi se izbjeći nepotrebno računanje iznosa za svaki upit
  - drugi primjer: saldo na tekućem računu?
- jedno od rješenja je održavanje sumarnih podatka pomoću trigera
  - kad god se dogodi promjena na računima ista se evidentira u tablici poslovnica

Materijalizirani pogledi

- neke baze podataka pružaju materijalizirane poglede, koji implementiraju takvu funkcionalnost
- jednostavni pogledi se mogu smatrati imenovanim SQL upitima
  - tj. izvedene relacije sa specifičnom shemom
- kada upit koristi jednostavan pogled, baza podataka umeće definiciju pogleda direktno u upit
  - omogućava se optimizacija ukupnog upita
  - ukoliko mnogo upita koristi isti pogled baza podataka mora iznova provoditi računanje

Materijalizirani pogledi

- materijalizirani pogledi stvaraju nove tablice koje se popunjavaju s rezultatima definicije pogleda
  - upiti mogu koristiti podatke iz materijaliziranog pogleda neograničeno puta bez da vrše ponovno računanje
  - baze podataka mogu izvršiti optimizirane dohvate podataka iz materijaliziranih pogleda, npr. koristeći indekse (indeksi uskoro!)
- međutim pojavljuje se novi problem?
  - što ukoliko se pogledom referencirane tablice promijene?
  - potrebno je preračunati sadržaj materijaliziranog pogleda
  - to se naziva *održavanje pogleda*

Održavanje materijaliziranih pogleda

- ukoliko baza podataka ne podržava materijalizirane poglede
  - održavanje se može izvoditi pomoću trigera na pogledom referenciranim tablicama
  - ručni pristup održavanja upita, opcija kada baza podataka ne podržava materijalizirane poglede
    - npr. Postgres, MySQL
- baze podataka koje podržavaju materijalizirane poglede će same provoditi održavanje
  - puno jednostavnije od prethodne opcije
  - u pravilu pružaju više opcija
    - trenutno održavanje (engl. immediate) – obnavljanje sadržaja nakon svake promjene
    - odgođeno održavanje (engl. deferred) – obnavljanje sadržaja periodički

Održavanje materijaliziranih pogleda

- jednostavan pristup za ažuriranje materijaliziranih pogleda
  - obnovi cijeli pogled iz početka nakon svake promjene
  - vrlo skup pristup, pogotovo ako se tablice na kojima se pogled temelji često mijenjaju
- bolji pristup je inkrementalno održavanje pogleda
  - temeljem definicije pogleda i specifičnih promjena u podacima na tablicama, ažuriraju se oni dijelovi upita koji su stvarno promijenjeni
- baze podataka s materijaliziranim pogledima će to same napraviti
- inkrementalno održavanje pogleda se može implementirati pomoću trigera ali je kompliciranо

DZ - 2 boda prva dva dobra rješenja

Autentifikacija i autorizacija

- sigurnosni sustavi moraju pružati dvije glavne značajke:
- autentifikacija (*engl. authentication*)
  - znanu kao A1, AuthN, Au
  - "Ja sam onaj za kojeg se izdajem!"
- autorizacija (*engl. authorization*)
  - znanu kao A2, AuthZ, Az
  - "Dozvoljeno mi je činiti ono što hoću!"
- svaka komponenta je beskorisna bez druge

Korisnička autorizacija

- SQL baza podataka provodi autentifikaciju korisnika
  - prilikom spajanja navode se korisničko ime i lozinka
  - baze podržavaju sigurno, enkripirano povezivanje (npr. SSL) i slično
- SQL baza podataka pruža autorizacijske mehanizme za različite operacije
  - različite operacije zahtijevaju različite privilegije
  - korisnicima se mogu dodijeliti privilegije za specifične operacije

    `GRANT`
  - korisnicima se mogu opozvati privilegije za specifične operacije, da bi ograničilo njihovo pravo

