POHRANJENE RUTINE U MySQL

IX auditorne vježbe

Teme

- Uvod u pohranjene rutine u MySQL
- Pisanje pohranjenih rutina
  - Početak i kraj, ime i parametri rutine
  - Return klauzula
  - Komentari
- Definisanje i upotreba varijabli
  - Lokalne varijable
  - Dodjeljivanje vrijednosti varijablama
- Izrazi u pohranjenim rutinama
- Pisanje bloka naredbi
- Vraćanje vrijednosti iz pohranjene funkcije
- Izvršavanje rutina

Pohranjene rutine

- Pohranjena rutina je skup SQL iskaza koji može biti pohranjen na server.
- Generički termin koji uključuje pohranjene procedure i pohranjene funkcije.
- Pohranjena procedura je rutina koja može vratiti vrijednost.
- Pohranjena funkcija je rutina koja vraća neku vrijednost
- MySQL koristi SQL:2003 sintaksu za pohranjene rutine koja se također koristi od strane IBM i DB2
- Pohranjene rutine se kreiraju sa naredama CREATE PROCEDURE i CREATE FUNCTION
- Pohranjena procedura ili funkcija je pridružena određenoj bazi podataka što ima sljedeće implikacije:
  - Kada se rutina poziva implicitno se izvodi USE db_name (a poništava kada rutina završava). USE iskaz unutar rutine nije dozvoljen
  - Naziv rutine se može kvalificirati sa nazivom baze podataka, npr. test.f()
  - Kada se briše baza podataka brišu se i sve rutine pridružene toj bazi

Pisanje rutina

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

CREATE
[DEFINER = { ***user*** | CURRENT_USER }]
FUNCTION ***sp_name*** ([***func_parameter***[,...]])
RETURNS ***type***
[***characteristic*** ...] ***routine_body***

proc_parameter:
[IN | OUT | INOUT] ***param_name type***

Pisanje rutina

func_parameter:

param_name type

type:

Any valid MySQL data type

characteristic:

COMMENT '***string***'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }

routine_body:

Valid SQL routine statement

Ime rutine

- Ime rutine se navodi odmah iza CREATE PROCEDURE ili CREATE FUNCTION .
- Ime procedure može biti dugo do 18 karaktera, i vrijede pravila za MySQL identifikatore.
- Ukoliko je naziv rutine isti kao naziv ugrađene SQL funkcije javlja se greška, osim ukoliko se koristi razmak između naziva i zagrada prilikom definisanja rutine i kasnijih poziva
- Kada se rutina želi u cijelosti proslijediti serveru potrebno je zamijeniti podrazumjevani delimiter znak koji je (;) naredbom DELIMITER, npr. DELIMITER //
- Po kreiranju procedure može se ponovo vratiti podrazumijevani delimiter znak

Parametri rutine

- Svaka rutina ima listu sa **nula ili više parametara,** navedenih u zagradama iza imena rutine.
- Svaki parametar je podrazumijevano IN parametar, a da bi se specificiralo drugačije koristi se OUT ili INOUT ispred naziva parametra
- Ovo je validno samo za procedure, parametri funkcije su uvijek IN parametri
- IN parametar prosljeđuje vrijednosti proceduri
- OUT parametar prosljeđuje vrijednost iz procedure natrag onome ko je poziva. Njegova početna vrijednost unutar procedure je NULL
- INOUT se inicijalizira od strane onoga ko poziva proceduru, može se mijenjati u proceduri, a te promjene su vidljive onome ko poziva proceduru nakon što procedura završi

Return klauzula i tijelo rutine

- RETURNS klauzula se može specificirati samo za funkciju, za koju je obavezna
- Njom se indicira povratni tip funkcije i tijelo funkcije mora sadržavati RETURN iskaz
- Routine_body se sastoji od validnih SQL iskaza za rutine. To može biti jednostavan iskaz kao SELECT ili INSERT, ili složeni iskaz napisan korištenjem BEGIN i END
- Složeni iskazi mogu sadržavati deklaracije, petlje i druge iskaze kontrole toka
- MySQL dozovljava da rutine sadrže DDL iskaze kao što su CREATE i DROP
- Također dozvoljava pohranjenim procedurama (ali ne i funkcijama) da sadrže SQL transakcijske iskaze kao što je COMMIT
- Iskazi koji vraćaju skup rezultata se mogu koristiti unutar pohranjenih procedura, ali ne i unutar pohranjenih funkcija

Brisanje rutine

- **Nakon kreiranja** SPL rutine, **nije moguće mijenjati** njeno tijelo.
- Umjesto toga, ona se mora izbrisati i ponovo kreirati.
- Kreirane rutine se brišu iz baze podataka naredbom DROP PROCEDURE ili DROP FUNCTION, na primjer.

```
DROP PROCEDURE povecaj_cijene;
DROP FUNCTION citaj_adresu;
```

- ALTER FUNCTION i ALTER PROCEDURE se mogu koristiti samo za promjenu karakteristika pohranjene rutine

