Funkcije

- U MySql-u postoje dve vrste funkcija:
  - **Ugrađene**
  - **Korisnički definisane**
- Ugrađene funkcije su sve one funkcije koje podrazumeva standardna forma MySQL servera, dok su korisnički definisane one koje su izgrađene naknadno, od strane korisnika
- Sve funkcije se generalno dele na **skalarne** i **agregatne**

Korisnički definisane funkcije

- Korisnički definisane funkcije mogu biti realizovane na dva načina: kroz SQL skriptu (DDL) ili kroz izvorni programski jezik MySQL servera (C).
- Kreiranje korisnički definisanih funkcija (UDF) kroz SQL je jednostavniji metod. Zapravo, sintaksa je veoma slična sintaksi za kreiranje uskladištenih procedura.

Kreiranje funkcije

- Funkcija se kreira na isti način kao i procedura, ali je neophodno navesti njen izlazni tip prilikom kreiranja
- Takođe se za funkcije duže od jednog reda koristi zamena delimitera

```
CREATE FUNCTION myFunction()
RETURNS varchar(20)
RETURN 'hello from UDF';
```

```
DELIMITER //
CREATE FUNCTION myFunction()
RETURNS varchar(20)
BEGIN
DECLARE x int;
set x = 10;
RETURN 'pozdrav';
END //
DELIMITER ;
```

Pozivanje funkcije

- Za razliku od procedure, funkciju je moguće umetnuti u sam upit, što nam daje na raspolaganje velike mogućnosti za intervenciju na podacima u trenutku kreiranja izlaza. Na primer, funkciju pozivamo njenim umetanjem u upit:

```
SELECT myFunction()
```

Parametrizacija funkcije

- Parametrizacija funkcije vrši se na isti način kao i parametrizacija uskladištene procedure

```
delimiter //
create function myFunction(p1 int, p2 int)
returns int
begin
declare p3 int;
set p3 = p1 + p2;
return p3;
end //
delimiter ;
```

```
SELECT myFunction(2,3)
```

Vežba

- Potrebno je napraviti upit koji će prikazati glumce, ali tako da, svaki put kada se pojavi ime glumca christian, bude napisano neko drugo ime

Rešenje

- Ovaj problem se može rešiti funkcijom

```
delimiter //
create function changeName(p1 varchar(50), p2 varchar(50),p3 varchar(50))
returns varchar(50)
begin
    if p1=p2 then return p3; end if;
    return p1;
end //
delimiter ;
```

Trigeri (okidači)

- Okidači su korisnički definisani blokovi koda koji se izvršavaju u trenutku intervencije na tabelama (insert, update i delete) i omogućavaju manipulisanje podacima pre nego što uistinu budu uneti, izmenjeni ili obrisani.
- Dele se na tri kategorije (**INSERT, UPDATE i DELETE**) i dve pod kategorije (**BEFORE i AFTER**) i u zavisnosti od tipa, nude određene opcije.
- Jedan okidač je objekat u jednoj bazi podataka i važi samo za tu bazu. Ukoliko se ta baza obriše, biće obrisani i svi njeni okidači.
- Kada kreiramo okidač, povezujemo ga sa određenom tabelom i akcijom koju želimo da pratimo. Pri tom, ne možemo uneti više istoimenih okidača u jednoj tabeli, niti postaviti iste tipove okidača na isti događaj u jednoj tabeli (na primer dva BEFORE INSERT okidača na jednoj tabeli)

Before triggers

- Karakteristika ovih okidača je da se događaju pre nego što podatak dođe do tabele. To nam omogućava da podatak presretnemo i izvršimo eventualnu intervenciju na njemu. U tu svrhu, MySQL u trenutku unosa kreira tabelu u memoriji, čija je struktura ista kao i ciljna tabela, s tom razlikom, što su jedini podaci ove tabele, u stvari, podaci koje unosimo ili menjamo:

```
create trigger checkName before insert on mytable
for each row set new.name = concat(new.name, "_test");
```

```
DELIMITER //
CREATE TRIGGER checkName BEFORE INSERT ON mytable
  FOR EACH ROW
    BEGIN
      if length(new.name) < 5 then
            set new.name = concat(new.name,"_test");
      END IF;
    END//
DELIMITER ;
```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT  = 'Ne moze';
```

Before triggers (update i delete)

- Update i delete trigeri funkcionišu isto kao i insert triger, samo se drugačije deklarišu

```
DELIMITER //
CREATE TRIGGER checkName BEFORE UPDATE ON mytable
 FOR EACH ROW
    BEGIN
      if length(old.name) < 4 then
            set new.name = password(new.name);
      END IF;
    END//
DELIMITER ;
```

After trigeri

- Ovi okidači aktiviraju se nakon unosa podataka. Znači da kada se nađemo u kodu okidača, imaćemo pristup samo već unetim podacima.
- *After Triggers* su dobri ukoliko želimo da paralelno sa glavnom tabelom ažuriramo i neku arhivsku, log ili bekap tabelu:

