

Napredne baze podataka

Procedure i funkcije



•Sadržaj predavanja

- Pohranjeni zadaci
- Procedure
- Funkcije

Pohranjeni zadaci

• Pohranjeni zadaci

- Koliko bi bilo jednostavnije napraviti nekoliko SQL naredbi i izvršiti ih za redom kao jednu naredbu?
- Promatramo ih kao korisnički definirane funkcije
- Pohranjeni na serveru za kasnije izvođenje
- Proširuju se mogućnosti SQL jezika uporabom:
 - lokalnih i globalnih varijabli
 - naredbi za kontrolu toka programa
 - naredbi za rukovanje iznimkama

• Zašto pohranjeni zadaci?

- Izbjegavanje repetitivnog pisanja SQL naredbi
 - Jednostavnije je napraviti nekoliko SQL naredbi i izvršiti ih za redom kao jednu naredbu
- Manje opterećivanje prijenosnog kanala
 - Umjesto da šalje niz SQL naredbi, klijent serveru šalje ime zadatka i argumente
- Izbjegavanje nedosljednosti u bazi
 - Dijeljenje logike s drugim aplikacijama – enkapsulacijom podataka osiguravamo dosljednost između aplikacija
 - Izbjegavanje pogrešaka prilikom pisanja upita
- Događivanje osnovne funkcionalnosti baze podataka
 - Aritmetika, varijable, petlje

• Podjela pohranjenih zadataka

- Pohranjene rutine (*stored routines*)
 - Pohranjene procedure (*stored procedures*)
 - Pozivaju se naredbom CALL
 - Ne vraćaju rezultat direktno (pomoću naredbe RETURN)
 - Mogu vratiti rezultat preko parametara (varijabli)
 - Pohranjene funkcije (*stored functions*)
 - Izvršavaju funkciju i vraćaju rezultat pomoću naredbe RETURN
- Okidači (*triggers*)
- Događaji (*events*)

•Prednosti

- **Bolje performanse**

- DBMS ne mora ponavljati prevođenje, validaciju i optimizaciju SQL upita koji se nalaze u SP (stored procedure)
 - Ponovnu optimizaciju SQL upita koji se koriste u SP sustav obavlja automatski, onda kad je to potrebno, npr. ako je nad nekom od relacija koje se koriste u SQL upitu naknadno kreiran indeks i slično
- Omogućena je uporaba klijent-poslužitelj arhitekture oslonjene na poslužitelj:
 - Reducira potrebne performanse mreže jer se prenosi samo rezultat
 - Razna sučelja, aplikacije napisane u npr. Java, PHP, VB, Excel, ... pozivaju pomoću npr. JDBC, JDBC/ODBC ili ODBC programskih sučelja iste SPL procedure

•Prednosti

- **Sigurnost i izolacija podataka** – dodjeljivanje dozvole (grant) na razini pohranjenog zadatka
 - **SQL omogućuje zaštitu podataka od neovlaštene uporabe na razini objekata** (relacije, atributi, pogledi-view)
 - korisniku se može ograničiti pristup do pojedinih objekata i vrsta operacije koju nad tim objektima može obaviti (brisanje, izmjena, unos, dohvat)
 - nije moguće ograničiti način na koji će korisnik obavljati operacije za koje je dobio dozvolu
 - **SPL omogućava zaštitu podataka od neovlaštene uporabe na razini pohranjenih zadataka**
 - korisniku se pridijeli dozvola za obavljanje definiranog pohranjenog zadatka, umjesto dozvole za pristup podacima
 - time je precizno određen način na koji korisnik smije obaviti operacije nad podacima

•Nedostatci

- Opterećuje DB server
 - CPU opterećenje i memorija
 - Umjesto dohvata za što su baze optimizirane, očekuje se rad sa logičkim operacijama i poslovnom logikom
- Limitiranost naredbi koje se mogu koristiti
 - jezik SQL je deklarativan, nije kompleksan kao C++, PHP, Java...
- Loš *error handling*
 - složenost za razvoj jer je potrebno znanje jezika
- Poslovna logika je u bazi umjesto u aplikaciji
- Dostatno odstupanje od SQL standarda među različitim tipovima baza kako bi se osigurala prednost te baze – teža migracija na drugi DBMS

