

# Predavanje IV.

*SQL pohranjene procedure*

## BAZE PODATAKA II

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

# Sadržaj

- ponavljanje prethodnih predavanja
  - odgođena ograničenja
  - datum i vrijeme
  - large data objects
  - korisnički definirani tipovi
  - privremene tablice
    - primjeri
    - korištenje
- SQL funkcije
- SQL procedure
- kursori
- iznimke
- upravitelji iznimkama
- primjer
- zadatak

# Ponavljjanje

- za neke operacije, najčešće one koje se izvode u okviru velike transakcije, je potrebno privremeno prekršiti ograničenje
- primjena ograničenja se odgađa (*engl. defer*) za kraj transakcije
  - na kraju transakcije sve ograničenja se provjeravaju prema odgođenim ograničenjima
- u SQL ograničenja se mogu definirati kao odgođena
  - neki sustavi ne podržavaju odgođena ograničenja!
- DEFERRABLE ograničenja:
  - INITIALLY IMMEDIATE primjenjuju se odmah kao zadano ponašanje
  - INITIALLY DEFERRED primjenjuju se na kraju transakcije kao zadano ponašanje

# Ponavljjanje

- vrijednosti datuma i vremena
  - TIME
  - DATE
  - TIMESTAMP
  - ... WITH TIMEZONE
  - .. (P)
  - funkcije: `CURRENT_DATE()` , `CURRENT_TIME()` , `NOW()`
- large objects
  - BLOB
  - CLOB

# Ponavljjanje

- privremene tablice sintaksa

`CREATE TEMPORARY TABLE ...`

- privremene tablice mogu značajno poboljšati performanse nekih upita
- pristup:
  - stvoriti privremenu tablicu za pohranu međurezultata koji su korisni ali zahtjevni za izračun
    - ne preporuča se korištenje puno (ijedno) ograničenja jer može usporiti izvođenje
  - popuniti privremenu tablicu pomoću `INSERT ... SELECT` naredbe
  - korištenje privremene tablice za računanje željenih rezultata
  - privremena tablica završava nakon prekida transakcije ili završetka sesije

# SQL funkcije

- SQL upiti mogu koristiti složene matematičke operacije i funkcije
  - neke koje smo koristili do sada
    - npr.?
  - korišćenje jednostavnih funkcija i agregacija
  - računanje i filtriranje rezultata
- ponekad, za potrebe aplikacije je potrebno izvoditi specifične operacije koje ne postoje u okviru već definiranih funkcija
  - npr. obračun kamata ili plaće
- SQL pruža mehanizam za definiranje funkcija
  - korisnički definirane funkcije (*engl. user defined functions (UDF)*)

# SQL funkcije

- mogu biti definirane u proceduralnom SQL jeziku ili nekom od vanjskih jezika
  - SQL:1999 i SQL:2003 standardi definiraju jezik kojim se deklariraju funkcije i procedure
- različiti izdavači definiraju vlastite jezike
  - Oracle: PL/SQL (Procedural Language for SQL)
  - Microsoft: T-SQL (Transact-SQL)
  - PostgreSQL: PL/pgSQL (Procedural Language/PostgreSQL)
  - MySQL: podrška za pohranjene procedure nastoji slijediti specifikaciju
- različite mogućnosti i sintaksa između pojedinih verzija

# Primjer SQL funkcije

- *napišite funkciju koja vraća ukupan broj kolegija na kojem neki nastavnik predaje*

```
CREATE FUNCTION broj_kolegija(  
    nastavnik_id INTEGER  
) RETURNS INTEGER  
BEGIN  
    DECLARE k_zbroj INTEGER;  
    SELECT count(*) INTO k_zbroj FROM predaje AS p  
        WHERE p.nastavnik_id = nastavnik_id;  
    RETURN k_zbroj;  
END
```

- funkcija može primati argumente i vraća vrijednost
- može koristiti SQL naredbe i druge operacije unutar tijela funkcije



# Primjer SQL funkcije

- definirana funkcija se može koristiti za pojedinačne profesore

```
SELECT broj_kolegija(127) AS broj_kolegija;
```

broj_kolegija
4

- može se uključiti u rezultate upita

```
SELECT ime, prezime, broj_kolegija(id) AS broj_kolegija  
FROM nastavnik ORDER BY broj_kolegija DESC;
```

- i u WHERE dio upita

```
SELECT ime, prezime FROM nastavnik  
WHERE broj_kolegija(id) > 2;
```

ime	prezime	broj_kolegija
Ivan	Topić	4
Stjepan	Kralj	2
Tomislav	Vidaković	2
Martino	Blažević	2
Matej	Jurić	2
Matej	Đurić	2
Saša	Marjanović	2

...