    `REVOKE`

Osnovne SQL privilegije

- najosnovniji skup privilegija
  - `SELECT, INSERT, UPDATE, DELETE`
  - omogućavaju ili zabranjuju korisnicima obavljanje određenih operacija
  - korisnicima baze se dodjeljuju privilegije za određenu operaciju na pojedinoj tablici
- sintaksa

```
GRANT SELECT ON racun TO bankar;
```

  - korisniku *bankar* se dodjeljuje privilegija da vrši upite na tablici *racun*

Davanje privilegija

- privilegije se mogu dodijeliti za više korisnika

```
GRANT SELECT, UPDATE ON racun
   TO bankar, menadzer;
GRANT INSERT, DELETE ON racun
   TO menadzer;
```

- bankari mogu dohvaćati i modificirati podatke na računu
- samo menadžeri mogu stvarati i brisati račune
- svaka tablica se mora posebno navoditi

Svi korisnici, sve privilegije

- može se navesti `PUBLIC` za dodjeljivanje privilegija svim korisnicima
  - uključuje i korisnike koji će se u budućnosti dodati

```
GRANT SELECT ON djelatnici TO PUBLIC;
```

- mogu se navesti `ALL PRIVILEGES` za dodjelu svih privilegija nekom korisniku

```
GRANT ALL PRIVILEGES ON racun
   TO admin;
```

Privilegije na razini stupca

- za INSERT i UPDATE operacije, privilegijama se mogu ograničiti specifični stupci u relaciji
  - UPDATE-om se mogu mijenjati određeni stupci
  - INSERT-om se mogu umetnuti podaci u određene stupce
- primjer: relacije djelatnik
  - djelatnici mogu smo promijeniti kontakt informacije
  - dozvoliti HR manipulaciju svim podacima djelatnika

```
GRANT UPDATE (osobni_telefon, email) ON djelatnik
                TO dnk_korisnik;

GRANT INSERT, UPDATE ON djelatnik TO hr_korisnik;
```

Oduzimanje privilegija

- engl. revoking
- oduzimanje privilegija je podjednako jednostavno

```
REVOKE priv1, ... ON relation
    FROM user1, ...;
```

- prilikom oduzimanja privilegija može se navesti više privilegija i više korisnika
- s `INSERT` i `UPDATE` mogu se oduzeti privilegije na pojedinačne stupce relacije
- negdje `DENY` eksplicitno zabranjuje neke privilegije

Privilegije i pogledi

- korisnicima se mogu dodijeliti privilegije na poglede
  - mogu se razlikovati u odnosu na privilegije na osnovne tablice
- kada se koristi pogled
  - provjeravaju se privilegije pogleda, ne privilegije tablica na kojima se pogled temelji
- primjer: djelatnik relacija
  - samo HR može vidjeti sve podatke o djelatniku
  - djelatnik može vidjeti samo ograničene podatke

Primjeri privilegija na pogledu

- SQL naredbe:

```
-- Oduzmimo sve pristupe djelatniku
REVOKE ALL PRIVILEGES ON djelatnik TO PUBLIC;
-- Odobrimo pristup djelatniku samo hr_korisniku
GRANT ALL PRIVILEGES ON djelatnik TO hr_korisnik;
-- Pogled kojim će pristupati drugi djelatnici
CREATE VIEW d_podaci AS
  SELECT ime, email, telefon
  FROM djelatnik;
GRANT SELECT ON d_podaci TO dnk_user;
```

- kada djelatnik koristi pogled, provjeravaju se samo privilegije za *d_podaci*

Obrada pogleda

- baza podataka se u pravilu odnosi prema pogledima kao imenovanim upitima
  - na mjesto referenciranja pogleda, unutar upita, se direktno umeće definicija pogleda
- SQL sustav radi autorizaciju prije nego se taj proces dogodi
  - baza podataka provjerava privilegije pristupa referenciranog pogleda
  - potom umeće njegovu definiciju u plan upita
  - to omogućava sustavima baze podataka da podržavaju drugačija ograničenja pristupa pogledu
    - u odnosu na tablice na kojima se temelji

