

Napredne baze podataka

Kontrola toka

Kursori



• Sadržaj predavanja

- Kontrola toka
- Kursori
- Conditions & handlers
- Privremene tablice

• Kontrola toka

IF uvjetovanje

IF uvjet THEN naredbe

[ELSEIF uvjet1 THEN naredbe];

[ELSE naredbe];

END IF;

- Moguće je ugnježđivanje IF unutar IF
- Vrijednost NULL nije ni TRUE niti FALSE

```
IF suma<0 THEN
    DELETE FROM...;
ELSEIF suma=0 THEN
    SELECT FROM...;
ELSE
    INSERT INTO...;
END IF;
```


•Kontrola toka

CASE uvjetovanje

CASE

WHEN expression_1 THEN commands_1

...

WHEN expression_n THEN commands_n

ELSE commands

END CASE

CASE expr

WHEN val_1 THEN commands_1

...

WHEN val_n-1 THEN commands_n-1

ELSE val_n

END CASE

• Kontrola toka

CASE uvjetovanje - primjer

- Napisati funkciju koja će dohvatiti trenutni datum s poslužitelja i ispisati koji je dan u tjednu.

```
DELIMITER //
```

```
CREATE FUNCTION danUTjednu() RETURNS VARCHAR(50)
```

```
DETERMINISTIC
```

```
    BEGIN
```

```
        DECLARE vrati VARCHAR(50);
```

```
        CASE DAYOFWEEK(CURDATE())
```

```
            WHEN 2 THEN SET vrati='Danas je ponedjeljak';
```

```
            WHEN 3 THEN SET vrati='Danas je utorak';
```

```
            WHEN 4 THEN SET vrati='Danas je srijeda';
```

```
            WHEN 5 THEN SET vrati='Danas je četvrtak';
```

```
            WHEN 6 THEN SET vrati='Danas je petak';
```

```
            ELSE SET vrati='Danas je vikend';
```

```
        END CASE;
```

```
        RETURN vrati;
```

```
    END; //
```

```
DELIMITER ;
```

```
SELECT danUTjednu();
```

•Petlje

WHILE petlja

WHILE uvjet **DO**

naredbe

END WHILE

```
DELIMITER $$
CREATE PROCEDURE proc()
BEGIN
    DECLARE var INT;
    SET var=1;
    WHILE var<=5 DO
        SET var=var+1;
        SELECT var;
    END WHILE;
END;
$$
DELIMITER ;
CALL proc();
```

U petlji zbrajamo $var=var+1$ sve dok var ne poprimi vrijednost 5.

Za svaku vrijednost var koju uvećamo ispišemo var (SELECT var;)

Rezultat:

2
3
4
5
6

•Petlje

REPEAT petlja

REPEAT

naredbe;

UNTIL uvjet

END REPEAT

```
DROP PROCEDURE proc;
DELIMITER $$
CREATE PROCEDURE proc()
BEGIN
    DECLARE var INT;
    SET var=1;
    REPEAT
        SET var=var+1;
        SELECT var;
        UNTIL var>5
    END REPEAT;
END;
$$
DELIMITER ;
CALL proc();
```

U petlji zbrajamo $var=var+1$ sve dok var ne poprimi vrijednost 5.

Za svaku vrijednost var koju uvećamo, ispišemo ga (SELECT var;)

Rezultat:

2
3
4
5
6

• Petlje

LOOP (LEAVE, ITERATE)

```
Ime_petlje: LOOP
    naredbe
    IF uvjet THEN
        LEAVE ime_petlje;
    END IF;
    IF uvjet THEN
        ITERATE ime_petlje;
    END IF;
END LOOP;
```

- LEAVE (kao break u C)
 - izlazi iz petlje ako je zadovoljen uvjet
- ITERATE (kao continue u C)
 - pokreće novu iteraciju petlje ako je zadovoljen uvjet

•Petlje

LOOP (LEAVE, ITERATE)

```
DROP PROCEDURE proc;
DELIMITER $$
CREATE PROCEDURE proc()
    BEGIN
        DECLARE var INT;
        SET var=1;
        petlja:LOOP
            SET var=var+1;
            SELECT var;
            IF var>5 THEN
                LEAVE petlja;
            END IF;
        END LOOP;
    END; $$
DELIMITER ;

CALL proc();
```

U petlji zbrajamo $var=var+1$ sve dok var ne poprimi vrijednost 5.

