POHRANJENI ZADACI

1. Općenito o pohranjenim zadacima

Pohranjeni zadaci neovisni su dijelovi koda namijenjeni višekratnoj uporabi koja enkapsulira određene radnje, a mogu se pozivati(unutar aplikacija) prema potrebi. Koncept pohranjenih zadataka blizak je programerskom svijetu kroz koji višekratna uporabljivost koda postaje obilježje SQL upita. U pohranjenim se zadacima prožimaju mogućnosti SQL upita s lokalnim i globalnim varijablama, naredbama za kontrolu toka programa te naredbama za rukovanje iznimkama.

Prilikom porasta složenosti poslovne logike aplikacija koje se vežu na bazu podataka, česta je pojava repetitivnog pisanja niza istih SQL naredbi. Primjerice, više puta piše se ista INSERT naredba za unos n-torke u tablicu ili isti izračuni nad određenim setom podataka. Tada je poželjno pohraniti niz ovakvih naredbi u pohranjeni zadatak koji će se nalaziti na poslužitelju baze podataka i izvršavati na način da aplikacija na klijentskoj strani ne mora prenositi niz naredbi do poslužitelja baze, već samo ime zadatka koji će obaviti upite nad bazom.

Prednosti pohranjenih zadataka

Pohranjivanje SQL upita na poslužitelju u obliku pohranjenih zadataka ima sljedeće prednosti:

- Pohranjeni zadaci nalaze se na poslužitelju baze podataka umjesto u samoj aplikaciji. U kontekstu klijent-poslužitelj arhitekture, brže je i **manje se opterećuje prijenosni kanal** ako se prenosi samo naziv pohranjenog zadatka, u odnosu na prijenos niza naredbi koje se trebaju izvršiti.
- **Izbjegava se repetitivno pisanje istih naredbi** čime se smanjuje mogućnost pogreške. Time se također dobiva na preglednosti i efikasnosti samoga koda aplikacije.
- Povećana je **prenosivost aplikacije na drugu platformu**. Razna sučelja i aplikacije napisane primjerice u jeziku Java, VB, Excel, pozivaju se primjerice pomoću JDBC, JDBC/ODBC ili ODBC programskih sučelja istog pohranjenog zadatka.
- **Povećana je sigurnost i izolacija podataka** jer postoji mogućnost dodjeljivanja dozvola (GRANT) za pristup podacima na razini pohranjenih zadataka. SPL (*Stored Procedure Language*) 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.
- Logika baze podataka ostaje u samoj bazi, čime se **povećava robusnost** aplikacija na razini **konzistentnosti podataka** i izbjegava nedosljednost podataka.
- **Nadograđene su osnovne funkcionalnosti** baze podataka aritmetikom, varijablama i petljama.

Nedostatci pohranjenih zadataka

Umjesto dohvata podataka za što su baze optimizirane, od pohranjenog zadatka očekuje se rad s logičkim operacijama i poslovnom logikom. Posljedica je opterećenje poslužitelja odnosno opterećenje CPU-a i memorije.

Postoje ograničenja u pohranjenim zadacima u obliku ograničenosti naredbi koje se mogu koristiti. SPL nije kompleksan kao C++, VB, Java i ostali.

Upravljanje pogreškama (*error handling*) je loše što uzrokuje složenost razvoja pohranjenog zadatka jer je potrebno dobro znanje SQL jezika.

Nedostatkom se može smatrati što je poslovna logika u bazi umjesto u aplikaciji. (Ovo je prednost u slučajevima operacija nad kompleksnim setom rezultata, kada je bolje koristiti pohranjeni zadatak kako se složenost entiteta ne bi prenosila u aplikacijski sloj.)

Podjela pohranjenih zadataka

Pohranjeni zadaci dijele se na:

- pohranjene rutine (*stored routines*):
  - pohranjene procedure (*stored procedures*),
  - pohranjene funkcije (*stored functions*),
- okidače (*triggers*),
- pogađaje (*events*).

Pohranjene rutine

Deklaracija procedura i deklaracija funkcija međusobno su vrlo slične te je sličan i način njihovog izvođenja. Razlike između procedura i funkcija mogu se svesti na sljedeće:

- Funkcije ne mogu vratiti skup rezultata.
- Funkcije ne mogu koristiti SQL naredbe koje podrazumijevaju transakcijsku obradu (COMMIT, ROLLBACK).
- Funkcija ne može sama sebe rekurzivno pozivati.
- Funkcije moraju vratiti rezultat.

