LINKgroup

Distance Learning System

MySql

Uskladištene rutine

Uskladištene rutine

- Uskladištene rutine su procedure i funkcije koje predstavljaju setove SQL naredbi koje su smeštene na serveru. Na ovaj način nije potrebno da klijenti ponavljaju određene naredbe više puta, već jednostavno mogu pozivati rutine
- Uskladištene rutine mogu biti posebno korisne u sledećim situacijama:
  - kada postoji više klijentskih aplikacija napisanih u različitim programskim jezicima, koje komuniciraju sa istom bazom podataka
  - kada je sigurnost imperativ; u bankarskim sistemima se na primer koriste uskladištene procedure i funkcije za sve operacije; ovo omogućava konzistentno i sigurno okruženje, a korišćenje rutina omogućava da je svaka operacija adekvatno logovana; u takvim scenarijima aplikacije i korisnici, nemaju direktan pristup tabelama baze podataka, već samo mogu da izvršavaju specifične uskladištene rutine

Stored procedures VS User-defined function

U sledećoj tabeli prikazane su osnovne razlike između procedura i funkcija

Stored procedures	**User-definied functions**
mogu vratiti null ili proizvoljan broj vrednosti	mogu vratiti samo jednu vrednost
mogu imati ulazne/uzlazne parametre	mogu imati samo ulazne parametre
moguće korišćenje SELECT i DML upita	moguće korišćenje samo SELECT upita
iz procedure mogu biti pozvane funkcije	iz funkcije ne može biti pozvana procedura
moguća obrada izuzetaka korišćenjem try-catch bloka	try-catch blokovi ne mogu biti korišćeni
ne mogu biti korišćene unutar SQL naredbi	mogu se koristiti unutar SQL naredbi
podržavaju transakcije	ne podržavaju transakcije

Uskladištene procedure

- Uskladištena (Stored) procedura je objekat na serveru, kao i svaki drugi. Stoga, da bismo je kreirali, koristimo DDL naredbu CREATE koja ima sledeću sintaksu:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body
```

```
DELIMITER //
    CREATE PROCEDURE my_procedure()
        BEGIN
            SELECT 'hello from stored procedure';
        END//
DELIMITER ;
```

Strukturni parametri procedure

- Procedura može biti izgrađena sa nekolicinom strukturnih parametara (**CONTAINS SQL, NO SQL, READS SQL DATA, MODIFIES SQL DATA**) koje opisuju prirodu procedure, ali nemaju sintaksnog uticaja na naše rukovanje njome, već pomažu samom MySQL-u prilikom kreiranja statistika i optimizacije:

```
CREATE PROCEDURE my_procedure ()
MODIFIES SQL DATA
BEGIN
//procedure body
END;
```

Determinističke i nedeterminističke procedure

Sličnu svrhu ima i parametar DETERMINISTIC (i NON DETERMINISTIC). Ovaj parametar, takođe, veoma utiče na rad optimizacionog Enginea, ali nema preteran uticaj na naše sintaksne obaveze prilikom pisanja procedure. Ako je procedura označena kao deterministic, znači da se njen izlaz nikada ne menja, kao u prvoj proceduri koju smo napisali, a koja emituje poruku *hello from stored procedure*. Ako procedura nije deterministik (što je podrazumevani parametar), MySQL ne očekuje da rezultat procedure bude uvek isti (na primer, ako procedura sadrži funkciju now(), koja prikazuje tačno vreme i naravno, nikada ne daje identičan rezultat).

Parametrizacija procedure

- Procedura može prihvatiti parametre, i to:
  - Ulazne parametre **(in)**
  - Ulazno izlazne parametre **(inout)**
  - Izlazne parametre **(out)**
- Podrazumevana vrsta (kretanje) parametara procedure su ulazni parametri **(in)**

```
create procedure myproc(p1 int, p2 int)
select p1+p2
```

Parametrizacija procedure

- Osim in parametara, procedura može imati i **out** i **inout** parametre
- Oba se ponašaju kao reference (na primer objektni parametri u Javi)

```
delimiter //
create procedure proba(out p1 int)
begin
set p1 = 25;
end//
delimiter ;

call proba(@a);
select @a;
```

Parametrizacija procedure

- Razlika između out i inout je u tome što **out** nije u stanju da prihvati ulaznu vrednost parametra, a **inout** jeste

Ne može, vraća null

```
delimiter //
create procedure proba(out p1 int)
begin
set p1 = p1 + 1;
end//
delimiter ;

set @a=1;
call proba(@a);
select @a;
```

Može, vraća 2

```
delimiter //
create procedure proba(inout p1 int)
begin
set p1 = p1 + 1;
end//
delimiter ;

set @a=1;
call proba(@a);
select @a;
```

Vežba 1

- Kreirati proceduru za unos korisnika, sa nazivom usp_insertuser koja kao parametar prihvata ime korisnika. Korisnik će biti unet sa statusom user (broj 1)

Vežba 1 - rešenje