Za svaku vrijednost var koju uvećamo, ispišemo ga (SELECT var;)

Rezultat:

2
3
4
5
6

•Primjer

- Napisati proceduru koja će ispisati prvih n prirodnih brojeva u jednoj varijabli. Brojevi moraju biti razdvojeni zarezom. (Zanemarite ako se zarez ispisuje i nakon zadnjeg broja.)

```
DELIMITER $$
DROP PROCEDURE IF EXISTS WhileLoopProc$$
CREATE PROCEDURE WhileLoopProc(IN n INT)
    BEGIN
        DECLARE var INT DEFAULT NULL;
        DECLARE str VARCHAR(255) DEFAULT NULL;
        SET var=1;
        SET str='';
        WHILE var<=n DO
            SET str=CONCAT(str,var,',');
            SET var=var + 1;
        END WHILE;
        SELECT str;
    END $$
DELIMITER ;
CALL WhileLoopProc(8);
```

•Primjer

- Napisati proceduru koja za zadani odjel broji radnike koji pripadaju tom odjelu. Ako odjel ima više od 10 radnika, procedura mora vratiti -1, a ako odjel nema radnika, procedura mora vratiti 0. U ostalim slučajevima, procedura vraća stvarni broj radnika. Napisati poziv procedure za odjel 100005. Napisati poziv procedure za odjel 5.

```
DELIMITER //
```

```
CREATE PROCEDURE odjel_rad(IN zadaniOdjel INT,OUT broj INT)
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO broj FROM radnik WHERE sifOdjel=zadaniOdjel;
```

```
    IF broj>10 THEN
```

```
        SET broj=-1;
```

```
    ELSEIF broj=0 THEN
```

```
        SET broj=0;
```

```
    END IF;
```

```
END;
```

```
//
```

```
DELIMITER ;
```



```
CALL odjel_rad(100005,@a);
```

```
SELECT @a;
```

• Primjer

- Napisati funkciju za unos novog odjela u tablicu *odjel*. Funkcija treba **provjeriti postoji li već odjel sa zadanim imenom**. Ako postoji, završiti s radom (vrati 0). Ako ne postoji, **pridijeliti mu šifru** i unijeti u tablicu (vrati 1). (Pretpostaviti da na šifri odjela ne postoji *autoincrement*.)

```
DELIMITER //
```

```
CREATE FUNCTION unesiOdjel(noviOdjel VARCHAR(50)) RETURNS INT
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE broj, vrati, zadnji INT DEFAULT NULL;
```

```
    SELECT COUNT(*) INTO broj FROM odjel WHERE nazivOdjel=noviOdjel;
```

```
    IF broj=0 THEN
```

```
        SELECT MAX(sifOdjel) INTO zadnji FROM odjel;
```

```
        INSERT INTO odjel (sifOdjel, nazivOdjel) VALUES ((zadnji+1),noviOdjel);
```

```
        SET vrati=1;
```

```
    ELSE
```

```
        SET vrati=0;
```

```
    END IF;
```

```
    RETURN vrati;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
SELECT unesiOdjel('Odjel za reklamacije');
```


•Kursori - zašto?

- Do sada smo vidjeli sintaksu
`SELECT SUM(cijena) INTO cijena FROM....`
Ili
`SELECT naziv INTO naziv1 FROM...`
- S time da moramo naglasiti da je naziv1 sadržavao samo jednu vrijednost (jednu n-torku)
- Što ako želimo u proceduri/funkciji doći do svih n-torki rezultata SQL query-a?
- Odgovor su kursori!

•Kursori

- **Upravljanje sa setom podataka** dobivenih pomoću SELECT upita
- Uzastopno dohvaćanje n-torki iz dobivenog seta rezultata (prilikom svake iteracije petlje)
- Služi za procesiranje pojedinih n-torki koje je baza vratila kao rezultat upita
- Svojstva
 - Ne mogu se koristiti za dohvat prethodnog podatka
 - *Read-only* - kursor se ne može promijeniti
 - Koriste se isključivo za dohvat podataka
 - ne za unos niti za brisanje
 - kod UPDATEa bi moglo rezultirati neočekivanim rezultatom
 - Unutar transakcije, automatski se brišu pozivom naredbe COMMIT
- Kada ne koristiti kursori?