# Argumenti i povratne vrijednosti

- funkcija može primiti bilo koji broj argumenata (0 i više)
- funkcije **moraju** vraćati vrijednost
  - potrebno je specificirati tip povratne vrijednosti s RETURNS ključnom riječi
- iz prethodnog primjera

```
CREATE FUNCTION broj_kolegija(  
    nastavnik_id INT  
) RETURNS INT
```

- zaglavlje funkcije se sastoji od naziva funkcije i RETURNS ključne riječi
- funkcija prima jedan argument tipa INTEGER
- funkcija vraća vrijednost tipa INTEGER

# Tablične funkcije

- SQL:2003 uključuje tablične funkcije (engl. table functions)
  - kao rezultat vraća tablicu
  - može se koristiti u FROM dijelu upita
- generalizacija pogleda
  - mogu se smatrati kao parametrizirani pogledi
  - funkcija se poziva sa specifičnim argumentima
  - rezultat je relacija temeljena na tim argumentima
- većina DBMS-ova pruža ovu značajku implementiranu na neki od načina
  - na neki od različitih načina!

# Tijela funkcije i varijable

- blokovi proceduralnih SQL naredbi su omeđeni s `BEGIN` i `END` ključnim riječima
  - definira složenu naredbu
  - mogu se stvarati ugniježđeni `BEGIN ... END` blokovi
- varijable se deklariraju pomoću `DECLARE` naredbe
  - moraju se deklarirati na početku bloka
  - inicijalna vrijednost je `NULL`
  - mogu se inicijalizirati s drugom vrijednosti koristeći `DEFAULT`
  - doseg (scope) varijable je blok
  - varijable u unutarnjem bloku mogu zakriti varijable vanjskog bloka

# Tijela funkcije i varijable

- tijelo funkcije broj\_kolegija

```
BEGIN
  DECLARE k_zbroj INTEGER;
  SELECT count(*) INTO k_zbroj FROM predaje AS p
    WHERE p.nastavnik_id = nastavnik_id;
  RETURN k_zbroj;
END
```

- jednostavna integer varijabla s postavljenom početnom vrijednosti

```
BEGIN
  DECLARE var INTEGER DEFAULT 0;
  ...
END
```

# Primjer funkcije u PL/SQL-u

- PL/SQL varijanta definiranja funkcije

```
CREATE [OR REPLACE] FUNCTION function_name (parameter_list)
    RETURN return_type
IS
    [declarative section]
BEGIN
    [executable section]
[EXCEPTION]
    [exception-handling section]
END;
```

- specifičnost:
  - parametri mogu biti IN, OUT, INOUT
  - deklarativni dio tijela funkcije

# Dodjeljivanje vrijednosti varijabli

- može se koristiti `SELECT ... INTO` sintaksa

- za dodjeljivanje rezultata upita varijabli

```
SELECT count(*) INTO k_zbroj FROM predaje AS p
      WHERE p.nastavnik_id = nastavnik_id;
```

- upit mora generirati jedan red
  - `SELECT INTO` može imati višestruko značenje, u ovoj formi se koristi za dodjeljivanje vrijednosti varijabli unutar tijela funkcije
    - može se koristiti i za stvaranje privremene tablice pomoću `SELECT` naredbe

- može se koristiti i `SET` ključna riječ

- npr. za dodjeljivanje rezultata matematičkog izraza u varijablu

```
SET rezultat = n * (n + 1) / 2;
```

## ...više varijabli

- može se dodijeliti vrijednost više varijabli koristeći SELECT INTO sintaksu
- primjer: želimo ukupan broj kolegija i njihovu ukupnu vrijednost ECTS bodova

```
DECLARE k_zbroj INTEGER;  
DECLARE ects_zbroj INTEGER;
```

```
SELECT COUNT(*), SUM(ects) INTO k_zbroj, ects_zbroj  
FROM predaje AS p NATURAL JOIN kolegij  
WHERE p.nastavnik_id = nastavnik_id;
```



# Primjer

- jednostavna funkcija koja računa sumu cijelih brojeva od 1 do N

```
CREATE FUNCTION suma_n(n INTEGER) RETURNS INTEGER
BEGIN
    DECLARE rezultat INTEGER DEFAULT 0;
    SET rezultat = n * (n + 1) / 2;
    RETURN rezultat;
END
```

- ili jednostavnije

```
CREATE FUNCTION suma_n(n INTEGER) RETURNS INTEGER
BEGIN
    RETURN n * (n + 1) / 2;
END
```

# Brisanje funkcija

- funkcije se ne mogu jednostavno prepisivati preko postojećih
  - kao i tablice, pogledi, ...
- prvi korak je brisanje postojeće funkcije

```
DROP FUNCTION suma_n;
```

- potom se stvara nova verzija

```
CREATE FUNCTION suma_n(n INTEGER) RETURNS INTEGER  
BEGIN  
    RETURN n * (n + 1) / 2;  
END
```

# SQL procedure

- funkcije imaju određena ograničenja
  - moraju vratiti vrijednost
  - svi argumenti su ulazni (postoje iznimke u nekim sustavima)
  - u pravilu ne mogu utjecati na trenutni status transakcije
    - tj. funkcija ne može napraviti commit ili rollback
  - u pravilu im nije dozvoljeno modificirati tablice, osim u nekim iznimnim slučajevima
- pohranjene procedure imaju općenitija svojstva bez navedenih ograničenja
  - ne mogu se koristiti na svim mjestima kao funkcije (npr. uz SELECT)
  - procedure ne vraćaju vrijednost na način kako to rade funkcije