Procedure

•Pohranjene procedure

- Pohranjena procedura ne može direktno vratiti rezultat
- Može primiti varijable koje mogu biti modificirane od strane procedure
- Procedure se definiraju u sljedećim koracima:
 1. Izradi proceduru (CREATE PROCEDURE)
 2. Definiraj ime procedure
 3. Definiraj ulaze i izlaze iz procedure
 4. Definiraj tijelo procedure (između BEGIN i END ako se sastoji od više naredbi)
- Ne smije imati ime rezervirano za neku ugrađenu funkciju (ne može se *override-ati*)

• Procedure - primjer

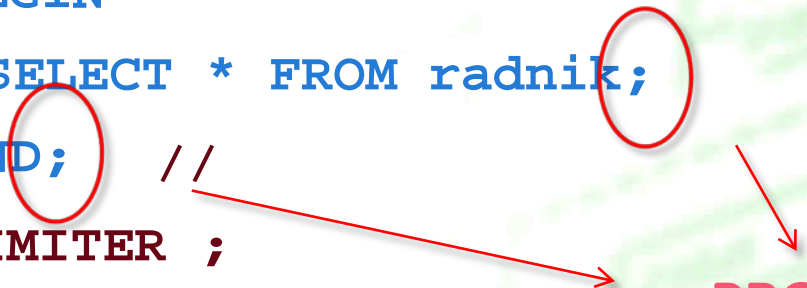
- Napisati proceduru koja će ispisati sve podatke o svim radnicima.

```
DELIMITER //
```

```
CREATE PROCEDURE ispisiRadnike()  
BEGIN  
    SELECT * FROM radnik;  
END; //
```

```
DELIMITER ;
```

PROBLEM!



- Ovime je procedura kreirana (ne i pozvana!)
- Procedura će direktno vratiti set rezultata

• **DELIMITER (graničnik)**

- Procedura se mora izvesti u cijelosti, a ne naredba po naredba
- Promijeniti pretpostavljenu vrijednost graničnika (;) na neku drugu vrijednost
- Može:
 - //
 - #
 - Pero
- Ne preporučuju se oznake koje bi se mogle koristiti u kodu procedure
 - *
 - (,)
 - =, >, <

•Poziv i brisanje procedure

- Procedure se pozivaju koristeći ključnu riječ CALL:

- **CALL ime_procedure()**

- Poziv procedure iz prethodnog primjera

- **CALL ispisiRadnike()**

- Brisanje procedure pomoću naredbe **DROP**

- **DROP PROCEDURE**

- **[IF EXISTS] ime_procedure**

- Brisanje procedure iz prethodnog primjera

- **DROP PROCEDURE ispisiRadnike**

• Provjera procedure

- SP je pohranjena kao objekt u rječniku baze podataka
 - izvorni kôd procedure
 - "izvršni" kôd - *interpreter instruction code* ili *p-code*:
 - unaprijed preveden i optimiran kôd procedure
- **SHOW PROCEDURE STATUS LIKE 'imeProcedure';**
- **SHOW CREATE PROCEDURE imeProcedure;**
- Pregled postojećih procedura
 - Tablica *routines* u bazi *information_schema*
 - INFORMATION_SCHEMA.ROUTINES
- Lokacija pohrane procedure
 - Tablica *proc* u bazi *mysql*
 - MYSQL.PROC

• Varijable u SQL-u

- Sve varijable u MySQL-u počinju znakom @ i vrijede u jednoj sesiji (spoju na poslužitelj) -> **GLOBALNE VARIJABLE**
- Iznimke su varijable deklarirane unutar pohranjenog zadatka (bloka BEGIN i END) koje ne moraju početi sa znakom @ -> **LOKALNE VARIJABLE**
- Ako se deklariraju bez znaka @, bit će vidljive samo u tom bloku
- Deklaracije moraju biti uvijek na vrhu procedure, poslije BEGIN i obavezno prije početka nekoga drugoga koda
- Dodjela vrijednosti varijabli
 - `SET @ime_varijable=vrijednost_varijable;`
Ili
 - `SET @ime_varijable:=vrijednost_varijable;`

•MySQL varijable - deklaracija

- Deklaracija varijabli

- **DECLARE ime_varijable tip(veličina) [DEFAULT originalna_vrijednost];**

- DECLARE br INT DEFAULT 0
- DECLARE x,y INT DEFAULT 0

- Dodjela vrijednosti varijabli kod deklaracije

- **DECLARE ime tip(veličina) DEFAULT vrijednost SET ime=nova_vrijednost;**

- DECLARE total INT DEFAULT 0 SET total=10;
- DECLARE ime tip(veličina) DEFAULT vrijednost;
SELECT ... **INTO** ime FROM ...;
- DECLARE n INT DEFAULT 0;
SELECT COUNT(sifKlijent) INTO n FROM klijent;