```
DELIMITER //
CREATE TRIGGER backup AFTER INSERT ON mytable
    FOR EACH ROW
        BEGIN
            insert into backup (name) values (new.name);
    END//
DELIMITER ;
```

After trigeri

- Kada jednom postavimo okidač na tabelu, onda je on deo te tabele u kontekstu njenih transakcija. To znači da se, prilikom ažuriranja podataka, uzima u obzir i izvršavanje okidača i da će se akcija smatrati neuspešnom ukoliko dođe do greške prilikom njegove aktivacije.
- Ukoliko dođe do greške prilikom aktivacije BEFORE okidača, neće doći ni do aktivacije AFTER okidača (ukoliko se nalaze na istoj naredbi iste tabele)

Kurzori

- Kurzori su posebni tipovi podataka koji omogućavaju sekvencijalno rukovanje podacima dobijenim određenim upitom. Iako kažemo tipovi podataka, rukovanje kurzorima nije baš jednostavno kao sa prostim tipovima. Oni imaju tabelarnu strukturu i zahtevaju specifičan set naredbi prilikom rukovanja.
- Drugim rečima kurzori se koriste kako bi se prošlo (iteriralo) kroz veliku kolekciju podataka. Pogledajmo na početku neke od osobenosti MySQL kurzora:
  - Nije moguće vršiti ažuriranje podataka korišćenjem kurzora, već samo čitanje
  - Kroz redove se može prolaziti samo u jednom smeru, i to tako kako je definisano SELECT upitom; nije moguće vršiti prolazak unazad ili slično
  - Kurzore je moguće koristiti samo unutar uskladištenih rutina (procedura i korisnički definisanih funkcija) i trigera

http://pastie.org/pastes/9993028/text

```
DELIMITER $$
CREATE PROCEDURE my_cursor()
BEGIN
 declare id int;
 declare fname varchar(50);
 DECLARE c CURSOR FOR SELECT actor_id,first_name FROM actor;
  OPEN c;
  REPEAT
   FETCH c INTO id,fname;
      if fname = 'CHRISTIAN' then
        update actor set first_name = 'CHRIS' where actor_id = id;
        end if;
    select fname;
   UNTIL isnull(id) END repeat;
  CLOSE c;
END
```

http://pastie.org/pastes/9993016/text

```
DELIMITER $$
CREATE PROCEDURE `film_cursor`()
BEGIN
 DECLARE sum float DEFAULT 0;
 DECLARE counter int DEFAULT 0;
 DECLARE a float;
 DECLARE e int default 0;
 DECLARE c CURSOR FOR SELECT length FROM film;

 DECLARE CONTINUE HANDLER FOR NOT FOUND SET e = 1;
  OPEN c;
  repeat
   FETCH c INTO a;
    IF NOT ISNULL(a) THEN
     SET sum = sum + a;
    END IF;

    SET counter = counter + 1;
   UNTIL e END repeat;
  CLOSE c;
 SELECT sum/counter;
END
```

Vežba

- Potrebno je napraviti tabelu bekap korisnika u koju će biti smešteni svi korisnici iz tabele users. Potrebno je napraviti logiku koja će da na svakog obrisanog ili unetog korisnika u tabeli users, reagovati tako što će istog korisnika obrisati ili uneti, u tabelu usersbackup.
- Pomoć (sintaksa za kreiranje insert i delete trigera):

```
delimiter //
create trigger setBackup after insert on users
  for each row
    begin
      ...
    end //
delimiter ;

delimiter //
create trigger deleteBackup after delete on users
  for each row
    begin
      ...
    end //
delimiter ;
```

Rešenje

```
create table usersbackup (id int, name varchar(50), password varchar(50),
status int);

delimiter //
create trigger setBackup after insert on users
  for each row
    begin
      insert into usersbackup (id,name,password,status)
      values (new.id,new.name,new.password,new.status);
end//
delimiter ;

      delimiter //
      create trigger deleteBackup after delete on users
        for each row
          begin
            delete from usersbackup where usersbackup.id = old.id;
          end//
      delimiter ;
```

Vežba

- Potrebno je napraviti before insert triger, koji će prilikom unosa korisnika istog imena u tabelu users svakom novom korisniku dodavati prefix existing.
- Pomoć:
- Kreiranje before insert trigera:

```
delimiter //
create trigger trg_checkUser before insert on users
  for each row
    begin
        ...
end//
delimiter ;
```

Rešenje

```
delimiter //
create trigger trg_checkUser before insert on users
  for each row
    begin
declare existing int;
      set existing = 0;
select count(id) from users where name = new.name into existing;
if existing > 0 then
set new.name = concat('existing_',new.name);
end if;
end//
delimiter ;
```