# Primjer procedure

- *napišite proceduru koja vraća broj kolegija i ukupan broj ects bodova za profesora*
  - rezultati se proslijeđuju pomoću OUT parametara

```
CREATE PROCEDURE opterecenje_nastavnika(  
    IN nastavnik_id INT,  
    OUT k_zbroj INTEGER,  
    OUT ects_zbroj INTEGER  
)  
BEGIN  
    SELECT COUNT(*), SUM(ects)  
    INTO k_zbroj, ects_zbroj  
    FROM predaje AS p NATURAL JOIN kolegij  
    WHERE p.nastavnik_id = nastavnik_id;  
END
```

**- zadani tip parametra je IN**

# Pozivanje procedure

- za poziv procedure koristi se ključna riječ **CALL**  
`CALL opterecenje_nastavnika (...)`
- za korištenje procedure potrebno je navesti i varijable za prihvrat vrijednosti koje dolaze iz funkcije
- MySQL sintaksa:  
`CALL opterecenje_nastavnika(127,@broj, @ects);`  
`SELECT @broj, @ects;`
- `@var` deklarira privremenu varijablu vezanu za sesiju

+-----+-----+	
@broj	@ects
+-----+-----+	
4	23
+-----+-----+	

# Kontrolna struktura selekcija

- SQL pruža IF-THEN-ELSE kontrolnu strukturu selekcije
  - također i CASE (tj. SWITCH)

```
IF cond1 THEN command1
ELSEIF cond2 THEN command2
ELSE command3
END IF
```

- pojedine sekvence mogu biti složene, tj. ne moraju se sastojati od samo jedne naredbe
  - složene sekvence moraj biti ograničene s BEGIN i END
- ELSEIF i ELSE se mogu izostaviti, ovisno o potrebi

# Kontrolna struktura iteracija

- SQL definira i kontrolne strukture iteracije

- WHILE petlja

```
DECLARE n INTEGER DEFAULT 0;  
WHILE n < 10 DO  
    SET n = n + 1;  
END WHILE;
```

- REPEAT petlja

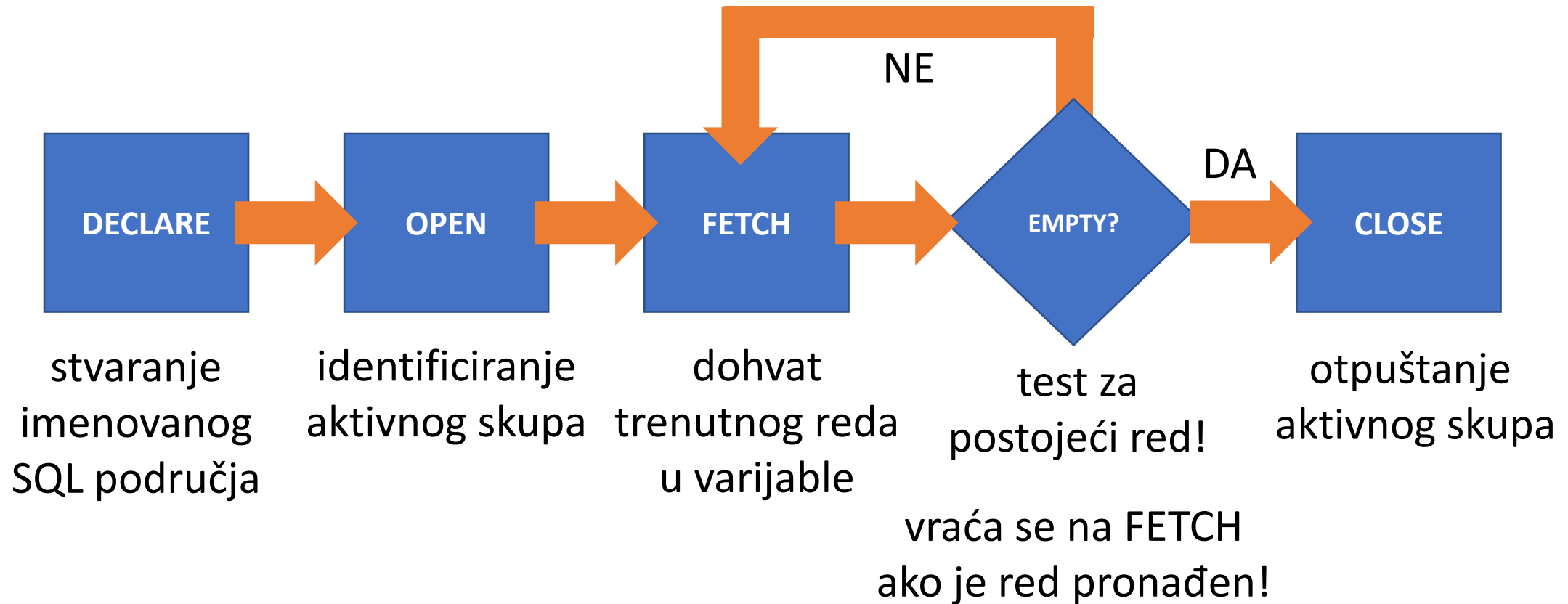
```
REPEAT  
    SET n = n - 1;  
UNTIL n = 0  
END REPEAT;
```