- Može i ovako

- DECLARE n INT DEFAULT 0;
- SET n=10;

•Komunikacija preko parametara

- Definirane vrste varijabli (parametara)
 - **IN**
 - Koristi se samo kao ulaz u proceduru
 - Procedura ne može mijenjati vrijednost ove varijable
 - Ako se ne navede tip ulaza, podrazumijeva se IN
 - **OUT**
 - Koristi se samo kao izlaz iz procedure
 - Procedura može mijenjati vrijednost ove varijable
 - **INOUT**
 - Koristi se kao izlaz i ulaz u proceduru
- Parametri se navode prilikom definicije procedure (u zagradi, odvojeni zarezom)
- **CREATE PROCEDURE imeProcedure ([IN] x tipPodatka, INOUT y tipPodatka, OUT z tipPodatka) ...tijelo procedure...**

• Procedure – primjer (IN)

- Napisati proceduru koja će ispisati podatke o svim radnicima u zadanom odjelu. Odjel je potrebno zadati prema nazivu.

Pretpostavka: nazivi
odjela su jedinstveni

```
DELIMITER //
```

```
CREATE PROCEDURE dohvatiRadnikePremaOdjelu
```

```
(IN naziv VARCHAR(50))
```

```
BEGIN
```

```
SELECT * FROM radnik NATURAL JOIN odjel WHERE  
nazivOdjel=naziv;
```

```
END //
```

```
DELIMITER ;
```

NAVESTI TOČNE ARGUMENTE
U SKLADU S DEFINICIJOM
PROCEDURE!

- Parametar *naziv* ulaznog je tipa jer preko njega proceduri predajemo vrijednost

```
CALL dohvatiRadnikePremaOdjelu('Balansiranje guma');
```

Što ako je ime varijable isto kao i ime atributa?

• Procedure – primjer (OUT)

- Napisati proceduru koja će vratiti broj radnika u zadanom odjelu.

```
DELIMITER //
```

```
CREATE PROCEDURE prebrojiRadnikeUOdjelu(IN naziv VARCHAR(50), OUT  
    broj INT)  
    BEGIN  
        SELECT COUNT(*) INTO broj FROM radnik NATURAL JOIN odjel WHERE  
            nazivOdjel=naziv;  
    END //  
DELIMITER ;
```

- Parametar *broj* izlaznog je tipa jer kada zovemo proceduru, uvijek očekujemo da nam se taj podatak vrati
- Poziv procedure -> rezultat pohranjujemo u varijablu @n -> ispis varijable pomoću SELECT naredbe

```
CALL prebrojiRadnikeUOdjelu  
    ('Balansiranje guma',@n);  
SELECT (@n);
```

• Procedure – primjer (INOUT)

- Napisati proceduru koja prima naziv mjesta i vraća naziv županije u kojoj se to mjesto nalazi.

```
DELIMITER //
```

```
CREATE PROCEDURE dohvatiZupaniju (INOUT naziv VARCHAR(255))
```

```
    BEGIN
```

```
        SELECT nazivZupanija INTO naziv FROM mjesto
```

```
            NATURAL JOIN zupanija WHERE nazivMjesto=naziv;
```

```
    END; //
```

```
DELIMITER ;
```

- Parametar *naziv* je ulazno-izlaznoga tipa. Zadajemo jednu vrijednost, a od procedure očekujemo da nam vrati neku drugu vrijednost.

```
SET @k='Bjelovar';
```

```
CALL dohvatiZupaniju(@k);
```

```
SELECT @k;
```

• Procedure - primjer

- Pohranjeni zadaci ne moraju nužno služiti za rad s podacima u bazi
- Napisati proceduru za izračun površine kruga. Napisati poziv procedure.

```
DELIMITER //
```

```
CREATE PROCEDURE površina (IN r DOUBLE, OUT a DOUBLE)
```

```
BEGIN
```

```
SET a = r * r * PI();
```

```
END //
```

```
DELIMITER ;
```

```
CALL površina(22, @a);
```

```
SELECT @a;
```