```
delimiter //
create PROCEDURE insertuser(newname varchar(50))
BEGIN
    insert into users (name,status) values (newname,1);
END//
```

Vežba 2

- Potrebno je napraviti uskladištenu proceduru koja kreira korisnika prema parametrizovanom imenu. Ukoliko korisnik sa tim imenom već postoji u tabeli, neće biti kreiran. Korisnik se unosi sa statusom user.

Vežba 2 - rešenje

```
delimiter //
create procedure insertuser(newname varchar(50))
begin
declare usersCount int;
select count(*) from users where username = newname into usersCount;
select usersCount;
if usersCount<1 then
    insert into users values (null,newname);
    select '0';
else
    select '-1';
end if;
end //
delimiter ;
```

Vežba 3

- Potrebno je napraviti proceduru koja će unositi korisnika u bazu. Procedura prihvata kao parametre ime i šifru korisnika. Ukoliko korisnik sa tim imenom ne postoji, uneće novog. Ukoliko korisnik postoji, biće mu zamenjena šifra novom. Korisnik se unosi sa statusom user.

Vežba 3 - rešenje

```
delimiter //
create PROCEDURE insertuser(newname varchar(50),newpass varchar(50))
BEGIN
   declare usersCount int;
   select count(id) from users where name=newname into usersCount;
   if usersCount<1 then
    insert into users (name,password,status) values (newname,password,1);
   else
    update users set password=newpass where name = newname;
   end if;
END//
delimiter ;
```

Vežba 4

- Napraviti proceduru koja će vratiti ime, prezime i ukupnu količinu potrošenog novca na porudžbine, na osnovu id-a korisnika prosleđenog kao parametar procedure

```
mysql> call usp_get_user_payments(15);
+------------+-----------+--------+
| first_name | last_name | am     |
+------------+-----------+--------+
| HELEN      | HARRIS    | 134.68 |
+------------+-----------+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Vežba 4 rešenje

- Napraviti proceduru koja će vratiti ime, prezime i ukupnu količinu potrošenog novca na porudžbine, na osnovu id-a korisnika prosleđenog kao parametar procedure

```
delimiter //
drop procedure if exists usp_get_user_payments;
create procedure usp_get_user_payments(customer_id int)
begin

select first_name,last_name,sum(amount) am from payment
join customer on
payment.customer_id = customer.customer_id
where payment.customer_id = customer_id
group by first_name,last_name
order by am desc;

end//
delimiter ;
```

Vežba 5 - Money transfer functionality

U bazi podataka postoji sledeća tabela:

id int
username varchar
balance numeric(9,2)

Kreirati uskladištenu procuduru koja će prebacivati novac sa jednog na drugog korisnika (kolona balance)
Procedura treba da prihvata tri parametra:
Prvi parametar ce biti id korisnika kome će biti oduzet novac
Drugi parametar id korisnika kome se dodeljuje novac
Treći parametar će biti suma koja ce biti oduzeta od jednog i dodeljena drugom korisniku

Procedura mora proveriti da li korisnik ima dovoljno novca na računu pre nego što transakcija bude izvršena (balance izvornog korisnika ne sme da ode u minus)

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 ;
```

Vežba

- Potrebno je napraviti funkciju koja, za uneti naziv države vraća ukupan ukupnu zaradu iz te države

```
mysql> select udf_get_sum_for_country('yugoslavia');
+-----------------------------------------+
| udf_get_sum_for_country('yugoslavia') |
+-----------------------------------------+
|                                  259.43 |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Rešenje

```
delimiter //
create function udf_get_sum_for_country(countryname varchar(256))
returns decimal(9,2)
deterministic
begin
return (select sum(payment.amount) as am from city
join address on address.city_id = city.city_id
join country on city.country_id = country.country_id
join customer on customer.address_id = address.address_id
join payment on payment.customer_id = customer.customer_id
where country.country like concat('%',countryname,'%')
group by country.country
order by am desc
limit 1);
end //
```

Vežba

- U bazi podataka se nalazi tabela ages_sms, koja prikazuje broj poslatih sms poruka u odnosu na starost korisnika
- Potrebno je napraviti funkciju koja će, na osnovu starosti novog korisnika, dati preporuku za paket (broj) sms poruka

```
mysql> select udf_predict_sms(40);
+----------------------+
| udf_predict_sms(40) |
+----------------------+
|                30.24 |
+----------------------+
1 row in set, 1 warning (0.00 sec)
```

Pomoć:

$$a = \frac{(\Sigma y\ \Sigma x^2) - (\Sigma x\ \Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x\ \Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$\mathbf{y = a + (b * x)}$$

Rešenje

```
create function udf_predict_sms(userage int)
returns decimal(9,2)
deterministic
begin

return (
select
    -- a --
    (((sum(messages) * sum(pow(user_age,2))) - (sum(user_age) * sum(user_age * messages)))
    /
    ((count(*)*sum(pow(user_age,2))) - pow(sum(user_age),2)))
    +
    -- b --
    ((((count(*) * sum(user_age * messages)) - (sum(user_age) * sum(messages)))
    /
    ((count(*) * sum(pow(user_age,2))) - (pow(sum(user_age),2)))) * userage)
    from ages_sms

);
end//
```