# Kursori

- ponekad (često!) je potrebno zadati upit te pojedinačno proći kroz sve redove rezultata
  - koristi se za izvršavanje složenih operacija koje se ne mogu obaviti klasičnim SQL upitom (u suštini namjena pohranjenih procedura)
- kursor (*engl. cursor*) je iteracija kroz redove nekog rezultata
  - odnose se na jedan redu u rezultatu upita
  - vrijednostima reda se može pristupati pomoću kursora
  - kursor se može pomicati naprijed kroz rezultate
- različite varijante kursora
  - read-only vs read-write
  - samo u smjeru naprijed vs oba smjera
  - static vs. dynamic



# Kontroliranje kursora



# Kursori - dodatak

- kursori mogu biti zahtjevni tj. skupi za bazu podataka
- može li se operacija napisati pomoću klasičnog SQL-a?
  - u pravilu je odgovor da
  - kursorom se mogu zadati sve operacije ali je izvršavanje sporije
- kursori mogu zadržavati resurse sustava dok ne završe
- sintaksa se značajno razlikuje ovisno o izdavaču
  - prenosivost je vrlo teška
- većina vanjskih API-ja za konekciju na bazu nude iste mogućnosti kao i kursor

# Pohranjene rutine i kursori

- kursori se mogu koristiti unutar pohranjenih procedura i korisnički definiranih funkcija
- sintaksa

```
DECLARE n INTEGER DEFAULT 0;  
FOR r AS SELECT primanja FROM nastavnik  
          WHERE fakultet ='Medicina'  
DO  
    SET n = n + r.balance;  
END FOR
```

- iteracija kroz primanja nastavnika s medicine i sumiranje plaća
- `r` je implicitno kursor
- primjer koda čiji rezultat se može postići s jednostavnom SQL naredbom

# MySQL kursor sintaksa

- za korištenje kursora potrebno je:

1. eksplicitno definirati kursor varijablu

```
DECLARE cur CURSOR FOR  
SELECT ... ;
```

2. otvoriti kursor za korištenje rezultata upita

```
OPEN cur;
```

3. dohvatiti vrijednosti iz kursora u varijable

```
FETCH cur INTO var1, var2, ... ;
```

- dohvaća se slijedeći red te se vrijednosti pohranjuju u varijable
- mora se navesti jednaki broj varijabli kao i stupaca u rezultatu
- kraj rezultata se označava s posebnom vrstom SQL iznimke

# MySQL kursor sintaksa

## 4. zatvoriti kursor na kraju operacije

`CLOSE surr,`

- kursor se zatvara automatski na kraju bloka

# Rukovanje iznimkama

- greške se mogu pojavljivati na puno mjesta unutar pohranjenih procedura
  - nazivaju se stanjima ili iznimkama (*engl. conditions*)
  - uključuju greške, upozorenja i druge iznimke
  - mogu se definirati i vlastite korisničke iznimke
- 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
  - handler može specificirati hoće li procedura nastaviti s izvođenjem ili će se izaći iz nje

# Iznimke

- predefinirane iznimke
  - NOT FOUND
    - upit nije dohvatio niti jedan zapis ili naredba nije obradila rezultate
  - SQLWARNING
    - signalizira ne kritični SQL događaj, tj. upozorenje
  - SQLEXCEPTION
    - dogodila se ozbiljna greška

# Iznimke

- iznimke se mogu definirati za specifične potrebe aplikacije
  - nedopušteno stanje na računu
  - stanje zaliha na nuli

- sintaksa za definiranje iznimki

```
DECLARE prekoracenje_racun CONDITION
```

```
DECLARE prazne_zalihe CONDITION
```

- ne podržava svaki sustav generičke iznimke
  - MySQL podržava dodjeljivanje imena već postojećim kodovima greški ali ne stvaranje novih



# Rukovatelji

- rukovatelj (*engl. handler*) se može definirati za specifičnu iznimku
- definira naredbu koja će se izvršiti
- također definira akciju koja će se dogoditi
  - nastavak izvršavanja pohranjene procedure gdje je stala
  - izlazak iz procedure
- sintaksa
  - a continue-handler  
**DECLARE CONTINUE HANDLER FOR condition statement**
  - an exit-handler  
**DECLARE EXIT HANDLER FOR condition statement**
  - umjesto jedne naredbe može se definirati blok naredbi


# Rukovatelji

- rukovatelji mogu biti zaduženi sa jednostavne operacije
  - postavljanje indikatora (*engl. flag*) da se dogodilo neko stanje
- mogu obaviti i složene operacije
  - umetnuti redove u drugu tablicu da bi evidentirali problem koji je nastao (tj. napraviti log)
  - pokrenuti proceduru narudžbe zaliha