•Zadatak

- Napisati proceduru koja vraća JMBG klijenta koji je posljednji unesen u bazu. (Pretpostaviti da je posljednji onaj s najvećom vrijednosti atributa sifKlijent.)

```
DROP PROCEDURE IF EXISTS zadnjiKlijent;  
DELIMITER //  
CREATE PROCEDURE zadnjiKlijent(OUT j1 VARCHAR(13))  
BEGIN  
    DECLARE sif INT;  
    SELECT MAX(sifklijent) INTO sif FROM klijent;  
    SELECT klijent.jmbgKlijent INTO j1 FROM klijent  
        WHERE klijent.sifklijent=sif;  
END; //  
DELIMITER ;
```

- Parametar *j1* izlaznog je tipa jer kada zovemo proceduru uvijek očekujemo da nam se taj podatak vrati

```
CALL zadnjiKlijent(@1);  
SELECT @1;
```


Funkcije

•Pohranjene funkcije

- Pohranjena funkcija direktno vraća rezultat
- Ne može vratiti set rezultata
 - Paziti da funkcija ne vraća skup n-torki kao rezultat SELECT naredbe
- Varijable koje mogu biti modificirane
- Funkcija može samo primiti, ali ne i vratiti

•Definiranje funkcije

1. Izradi funkciju (CREATE FUNCTION)
2. Definiraj ime funkcije
3. U zagradi slijede parametri koji ulaze u funkciju
Svi su parametri ulaznog tipa (drugi tipovi nisu dozvoljeni)!
4. Slijedi ključna riječ **RETURNS** i **tip parametra koji funkcija vraća**
5. Prije tijela funkcije slijedi karakteristika
DETERMINISTIC
ili
NOT DETERMINISTIC (u ovom slučaju potrebno je navesti dodatne informacije o funkciji – *vidi idući slide*)
6. Definiraj tijelo funkcije (između BEGIN i END)
7. Ključna riječ **RETURN** koja vraća varijablu kao podatak iz funkcije

•Definiranje funkcije - karakteristika

- *characteristic:*

COMMENT '*string*'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL

| NO SQL

| READS SQL DATA

| MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

Vidi: <http://dev.mysql.com/doc/refman/5.7/en/create-procedure.html>

•Primjer funkcije

- Napisati funkciju za izračun površine kruga.

```
DELIMITER //  
CREATE FUNCTION površina(IN r double)  
  RETURNS double  
  DETERMINISTIC  
  BEGIN  
    DECLARE a double;  
    SET a = r * r * pi();  
    RETURN a;  
  END;  
//  
DELIMITER ;
```

NIJE POTREBNO NAVODITI!
ZAŠTO?

• Poziv i brisanje funkcije

- **SELECT ime_funkcije()**

- Nije potrebo koristiti CALL
- Poziv kao i kod ostalih ugrađenih funkcija
- Poziv funkcije iz prethodnog primjera:

```
SELECT površina(10);
```

- **DROP FUNCTION [IF EXISTS]
ime_funkcije**

- Brisanje funkcije iz prethodnog primjera:

```
DROP FUNCTION površina;
```

•Deterministic vs. Not deterministic

- Deterministički = strogo određeno
 - Rezultat funkcije uvijek će biti isti za iste ulazne parametre
- Problem kod replikacije
 - Prenosi se naredba, a ne podaci koje generira funkcija
 - Ako je funkcija not deterministic i primjerice generira nasumične brojeve, kod prijenosa u repliciranu bazu dobile bi se različite vrijednosti -> NEKONZISTENTNOST PODATAKA!

•Deterministic vs. Not deterministic

- Kod promjene lozinke zadajemo:

```
UPDATE tablica
```

```
SET
```

```
lozinka=ROUND(RAND()*1000)
```

```
WHERE korisnk='Marko';
```

- Kod repliciranja prenosi se SQL naredba, a ne sadržaj tablice!
- Not deterministic -> ne možemo predvidjeti što će se pojaviti na slave poslužitelju
- Deterministic -> “čudom” će se SQL engine pobrinuti da vrijednosti budu iste na oba poslužitelja

Podaci prije:

MASTER poslužitelj

pero	1234
marko	2323

Slave poslužitelj

pero	1234
marko	2323

Bez deterministic

MASTER poslužitelj

pero	1234
marko	3456

Slave poslužitelj

pero	1234
marko	?