•Rad sa kursorom (1)

- Kursor mora biti deklariran nakon deklaracije svih varijabli, ali prije bilo kojega dijela koda!
- **DECLARE imeKursora CURSOR FOR select_izraz;**

- Npr

- DECLARE curl CURSOR FOR SELECT ime
FROM osoba NATURAL JOIN mjesto;
- DECLARE curl CURSOR FOR
SELECT ime, prezime FROM osoba;
- ~~DECLARE curl CURSOR FOR
SELECT * FROM osoba;~~
- Zato što kasnije neće biti moguće odrediti koji će se atribut odnositi na koju varijablu!

•Rad sa kursorom (2)

- **OPEN ime_kursora**

- Pokreće kursor i aktivira podatke za čitanje

- **FETCH ime_kursora INTO varijabla [,varijable...]**

- Dohvaća podatke iz SQL naredbe
 - Broj navedenih varijabli mora odgovarati broju atributa u SELECT dijelu naredbe!
 - Nakon svakoga uspješnoga dohvata automatski prelazi na sljedeću n-torku
 - U slučaju da nema sljedeće n-torke, prijavljuje grešku (Error: **1329** SQLSTATE: **02000** (**ER_SP_FETCH_NO_DATA**) ; Message: No data - zero rows fetched, selected, or processed)

- **CLOSE ime_kursora;**

- Zatvara kursor i oslobađa resurse
 - Poželjno upotrebljavati, makar engine garantira oslobađanje resursa pri završetku procedure/funkcije

•Kursor - primjer

- Napisati proceduru koja će dohvatiti svaki kvar zasebno i uz njegovo ime ispisati radi li se o velikom ili malom kvaru. Kriterij je iznos atributa *satiKvar*. Kvar se smatra velikim ako je *satiKvar* veći od 5.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS etiketiraj$$
CREATE PROCEDURE etiketiraj()
    BEGIN
        DECLARE naziv_vrijednost VARCHAR(255);
        DECLARE sati_vrijednost INT;
        DECLARE kur CURSOR FOR SELECT nazivKvar, satiKvar FROM kvar;
        OPEN kur;
        petlja:LOOP
            FETCH kur INTO naziv_vrijednost, sati_vrijednost;
            IF sati_vrijednost>5
                THEN SELECT naziv_vrijednost AS ime, 'veliki' AS velicina;
            ELSE
                SELECT naziv_vrijednost AS ime, 'mali' AS velicina;
            END IF;
        END LOOP petlja;
        CLOSE kur;
        SELECT 'kraj'; /*naredba dodana na kraju*/
    END; $$
DELIMITER ;
CALL etiketiraj();
```

Primijetiti: posljednji redak vraća obavijest o pogreški
“No data - zero rows fetched, selected, or processed”
Rješenje: prebrojati retke (COUNT) + REPEAT/WHILE
umjesto LOOP PETELJE
Ili
CONDITIONS & HANDLERS

• Conditions & Handlers

- Služe da bismo preusmjerili tok koda u trenutku pojave nekog uvjeta – **CONDITION**
- Pokrivaju što se događa u trenutku pojave više događaja tj. grešaka – **HANDLER**
 - **TIP**
 - CONTINUE – nastavlja izvršavanje bloka bez prekida
 - EXIT – prekida se izvršavanje bloka u kojem se *handler* nalazi
 - Pretpostavljena akcija za sve pogreške neobuhvaćene handlerom
 - UNDO – bit će implementiran u nadolazećim verzijama
 - **UVJET**
 - može biti SQLSTATE vrijednost ili
 - mysql_error_code

• Conditions & Handlers - deklaracija

1. odrediti na koju se pogrešku odnosi:

- **DECLARE ime CONDITION FOR uvjet**