# Ukupan broj ECTS-a na fakultetu

- koristiti ćemo funkciju jer vraćamo vrijednost - MySQL
- i pretpostaviti da ECTS bodovi mogu biti decimalni, u stvarnosti ne mogu

```
CREATE FUNCTION ects_total(naziv_fakulteta VARCHAR(20))  
RETURNS NUMERIC(12,2)  
READS SQL DATA  
DETERMINISTIC  
BEGIN  
    -- Varijable potrebne za operaciju  
    DECLARE ects NUMERIC(12,2);  
    DECLARE total NUMERIC(12,2) DEFAULT 0;
```



# Ukupan broj ECTS-a na fakultetu

```
-- Kursor i indikator da označava kada je dohvaćanje gotovo
DECLARE done INT DEFAULT 0;
DECLARE cur CURSOR FOR
    SELECT k.ects
    FROM nastavnik n JOIN predaje p
    ON n.id = p.nastavnik_id
    NATURAL JOIN kolegij k
    WHERE n.fakultet = naziv_fakulteta;

-- Kada je dohvat gotov, rukovatelj postavlja indikator
-- 02000 je MySQL greška za "zero rows fetched"
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
    SET done = 1;
```

# Ukupan broj ECTS-a na fakultetu

```
OPEN cur;  
  REPEAT  
    FETCH cur INTO ects;  
    IF NOT done THEN  
      SET total = total + ects;  
    END IF;  
  UNTIL done END REPEAT;  
CLOSE cur;  
RETURN total;  
END
```

# Korištenje definirane funkcije

- možemo izračunati ukupan broj ects-a po svim fakultetima na sveučilištu

```
SELECT DISTINCT fakultet, ects_total(fakultet)
      AS 'Total ects' FROM nastavnik;
```

fakultet	Total ects
Informatika	35.00
Fizika	60.00
Književnost	57.00
Medicina	14.00
Kemija	39.00
Strojarstvo	55.00

# Pohranjene rutine - prednosti

- vrlo učinkovito za manipuliranje velikih količina podataka na zahtjevne načine unutar baze podataka
  - nema gubitka vremena u komunikaciji slanja naredbi ili izmjene podataka
  - baze podataka mogu često izvoditi te operacije učinkovitije nego što mogu aplikacije
- koriste se za pružanje sigurnog sučelja prema podacima
  - slično konceptu skrivanja podataka objektno orijentiranog programiranja
  - skrivanje tablice i omogućavanje interakcije kroz pohranjene rutine
- enkapsulacija poslovnih pravila u pohranjene procedure
  - zabrana nedozvoljenih stanja preusmjeravajući sve operacije kroz procedure

# Pohranjene rutine - nedostatci

- povećanje opterećenja na bazu podataka
  - može negativno utjecati na performanse svih operacija koje se izvode na DBMS
  - koristi se kada operacija baš zahtjeva korištenje pohranjene procedure
    - većina njih ne trebaju pohranjene procedure
    - npr. prikazani primjer
- vrlo ih je teško migrirati na drugi DBMS
  - različiti proceduralni jezici imaju različite mogućnosti i ograničenja
  - ovisi o izdavaču



# Zadatak

- definirajte shemu baze podataka i napišite pohranjene rutine kojima se upravlja prodajom i skladištem proizvoda neke trgovine
- podržane operacije
  - naručivanje proizvoda
  - zaprimanje proizvoda
  - prodaja proizvoda
- shema:

```
proizvod(proizvod_id, sifra, naziv, stanje, cijena)  
narudzbenica(narudzbenica_id, proizvod_id, kolicina, datum)  
racun(racun_id, datum)  
racun_stavka(stavka_id, racun_id, proizvod_id, kolicina)
```

# Zadatak

- stvorimo bazu podataka po zadanoj shemi

```
CREATE TABLE proizvod(  
    proizvod_id SERIAL PRIMARY KEY,  
    sifra VARCHAR(6) NOT NULL UNIQUE,  
    naziv VARCHAR(100) NOT NULL,  
    stanje INT NOT NULL DEFAULT 0,  
    cijena NUMERIC(15,2) NOT NULL)
```

- dodajmo dva proizvoda (i recimo da je to napravljeno iz programa)

```
INSERT INTO proizvod(sifra, naziv, stanje, cijena)  
VALUES('P100', 'Racunalo Asus',10,3500.00);  
INSERT INTO proizvod(sifra, naziv, cijena)  
VALUES('P101', 'Tipkovnica',200.00);
```

# Zadatak

- tablicu narudžbenica

```
CREATE TABLE narudzbenica(  
    narudzbenica_id SERIAL PRIMARY KEY,  
    proizvod_id BIGINT UNSIGNED NOT NULL,  
    kolicina INT NOT NULL,  
    datum_vrijeme TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),  
    aktivna CHAR(1) NOT NULL DEFAULT 'D',  
    FOREIGN KEY (proizvod_id) REFERENCES proizvod(proizvod_id),  
    CHECK(aktivna IN ('D','N'))  
)
```