2. Varijable

Pohranjeni zadaci će sa bazom podataka komunicirati putem varijabli. Razlikujemo:

- GLOBALNE VARIJABLE
  - varijable u MySQL-u koje počinju znakom @,
  - vrijede u jednoj sesiji (spoju na poslužitelj),
  - nije ih potrebno deklarirati.
- LOKALNE VARIJABLE
  - varijable deklarirane unutar bloka BEGIN … END pohranjenog zadatka,
  - ne moraju početi sa znakom @,
  - vidljive su unutar bloka pohranjenog zadatka,
  - deklaracija se mora nalaziti uvijek na vrhu procedure, poslije BEGIN i obavezno prije početka nekoga drugoga koda.

Varijabli je moguće dodijeliti vrijednost ključnom riječi **SET**:

```
SET var1=1234;
SET @var2=5678;
```

U varijablu je moguće pohraniti (jednu) vrijednost izvođenja upita koristeći ključnu riječ **INTO**:

```
SELECT AVG(koefPlaca) INTO prosjek1 FROM radnik;
SELECT AVG(koefPlaca) INTO @prosjek2 FROM radnik;
```

Moguće je i:

```
SET prosjek1=(SELECT AVG(koefPlaca) FROM radnik);
SET @prosjek2=(SELECT AVG(koefPlaca) FROM radnik);
```

3. Pohranjene procedure

Za pohranjene procedure karakteristično je da obavljaju niz SQL naredbi te da mogu vratiti skup rezultata. Rezultat ne mogu vratiti direktno (kao što će to kasnije moći funkcija) već rezultat vraćaju preko varijabli ili koristeći SELECT naredbu. Procedura može primiti varijable koje mogu biti modificirane unutar tijela procedure.

Procedura se definira prema sljedećim koracima:

1. Izradi proceduru (ključna riječ **CREATE PROCEDURE**).
2. Definiraj ime procedure.
3. Definiraj ulazne i izlazne parametre.
4. Definiraj tijelo procedure (ako se procedura sastoji od više naredbi, potrebno ih je definirati unutar **BEGIN … END** bloka).

Osnovna sintaksa:

```
CREATE PROCEDURE ime_procedure ([parametri[,...]])
    tijelo_procedure
```

Pohranjenom zadatku može se dodijeliti ime od najviše 64 znakova, pri čemu se ime ne smije sastojati samo od brojki. Ime može sadržavati i posebne znakove, ali u tom slučaju moraju se označiti s unazadnim navodnikom (`). Preporuča se izbjegavati nazivanje pohranjenih podataka imenima baze, tablice ili atributa.

Izrada i pozivanje procedure

i. Odabir graničnika

Proceduru kao vrstu pohranjenog zadatka želimo izvesti na način da se izvedu sve naredbe od CREATE PROCEDURE do END. U pretpostavljenom načinu rada DBMS prekida izvođenje upita kada dođe do kraja trenutnog upita. Kraj upita označava **graničnik** (***DELIMITER***) čija je pretpostavljena vrijednost „ ;“ (točka zarez). Graničnik je znak ili niz znakova koji govori MySQL-u da se na tom mjestu nalazi kraj SQL naredbe.

Da bi se izbjeglo zaustavljanje izvođenja pohranjenog zadatka prilikom nailaska na prvi „;“, potrebno je privremeno izmijeniti vrijednost graničnika na neku drugu vrijednost – u ovom primjeru mijenja se na „//“.

Graničnik može poprimiti bilo koju vrijednost,

- //
- #
- nekoImeGranicnika

ali se ne preporuča korištenje oznaka koje bi se mogle pronaći u tijelu procedure:

- *
- (,)
- =, <, >

Po završetku procedure graničnik je potrebno vratiti na njegovu inicijalnu vrijednost naredbom „DELIMITER ;“.

ii. Rad s procedurom

i. Kreiranje procedure

Primjer:

Napisati proceduru koja ispisuje sve podatke o svim klijentima:

```
DELIMITER //
CREATE PROCEDURE ispisiKlijente()
BEGIN
        SELECT * FROM klijent;

END; //
DELIMITER ;
```

Privremena promjena graničnika
Deklaracija procedure
Tijelo procedure
Vraćanje vrijednosti graničnika na ;

Gornjim je kodom procedura kreirana i pohranjena, ali ne i pozvana. Nakon deklaracije procedure slijedi tijelo procedure. Ako se tijelo sastoji od više naredbi, obavezno je definirati ih unutar BEGIN … END bloka.

ii. Pozivanje procedure

Procedura se poziva ključnom riječi CALL nakon čega slijedi ime procedure te navođenje vrijednosti parametara koje se proceduri predaje i varijabli u koje se rezultat pohranjuje.