```
DECLARE err_table_exists CONDITION FOR 1050;  
DECLARE err_table_exists CONDITION FOR SQLSTATE '42S01';
```

2. odrediti *handler* za zadani tip pogreške (sadrži naredbe koje će se izvesti u slučaju nastanka definirane pogreške)

- **DECLARE tip HANDLER FOR uvjeti**

```
DECLARE EXIT HANDLER FOR err_table_exists  
BEGIN  
    SET @table=-1;  
END;
```

- **NAPOMENA:** moguće je definirati tip greške i unutar same deklaracije *handlera* (donji je kod ekvivalent gornjim koracima)

```
DECLARE EXIT HANDLER FOR 1050  
BEGIN  
    SET @table=-1;
```

•Uvjeti

- SQLSTATE vrijednost – string od 5 znamenaka
 - ANSI SQL standardiziran
 - Ne preporuča se koristiti pogreška '00000' tj '0' – kod za uspješno izvršavanje
 - SQLWARNING = SQLSTATE koji počinje sa '01'
 - NOT FOUND = SQLSTATE koji počinje sa '02'
 - SQLEXCEPTION = sve što ne počinje sa '00', '01' ili '02'
- MySQL error code – broj
 - Error codes:
<http://dev.mysql.com/doc/refman/5.7/en/error-messages-server.html>

•Conditions & Handlers

```
DELIMITER //
```

```
CREATE PROCEDURE handlerdemo()  
BEGIN  
    DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'  
        SET @x=1;  
    SET @x=1;  
    INSERT INTO test.t VALUES(1);  
    SET @x=2;  
    INSERT INTO test.t VALUES(1);  
    SET @x=3;  
END;  
//  
DELIMITER;
```

```
CALL handlerdemo();  
SELECT @x  
→ vraća 3
```

Ide na sljedeću naredbu

Definira slučaj
pojave greške
Duplicate key

Glatko prolazi

Uzrokuje grešku, ali nastavlja
sa izvođenjem – dokaz da
nastavlja

•Primjer

- Napisati proceduru koja će za zadani poštanski broj pronaći sve nazive mjesta u županiji u kojoj se nalazi zadano mjesto. Neka procedura ispiše nazive tih mjesta na ekran.

```
DELIMITER //
```

```
CREATE PROCEDURE pbr(ulaz INT)
```

```
BEGIN
```

```
    DECLARE p_zup INT;
```

```
    DECLARE naziv VARCHAR(50);
```

```
    DECLARE error INT DEFAULT 0;
```

```
    DECLARE kursor CURSOR FOR SELECT nazivMjesto
```

```
        FROM mjesto WHERE mjesto.sifZupanija=p_zup;
```

```
    DECLARE CONTINUE HANDLER FOR NOT FOUND
```

```
        SET error=1;
```

Uvjet je već kod deklaracije kursora

- Ili umjesto NOT FOUND napisati SQLSTATE '02000'

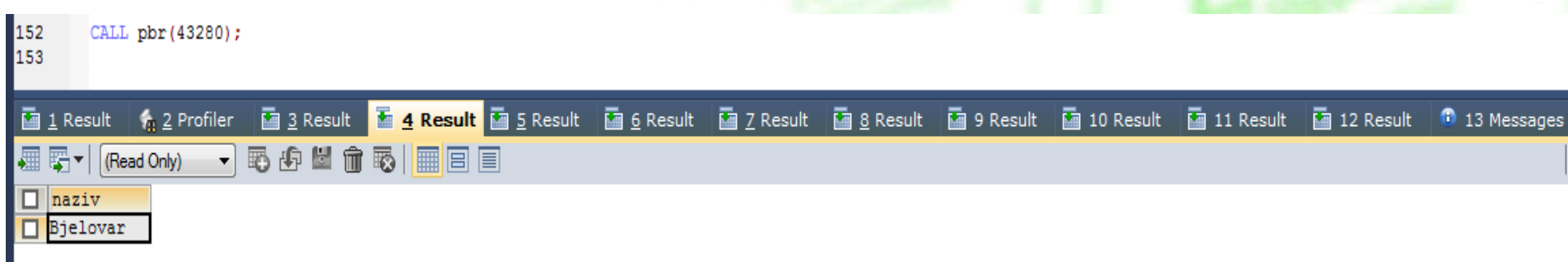
•Primjer (nastavak)