- nećemo dodavati narudžbenice ručno, to ćemo prepustiti pohranjenoj proceduri

# Zadatak

- račun i stavke računa

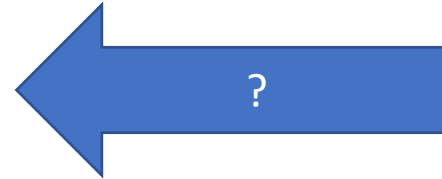
```
CREATE TABLE racun(  
    racun_id SERIAL PRIMARY KEY,  
    datum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP())
```

```
CREATE TABLE racun_stavka(  
    stavka_id SERIAL PRIMARY KEY,  
    racun_id BIGINT UNSIGNED NOT NULL,  
    proizvod_id BIGINT UNSIGNED NOT NULL,  
    kolicina INT NOT NULL,  
    FOREIGN KEY (racun_id) REFERENCES racun(racun_id),  
    FOREIGN KEY (proizvod_id) REFERENCES proizvod(proizvod_id),  
    CHECK(kolicina > 0))
```

# Zadatak

- dodajmo dva računa i njihove stavke
- to bi se tipično dodalo iz programa i u ovom slučaju simuliramo unos iz programa

```
INSERT INTO racun VALUES ();  
INSERT INTO racun VALUES ();
```



```
INSERT INTO racun_stavka(racun_id, proizvod_id, kolicina)  
VALUES (1,1,5);  
INSERT INTO racun_stavka(racun_id, proizvod_id, kolicina)  
VALUES (1,2,3);  
INSERT INTO racun_stavka(racun_id, proizvod_id, kolicina)  
VALUES (2,1,100);
```

# Zadatak

- tablice

*proizvod*

proizvod_id	sifra	naziv	stanje	cijena
1	P100	Racunalo Asus	10	3500.00
2	P101	Tipkovnica	0	200.00

*racun NATURAL JOIN racun\_stavka*

racun_id	datum	stavke_id	proizvod_id	kolicina
1	2019-11-13 11:47:33	1	1	5
1	2019-11-13 11:47:33	2	2	3
2	2019-11-13 11:47:33	3	1	100

*narudzbenica*

Empty set

# Zadatak

- stvaranje narudžbenice za proizvod na skladištu
  - provjeravamo količinu proizvoda na skladištu, politika tvrtke je da se ne naručuju proizvodi koji imaju stanje veće od 5 na skladištu
  - želimo informaciju o statusu narudžbe

```
CREATE PROCEDURE izdaj_narudzbenicu (IN sifra_proizvoda VARCHAR(6),  
                                     IN kolicina INT, OUT status_narudzbe VARCHAR(100))  
  
BEGIN  
    DECLARE stvarno_stanje INT DEFAULT NULL;  
    DECLARE pro_id INT;  
  
    SELECT proizvod_id, stanje INTO pro_id, stvarno_stanje  
    FROM proizvod  
    WHERE sifra = sifra_proizvoda;
```



potrebne varijable



dohvat podataka

# Zadatak

```
IF stvarno_stanje IS NULL THEN
    SET status_narudzbe = 'Proizvod nije unesen u evidenciju!';
ELSEIF stvarno_stanje >= 5 THEN
    SET status_narudzbe = 'Narudzba otkazana, stanje na skladistu dostatno!';
ELSE
    INSERT INTO narudzbenica(proizvod_id, kolicina) VALUES(pro_id, kolicina);
    SET status_narudzbe = 'Narudzbenica stvorena';
END IF;
END;
```

- unutar pohranjene procedure implementirali smo poslovnu logiku za kreiranje narudžbenice
- otvaramo pristup tablici narudžbenica isključivo kroz tu proceduru



# Zadatak

- korišćenje procedure za izdavanje narudžbenice

```
CALL izdaj_narudzbenicu('P100', 10, @status);
```

```
SELECT @status;
```

```
+-----+
| @status                                     |
+-----+
| Narudzba otkazana, stanje na skladistu dostatno! |
+-----+
```

```
CALL izdaj_narudzbenicu('P102', 10, @status);
```

```
SELECT @status;
```

```
+-----+
| @status                                     |
+-----+
| Proizvod nije unesen u evidenciju! |
+-----+
```

```
CALL izdaj_narudzbenicu('P101', 10, @status);
```

```
SELECT @status;
```

```
+-----+
| @status                                     |
+-----+
| Narudzbenica stvorena |
+-----+
```

# Zadatak

- zaprimanje proizvoda
  - proizvodi se zaprimaju na temelju narudžbenice
  - kada proizvodi stignu u skladište narudžbenica više nije aktivna

```
CREATE PROCEDURE zaprimanje_proizvoda
    (IN narudz_id INT, OUT status VARCHAR(100))
BEGIN
    DECLARE akt CHAR(1) DEFAULT 'N';
    DECLARE pro INT DEFAULT NULL;
    DECLARE kol INT DEFAULT NULL;

    SELECT proizvod_id, kolicina, aktivna INTO pro, kol, akt
    FROM narudzbenica
    WHERE narudzbenica_id = narudz_id;
```

# Zadatak

```
IF akt IS NULL OR akt='N' THEN
    SET status = 'Ne postoji navedena narudzbenica!';
ELSE
    UPDATE proizvod SET stanje = stanje + kol WHERE proizvod_id = pro;
    UPDATE narudzbenica SET aktivna = 'N' WHERE narudzbenica_id = narudz_id;
    SET status = 'Proizvodi zaprimljeni!';
END IF;
END
```

- ne možemo dva puta zaprimiti proizvode iz iste narudžbenice – zašto?
- kada nam se dostave proizvodi iz narudžbenice dohvaćamo vrstu i količinu proizvoda te ažuriramo stanje na skladištu – kako?