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
    DATA}
  | SQL SECURITY { DEFINER | INVOKER }
```

Definisanje i upotreba varijabli

- Za **definisanje** varijabli u rutini koristi se naredba **DECLARE**

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

- Ova naredba deklariše lokalnu varijablu u pohranjenoj rutini
- Može se koristiti DEFAULT klauzula kako bi se varijabli dodijelila podrazumijevana vrijednost. Bez nje početna vrijednost je NULL
- Deklaracija varijabli se mora pojaviti prije deklaracije kursora
- Nazivi lokalnih varijabli nisu case sensitive i vrijede pravila kao za ostale MySQL identifikatore
- Vidljivost lokalnih varijabli je BEGIN ... END blok unutar kojeg su definisane
- Varijabli se može pristupiti iz ugniježdenih blokova osim ukoliko u njima nije deklarisana lokalna varijabla sa istim imenom
- Lokalna varijabla ne bi trebala imati isti naziv kao kolona u tabeli. MySQL interpretira referencu na kolonu kao naziv varijable u tom slučaju

Definisanje i upotreba varijabli

```
CREATE PROCEDURE scope()
  BEGIN
    DECLARE x,y,z INT;
    SET x = 5;
    SET y = 10;
    SET z = x + y; --z je 15
    BEGIN
      DECLARE x, q INT;
      DECLARE z CHAR(5);
      SET x = 100;
      SET q = x + y; -- q = 110
      SET z = 'silly'; -- z dobija karakter vrijednost
    END;
    SET y = x; -- y je sada 5
    SET x = z; -- z je sada 15, a ne 'silly'
END;
```

Dodjeljivanje vrijednosti varijablama

- Varijabli u rutini se vrijednost može dodijeliti upotrebom:
  - SET naredbe.
  - SELECT…INTO naredbe.

```
DECLARE total INT DEFAULT 0
SET total = 10;
```

```
DECLARE total INT DEFAULT 0
SELECT COUNT(*) INTO total FROM proizvod
```

- Varijabla koja počinje sa znakom @ je varijabla sesije te joj se može pristupiti sve dok se sesija ne okonča

Pisanje bloka naredbi

- Za pisanje složenog bloka naredbi koristi se BEGIN ... END sintaksa

```
[begin_label:] BEGIN
  [statement_list]
END [end_label]
```

- Složeni iskaz može sadržavati više naredbi obuhvaćenih BEGIN i END ključnim riječima
- Statement_list predstavlja listu jedne ili više naredbi od kojih se svaka završava sa (;)
- BEGIN ... END blokovi mogu biti ugniježđeni

Upotreba IF - ELSEIF - ELSE strukture

- Pohranjena rutina u sljedećem primjeru koristi jednu **IF-ELSEIF-ELSE** strukturu za komparaciju svoja dva argumenta.

```
CREATE FUNCTION str_compare(str1 CHAR(20), str2 CHAR(20))
  RETURNS INTEGER
  BEGIN
     DECLARE result INTEGER;
     IF str1 > str2 THEN
         SET result = 1;
     ELSEIF str2 > str1 THEN
         SET result = -1;
     ELSE
         SET result = 0;
     END IF;
     RETURN result;
END;
```

Upotreba IF - ELSEIF - ELSE strukture

- **IF - ELSEIF - ELSE** struktura u rutini **može** imati slijedeća **4 dijela**:
  - **IF…THEN uslov** : ako je uslov iza IF naredbe ispunjen (TRUE), rutina izvršava naredbe u IF bloku, inače izvršava ELSEIF uslov.
  - **Jedna ili više ELSEIF uslova (opciono)**: rutina provjerava ELSEIF uslov samo ako IF uslov nije ispunjen. Ako je ELSEIF uslov ispunjen, rutina izvršava naredbe u tom ELSEIF bloku, inače provjerava slijedeći ELSEIF blok (ako postoji) ili izvršava ELSE naredbu.
  - **ELSE uslov (opciono)**: rutina izvršava naredbe u ELSE bloku ako nije ispunjen IF uslov niti bilo koji ELSEIF uslov.
  - **END IF naredba**: završava ovaj blok naredbi.

Izvršavanje rutina

- Za poziv pohranjene procedure koristi se CALL naredba
- Pohranjena funkcija se poziva korištenjem njenog imena u izrazu
- CALL poziv može vratiti vrijednosti onome ko poziva proceduru korištenjem parametara koji su deklarisani kao OUT i INOUT
- Potrebno je proslijediti parametar kao korisničku varijablu, a potom provjeriti vrijednost varijable nakon što procedura završi
- Prije poziva procedure potrebno je inicijalizirati varijablu koja se prosljeđuje kao INOUT parametar

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
    BEGIN
        SELECT VERSION() INTO ver_param;
        SET incr_param = incr_param + 1;
END;
SET @increment = 10;
CALL p(@version, @increment);
SELECT @version, @increment;
```

Izvršavanje rutina

- U pripremljenom CALL izrazu korištenim sa PREPARE i EXECUTE mogu se koristiti placeholder-i za IN parametre
- Za OUT i INOUT parametre podrška za placeholder-e postoji od verzije MySQL 5.5.3

```
SET @increment = 10;
PREPARE s FROM 'CALL p(?, ?)';
EXECUTE s USING @version, @increment;
SELECT @version, @increment;
```