```
SELECT sifzupanija INTO p_zup FROM mjesto
      WHERE mjesto.pbrmjesto=ulaz;
OPEN kursor;
petlja: LOOP
      FETCH kursor INTO naziv;
      IF error=1 THEN
            LEAVE petlja;
      END IF;
      SELECT naziv;
END LOOP;
CLOSE kursor;
END;
//

CALL pbr(43280);
```

• Primjer (nastavak) – privremena tablica

- Navedeni program vratit će onoliko setova rezultata koliko je puta pozvan SELECT
 - Svaki SELECT zaseban je set rezultata
 - Takav ispis je nepregledan



- Pozor:
 - Nemamo mogućnost vratiti više n-torki u glavni program
 - Rješenje: snimit ćemo ih u **privremenu tablicu!**
 - Modificirat ćemo program

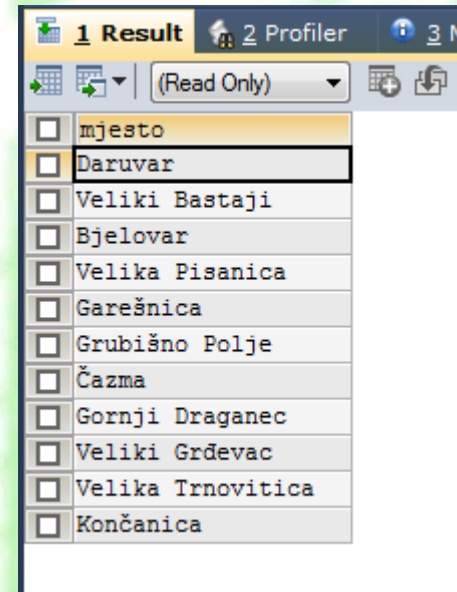
• Primjer (nastavak) – privremena tablica

- Napisati proceduru koja će za zadani poštanski broj pronaći sve nazive mjesta u županiji u kojoj se nalazi zadano mjesto. Neka procedura ispiše nazive tih mjesta na ekran.

```
DROP PROCEDURE pbr;  
  
DELIMITER //  
  
CREATE PROCEDURE pbr(ulaz INT)  
BEGIN  
  
    DECLARE p_zup INT;  
    DECLARE naziv VARCHAR(50);  
    DECLARE error INT DEFAULT 0;  
    DECLARE kursor CURSOR FOR SELECT nazivMjesto  
        FROM mjesto WHERE mjesto.sifZupanija=p_zup;  
    DECLARE CONTINUE HANDLER FOR NOT FOUND  
        SET error=1;
```

• Primjer (nastavak) – privremena tablica

```
DROP TEMPORARY TABLE IF EXISTS bbzup;  
CREATE TEMPORARY TABLE bbzup(mjesto VARCHAR(50));  
SELECT sifzupanija INTO p_zup FROM mjesto  
      WHERE mjesto.pbrmjesto=ulaz;  
OPEN kursor;  
petlja: LOOP  
      FETCH kursor INTO naziv;  
      IF error=1 THEN  
          LEAVE petlja;  
      END IF;  
      SELECT naziv; → INSERT INTO bbzup VALUES(naziv);  
END LOOP;  
SELECT * FROM bbzup;  
CLOSE kursor;  
END; //  
DELIMITER ;  
CALL pbr(43280);
```



<input type="checkbox"/>	mjesto
<input checked="" type="checkbox"/>	Daruvar
<input type="checkbox"/>	Veliki Bastaji
<input type="checkbox"/>	Bjelovar
<input type="checkbox"/>	Velika Pisanica
<input type="checkbox"/>	Garešnica
<input type="checkbox"/>	Grubišno Polje
<input type="checkbox"/>	Čazma
<input type="checkbox"/>	Gornji Draganec
<input type="checkbox"/>	Veliki Grđevac
<input type="checkbox"/>	Velika Trnovitica
<input type="checkbox"/>	Končanica

• Primjer – privremena tablica&funkcija

- Možemo li isto rješenje realizirati izradom funkcije? (Pazi: funkcija ne može ispisati sadržaj privremene tablice)