# Zadatak

- korišćenje procedure za zaprimanje proizvoda

```
CALL zaprimanje_proizvoda(2, @status);  
SELECT @status;
```

```
+-----+  
| @status |  
+-----+  
| Ne postoji navedena narudzbenica! |  
+-----+
```

```
CALL zaprimanje_proizvoda(1, @status);  
SELECT @status;
```

```
+-----+  
| @status |  
+-----+  
| Proizvodi zaprimljeni! |  
+-----+
```

```
CALL zaprimanje_proizvoda(1, @status);  
SELECT @status;
```

```
+-----+  
| @status |  
+-----+  
| Ne postoji navedena narudzbenica! |  
+-----+
```

# Zadatak

- tablice

## *proizvod*

proizvod_id	sifra	naziv	stanje	cijena
1	P100	Racunalo Asus	10	3500.00
2	P101	Tipkovnica	10	200.00

## *racun NATURAL JOIN racun\_stavka*

racun_id	datum	stavke_id	proizvod_id	kolicina
1	2019-11-13 11:47:33	1	1	5
1	2019-11-13 11:47:33	2	2	3
2	2019-11-13 11:47:33	3	1	100

## *narudzbenica*

narudzbenica_id	proizvod_id	kolicina	datum_vrijeme	aktivna
1	2	10	2019-11-13 12:07:51	N

# Zadatak

- prodaja proizvoda – račun je već napravljen
  - smanjuje se stanje na zalihama
  - ukoliko stanje padne ispod 5 proizvoda automatski se izdaje nova narudžbenica

```
CREATE PROCEDURE obradi_racun(IN rac_id INT, OUT status VARCHAR(200))
BEGIN
    DECLARE pro INT DEFAULT NULL;
    DECLARE pro_sif VARCHAR(6) DEFAULT NULL;
    DECLARE sta INT DEFAULT NULL;
    DECLARE n_sta INT DEFAULT NULL;
    ...

```

# Zadatak

```
DECLARE stanje_nedostatno CONDITION FOR SQLSTATE '45000';
```

```
DECLARE done INT DEFAULT 0;
```

```
DECLARE cur CURSOR FOR
```

```
    SELECT p.proizvod_id, p.sifra, p.stanje,  
           p.stanje-rs.kolicina AS 'Novo stanje'
```

```
    FROM proizvod p NATURAL JOIN racun r
```

```
    NATURAL JOIN racun_stavka rs
```

```
    WHERE r.racun_id = rac_id;
```