Sa deterministic

MASTER poslužitelj

pero	1234
marko	3456

Slave poslužitelj

pero	1234
marko	3456

•Provjera funkcije

- `SHOW FUNCTION STATUS LIKE 'imeFunkcije';`
- `SHOW CREATE FUNCTION imeFunkcije;`
- Pregled postojećih funkcija
 - Tablica *routines* u bazi *information_schema*
 - `INFORMATION_SCHEMA.ROUTINES`
- Lokacija pohrane funkcije
 - Tablica *proc* u bazi *mysql*
 - `MYSQL.PROC`

•Funkcije & varijable

- Funkcija može osim u lokalne, također čitati iz i pisati u varijable definirane u sesiji (globalne)

```
DELIMITER //  
CREATE FUNCTION površinaVar() RETURNS INT  
    DETERMINISTIC  
    BEGIN  
        SET @pov= PI() * @radius * @radius;  
        RETURN NULL;  
    END//  
DELIMITER ;
```

Poziv funkcije:

```
SET @radius=2;  
SELECT površinaVar();  
SELECT @ pov;
```

•Primjer (pisanje u bazu)

- Funkcije (kao i procedure) mogu pisati u i brisati iz baze
- Napisati funkciju koja će sve radnike iz zadanog odjela 'promaknuti' u viši odjel. Funkcija neka vraća vrijednost 1.(Pretpostavka je da je viši odjel onaj koji ima za jedan veću vrijednost atributa sifOdjel.)

```
DROP FUNCTION promakni;  
DELIMITER //  
CREATE FUNCTION promakni(odj INT) RETURNS INT  
DETERMINISTIC  
BEGIN  
    UPDATE radnik SET sifodjel=sifodjel+1  
        WHERE sifodjel=odj;  
    RETURN 1;  
END//  
DELIMITER ;  
SELECT promakni(1);
```


•Primjer

- Napisati funkciju koja će za zadanog radnika ispisati koliko klijenata živi u mjestu u kojem živi i radnik.

```
DROP FUNCTION IF EXISTS brojKlijente;  
DELIMITER //  
CREATE FUNCTION brojKlijente(zadaniRadnik INT) RETURNS INT  
DETERMINISTIC  
BEGIN  
    DECLARE br INT DEFAULT NULL;  
    SELECT COUNT(sifKlijent) INTO br  
        FROM mjesto JOIN klijent ON pbrKlijent=pbrMjesto  
        NATURAL JOIN nalog  
        NATURAL JOIN radnik  
        WHERE pbrklijent=pbrstan  
            AND radnik.sifradnik=zadaniRadnik;  
    RETURN br;  
END//  
DELIMITER ;  
  
SELECT brojKlijente(277);
```

•Primjer

- Modificirati tablicu radnik tako da dodate novi atribut najvecaPlaca odgovarajućeg tipa. Napisati proceduru koja će: primiti šifru radnika -> za tog radnika pronaći županiju stanovanja -> u novostvoreni atribut najvecaPlaca unijeti koliki je najveći iznos plaće koju ima bilo koji radnik iz dohvaćene županije (županije stanovanja unesenog radnika). Iznos plaće potrebno je računati kao umnožak vrijednosti atributa koefPlace i iznosOsnovice

```
ALTER TABLE radnik ADD najvecaPlaca INT(11);
DELIMITER %%
DROP PROCEDURE IF EXISTS upisiMaksPlacu %%
CREATE PROCEDURE upisiMaksPlacu (IN rad INT)
BEGIN
    DECLARE zup INT DEFAULT NULL;
    DECLARE placa DECIMAL(6,2) DEFAULT NULL;
    SELECT sifZupanija INTO zup FROM radnik JOIN mjesto ON pbrStan=pbrMjesto
    WHERE sifRadnik=rad;
    SELECT MAX(koefPlaca*iznosOsnovice) INTO placa FROM radnik JOIN mjesto ON
    pbrStan=pbrMjesto WHERE mjesto.sifZupanija=zup;
    UPDATE radnik SET najvecaPlaca=placa WHERE sifRadnik=rad;
END;
%%
DELIMITER ;
Poziv PROCEDURE:
CALL upisiMaksPlacu(518);
Provjera rezultata:
SELECT * FROM radnik WHERE sifRadnik=518;
```

•Za kraj

- Funkcija ne smije imati ime kao neka već ugrađena funkcija u MySQL-u
- Nakon što proceduru/funkciju pohranimo, više ne možemo mijenjati njezino tijelo
 - `DROP PROCEDURE/FUNCTION [IF EXISTS] ime;`
- Pohranjene rutine mogu se backupirati zajedno s bazom