```
DROP FUNCTION IF EXISTS pbr;  
DELIMITER //  
CREATE FUNCTION pbr(ulaz INT) RETURNS INT DETERMINISTIC  
BEGIN  
    DECLARE p_zup INT;  
    DECLARE naziv VARCHAR(50);  
    DECLARE error INT DEFAULT 0;  
    DECLARE kursor CURSOR FOR SELECT nazivMjesto  
        FROM mjesto WHERE mjesto.sifZupanija=p_zup;  
    DECLARE CONTINUE HANDLER FOR NOT FOUND  
        SET error=1;
```

•Primjer – privremena tablica&funkcija

```
DROP TEMPORARY TABLE IF EXISTS bbzup;
CREATE TEMPORARY TABLE bbzup(mjesto VARCHAR(50));
SELECT sifzupanija INTO p_zup FROM mjesto WHERE mjesto.pbrmjesto=ulaz;
OPEN kursor;
petlja: LOOP
    FETCH kursor INTO naziv;
    IF error=1 THEN
        LEAVE petlja;
    END IF;
    INSERT INTO bbzup VALUES(naziv);
END LOOP;
CLOSE kursor;
RETURN 1;
END; //
```

```
DELIMITER ;
```



```
SELECT pbr(43280);
```


• Handleri – primjeri uporabe

- DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
SET I_error=1;
- DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
ROLLBACK;
SELECT 'Error occurred – terminating';
END;
- DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
SELECT 'Duplicate key in index';
- DECLARE CONTINUE HANDLER FOR NOT FOUND
SET I_done=1;
- DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
SET I_done=1;

•Zadatak

- Napisati proceduru koja će svim radnicima koji imaju koeficijent plaće manji od 1.00 povišati ga **ZA** 1.00. Ostalim radnicima čiji je koeficijent plaće veći od 2.00 smanjiti ga **ZA** 0.50. Procedura mora vratiti *broj n-torki koje je obradila, broj radnika kojima je plaća uvećana te broj radnika kojima je plaća smanjena*.

```
DROP PROCEDURE IF EXISTS korigiraj_koef;
DELIMITER //
CREATE PROCEDURE korigiraj_koef()
BEGIN
    DECLARE koef DECIMAL(3,2);
    DECLARE sif, dohvaceno, smanjena, povecana INT;
    DECLARE flag BOOL;
    DECLARE kursor CURSOR FOR SELECT sifRadnik, koefPlaca FROM radnik;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET flag=TRUE;
    SET flag=FALSE;
    SET smanjena=0;
    SET povecana=0;

    OPEN kursor;
    SELECT FOUND_ROWS() INTO dohvaceno;
```

•Zadatak (nastavak)

```
petlja:LOOP
    FETCH kursor INTO sif, koef;
    IF flag=TRUE THEN
        LEAVE petlja;
    END IF;
    IF koef<1.00 THEN
        SET koef=koef+1;
        UPDATE radnik SET koefPlaca=koef WHERE sifRadnik=sif;
        SET povecana=povecana+1;
    ELSEIF koef>=2.00 THEN
        SET koef=koef-0.5;
        UPDATE radnik SET koefPlaca=koef WHERE sifRadnik=sif;
        SET smanjena=smanjena+1;
    END IF;

END LOOP;
CLOSE kursor;
SELECT dohvaceno AS dohvaceno_rezultata,
       smanjena AS smanjena_placa, povecana AS povecana_placa;