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
```

```
    SET done = 1;
```

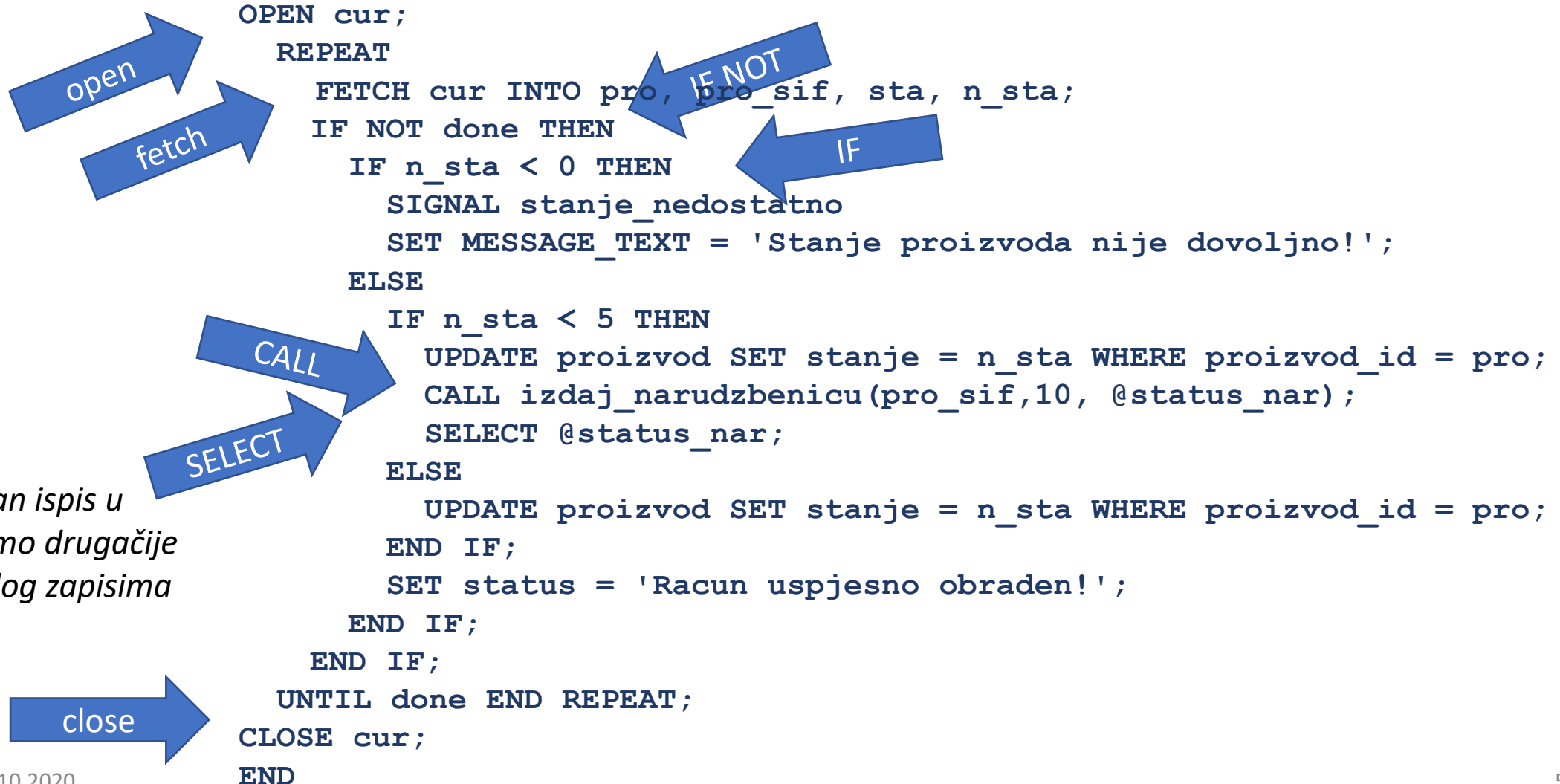
```
...
```

unhandled user-  
defined exception

zašto kursor?

zero rows fetched

# Zadatak



*jednostavan ispis u  
pravilu ćemo drugačije  
upravljati log zapisima*



# Zadatak

- obrada zaliha nakon izdavanja računa

*proizvod*

proizvod_id	sifra	naziv	stanje	cijena
1	P100	Racunalo Asus	5	3500.00
2	P101	Tipkovnica	7	200.00

*racun NATURAL JOIN racun\_stavka*

racun_id	datum	stavke_id	proizvod_id	kolicina
1	2019-11-13 11:47:33	1	1	5
1	2019-11-13 11:47:33	2	2	3
2	2019-11-13 11:47:33	3	1	100

*narudzbenica*

narudzbenica_id	proizvod_id	kolicina	datum_vrijeme	aktivna
1	2	10	2019-11-13 12:07:51	N

```
CALL obradi_racun(1,@status);
```

```
SELECT @status;
```

```
+-----+
| @status |
+-----+
| Racun uspjesno obraden! |
+-----+
```

# Zadatak

- ako ponovno pozovemo obradu računa (*nije dobro ponašanje*)

*proizvod*

proizvod_id	sifra	naziv	stanje	cijena
1	P100	Racunalo Asus	0	3500.00
2	P101	Tipkovnica	4	200.00

*racun NATURAL JOIN racun\_stavka*

racun_id	datum	stavke_id	proizvod_id	kolicina
1	2019-11-13 11:47:33	1	1	5
1	2019-11-13 11:47:33	2	2	3
2	2019-11-13 11:47:33	3	1	100

*narudzbenica*

narudzbenica_id	proizvod_id	kolicina	datum_vrijeme	aktivna
1	2	10	2019-11-13 12:07:51	N
2	1	10	2019-11-13 13:01:43	D
3	2	10	2019-11-13 13:01:43	D

```
CALL obradi_racun(1,@status);  
SELECT @status;
```

```
+-----+  
| @status_nar |  
+-----+  
| Narudzbenica stvorena |  
+-----+
```

```
+-----+  
| @status_nar |  
+-----+  
| Narudzbenica stvorena |  
+-----+
```

```
+-----+  
| @status |  
+-----+  
| Racun uspjesno obraden! |  
+-----+
```

# Zadatak

- ako ponovno pozovemo obradu računa (*još jednom*)

```
CALL obradi_racun(1,@status);
```

```
SELECT @status;
```

```
ERROR 1644 (45000): Stanje proizvoda nije dovoljno!
```

# Zadatak - zaključak

- kreirane su tri pohranjene procedure
  - poziv procedure iz procedure
- jednostavan primjer obrade stanja na skladištu u nekoj trgovini koristeći pohranjene procedure
  - proces nije dorađen do kraja
  - bolje upravljanje greškama
- obrada računa kada jedan proizvod nema dovoljno stanje?
  - drugi proizvodi će biti umanjeni na skladištu?
  - željeno ponašanje ili problem?
  - transakcije, slijede u nastavku

# Literatura

- Pročitati
  - [DSC] poglavlje 5.2.
  - Caltech CS121 - 9
- Slijedeće predavanje
  - [DSC] poglavlje 4.6.
  - [DSC] poglavlje 5.3.
  - Caltech CS121 - 10