Druge privilegije

- postoje i druge privilegije u SQL-u
  - `EXECUTE` - dodjeljuje se privilegija za izvođenje funkcije ili pohranjene procedure
  - `CREATE` – dodjeljuje se privilegija za stvaranje tablica, pogleda i ostalih shema objekata
  - `REFERENCES` – dodjeljuje se privilegija za stvaranje ograničenja stranog ključa ili CHECK ograničenja
- većina DBMS-a pruža i druge privilegije
  - Postgre omogućava 11, MySQL 27
  - Oracle otprilike 200

REFERENCES privilegija

- ograničenje stranog ključa ograničava neke akcije korisnika
  - redovi u referencirajućoj relaciji ograničavaju UPDATE i DELETE u referenciranoj relaciji
  - korisnik koji dodaje strani ključ može zabraniti te operacije svim korisnicima baze podataka
- zbog toga je potrebna REFERENCES privilegija da bi se mogli dodati strani ključevi
- REFERENCES zahtjeva da se navedu i relacija i atributi jer se mogu stvoriti strani ključevi koji uključuju te atribute

Pravo dodjeljivanja privilegija

- korisnici ne mogu automatski dodjeljivati privilegije koje posjeduju drugim korisnicima
- to se eksplicitno treba navesti

```
GRANT SELECT ON djelatnik TO hr_korisnik
   WITH GRANT OPTION;
```

  - WITH GRANT OPTION omogućava prosljeđivanje dalje privilegija
- može dovesti to zbunjujućih situacija:
  - ako Ivan odobri privilegiju Ani a potom njegove privilegija bude oduzeta, da li to utječe na Anu?
  - ako Ivan i Ana odobre privilegiju Marku a potom Ivan povuče privilegiju, dali Marko ima privilegiju?

Autorizacija - dodatak

- SQL mehanizam autorizacije je vrlo bogat
- ali ima i neke nedostatke
  - nije moguće davati/oduzeti privilegije na razini reda
    - npr. nije moguće dodijeliti privilegiju da korisnik vidi samo svoje zapise u tablici računa banke
    - da postoje SELECT trigeri to bi mogli implementirati
    - može se postići koristeći funkcije koje vraćaju tablice
  - postoji značajna razlika u implementaciji sigurnosti u različitim bazama podataka

Autorizacija - dodatak

- većina aplikacija se ne oslanja u prevelikoj mjeri na autorizaciju baze podataka
  - aplikacije mogu implementirati široki raspon autorizacijskih pravila, kompleksnost implementacije raste
  - web aplikacije su dobar primjer takvog slučaja
  - pristup bazi podataka se u pravilu vrši pomoću jednog korisnika koji ima sva prava za rad s bazom podataka (oprezno koristiti!)
- aplikacije same izvode autentifikaciju i autorizaciju
  - kontrole pristupa mogu biti raspršene po kodu aplikacije, što može dovesti do sigurnosnih propusta
  - ap. serveri s deklarativnom sigurnosnom specifikacijom učinkovito rješavaju taj problem (npr. J2EE Declarative Security)

Autorizacija - dodatak

- dobro je u nekoj mjeri koristiti SQL auth mehanizam
  - baza podataka jednostavno ne dozvoljava pristup zaštićenim podacima niti neovlaštenu promjenu sheme
- za velike, bitne aplikacije baze podataka potrebno je na neki način koristiti SQL mehanizam autorizacije
  - dobro je barem stvoriti DBMS korisnika za svaku ulogu (rolu) koju podržava aplikacija
  - "admin" korisnika za administraciju s manje ograničenja
  - "običnog" korisnika s ograničenim pravima
  - na taj način se značajno može smanjiti opasnosti od SQL napada

Literatura

- Pročitati
  - [DSC] poglavlje 4.6.
  - [DSC] poglavlje 5.3.
  - Caltech CS121 - 10
- Slijedeće predavanje
  - [DSC] poglavlje 11.1. – 11.4
  - korisno ([DSC] poglavlje 10.5)
  - Caltech CS121 - 11