END; //
DELIMITER ;
CALL korigiraj_koef();
```

The screenshot shows a database client window with four tabs: '1 Result', '2 Profiler', '3 Messages', and '4 Table Data'. The '1 Result' tab is active, displaying a table with three columns: 'dohvaceno_rezultata', 'smanjena_placa', and 'povecana_placa'. The first row of data shows values 98, 30, and 41 respectively. Below the table, there is a text area containing the command 'CALL korigiraj_koef()'. The interface also includes a 'Read-only result' dropdown and radio buttons for 'All rows' and 'Rows in a range'.

dohvaceno_rezultata	smanjena_placa	povecana_placa
98	30	41

CALL korigiraj_koef()

•Zadatak

- Napisati proceduru koja će primiti šifre dvaju radnika (rOduzmiID i rDodajID) te iznos. Prvom radniku je potrebno smanjiti iznosOsnovice za zadani iznos, a drugom ga uvećati.
- Potrebno je provjeriti da li prvi radnik ima iznosOsnovice veći od zadanog iznosa. Također, ne smije se dogoditi da je prvom iznos oduzet, a drugome nije dodan.

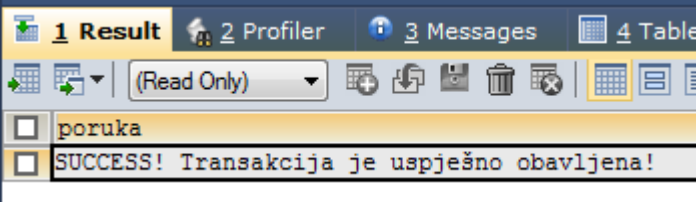
```
DROP PROCEDURE IF EXISTS obaviTransakciju ;  
DELIMITER &&  
CREATE PROCEDURE obaviTransakciju  
    (IN rOduzmiID INT,  
    IN rDodajID INT,  
    IN iznos DOUBLE,  
    OUT poruka VARCHAR(50) )
```


•Zadatak (nastavak)

```
BEGIN
  DECLARE rltrenutnoStanje DOUBLE(10,2) DEFAULT 0;
  BEGIN
    DECLARE EXIT HANDLER FOR NOT FOUND
      SET poruka='Radnik nije pronađen pod
zadanim brojem!';
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      BEGIN
        ROLLBACK;
        SET poruka='Greska!';
      END;
    SET autocommit=0;
    START TRANSACTION;
    SELECT iznosOsnovice INTO rltrenutnoStanje
      FROM radnik WHERE sifRadnik=rOduzmiID;
```

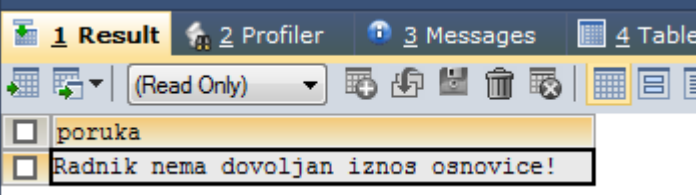
•Zadatak (nastavak)

```
/*'ako ima dovoljno', obavi transakciju*/  
IF rltrenutnoStanje>iznos THEN  
    UPDATE radnik SET iznosOsnovice=iznosOsnovice-iznos  
        WHERE sifRadnik=rOduzmiID;  
    UPDATE radnik SET iznosOsnovice=iznosOsnovice+iznos  
        WHERE sifRadnik=rDodajID;  
  
ELSE  
    SET poruka='Radnik nema dovoljan iznos osnovice!';  
END IF;  
END;  
IF poruka != '' THEN ROLLBACK;  
    ELSE COMMIT;  
    SET poruka='SUCCESS! Transakcija je uspješno obavljena!';  
END IF;  
SELECT poruka;  
SET autocommit=1;  
END &&  
DELIMITER ;  
  
CALL obaviTransakciju(518,514,1000,@a);  
CALL obaviTransakciju(518,514,5000,@a);
```



The screenshot shows a SQL client interface with a toolbar and tabs for '1 Result', '2 Profiler', '3 Messages', and '4 Tables'. The '1 Result' tab is active, displaying a table with two rows. The first row is labeled 'poruka' and the second row contains the text 'SUCCESS! Transakcija je uspješno obavljena!'. An arrow from the first CALL statement in the code points to this screenshot.

poruka
SUCCESS! Transakcija je uspješno obavljena!



The screenshot shows a SQL client interface with a toolbar and tabs for '1 Result', '2 Profiler', '3 Messages', and '4 Tables'. The '1 Result' tab is active, displaying a table with two rows. The first row is labeled 'poruka' and the second row contains the text 'Radnik nema dovoljan iznos osnovice!'. An arrow from the second CALL statement in the code points to this screenshot.

poruka
Radnik nema dovoljan iznos osnovice!