Osnovna sintaksa:

```
CALL ime_procedure ([parametri[,...]])
```

Primjer poziva gornje procedure:

```
CALL ispisiKlijente();
```

Pozivanje gornje procedure rezultirat će sljedećim skupom podataka:

sifKlijent	imeKlijent	prezimeKlijent	pbrKlijent	pbrReg	datUnosKlijent	jmbgKlijent
1137	Jure	Ribarić	22000	22000	1986-10-29	2910986392304
1139	Niko	Marušić	48000	48000	1987-08-19	1908987173977
1140	Davor	Vurnek	20000	20000	1987-10-26	2610987300802
1141	Zoran	Habajec	21000	21000	1987-07-22	2207987301807
1143	Davor	Voras	20000	10000	0000-00-00	
1144	Zvonimir	Ozimec	21000	21000	1988-10-04	0410988390015
1145	Jurica	Bašić	10000	10000	1988-07-02	0207988361605
1147	Alenka	Vukojević	21000	21000	1988-06-10	1006988311906
1148	Antonijo	Javorina	10000	10000	1987-07-13	1307987330068
1149	Nikola	Bačić	20000	10000	1987-04-05	0504987330122
1151	Karlo	Krsnik	10010	10000	1987-08-15	1508987320522
1152	Božidar	Tomić	20250	20250	1988-08-03	0308988395074

iii. Izmjena procedure

MySQL omogućava naredbom ALTER PROCEDURE izmijeniti samo određene karakteristike procedure. Ako postoji potreba za izmjenom tijela procedure, s obzirom da DBMS ne dozvoljava više procedura istog imena, potrebno je proceduru obrisati te je nakon toga ponovno izraditi sa izmijenjenim tijelom.

iv. Brisanje procedure

Osnovna sintaksa:

```
DROP PROCEDURE [IF EXISTS] ime_procedure;
```

Klauzula IF EXISTS sprečava pojavu pogreške prilikom izvođenja naredbe brisanja ako procedura s danim imenom ne postoji. U tom se slučaju neće dogoditi pogreška već će se samo ispisati upozorenje o nepostojanju procedure koja se pokušava obrisati.

Primjer:

```
DROP PROCEDURE IF EXISTS proceduraKojaNePostoji();
```

Ako *proceduraKojaNePostoji* ne postoji u bazi podataka, MySQL će vratiti samo upozorenje kao u nastavku.

```
Query:  drop procedure if exists proceduraKojaNePostoji()

Error Code: 1064
You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '()' at line 1

Execution Time : 0 sec
Transfer Time  : 0 sec
Total Time     : 0 sec
```

iii. Definiranje parametara

Procedura *ispisiKlijente* tip je pohranjenog zadatka koji nema ulazne parametre te će prilikom poziva procedure uvijek vratiti isti skup rezultata (svi podaci o svim klijentima). Potpuniji smisao procedura dobiva ako joj se definira ulazni parametar u odnosu na kojeg je u mogućnosti vratiti skup rezultata koji će ovisiti o tom parametru. Parametar nije ništa drugo već varijabla pomoću koje procedura komunicira s bazom podataka

Osnovna sintaksa:

```
CREATE PROCEDURE imeProcedure ([IN] x tipPodatka,  INOUT y tipPodatka, OUT z
tipPodatka)
        ...tijelo procedure...
```

Postoje sljedeće vrste parametara:

- Ulazni parametri – IN
  - Koriste se kao ulaz u proceduru
  - Procedura ne može mijenjati vrijednost ove varijable
  - Ako se ne navede tip ulaza, podrazumijeva se IN
- Izlazni parametri – OUT
  - Koristi se samo kao izlaz iz procedure
  - Procedura može mijenjati vrijednost ove varijable
- Ulazno-izlazni parametri – IN/OUT
  - Koristi se kao izlaz i ulaz u proceduru

Primjer (ulazni parametar):

Napisati proceduru koja ispisuje sve podatke klijentima sa zadanim poštanskim brojem stanovanja.

```
DELIMITER //
CREATE PROCEDURE ispisiKlijenteIzMjesta(IN zadanoMjesto VARCHAR(50))
BEGIN
        SELECT * FROM klijent JOIN mjesto ON pbrKlijent=pbrMjesto
                WHERE nazivMjesto=zadanoMjesto;
END; //
DELIMITER ;
```

Potrebno je voditi računa o navođenju parametara prilikom poziva procedure. Proceduri treba predati točno onoliko parametara koliko je navedeno prilikom deklaracije procedure

Primjer poziva procedure:

```
CALL ispisiKlijenteIzMjesta('Zagreb');
```

Rezultat pozivanja procedure sa vrijednošću ulaznog parametra 'Zagreb':

sifKlijent	imeKlijent	prezimeKlijent	pbrKlijent	pbrReg	datUnosKlijent	jmbgKlijent	pbrMjesto	nazivMjesto	sifZupanija
1145	Jurica	Bašić	10000	10000	1988-07-02	0207988361605	10000	Zagreb	21
1148	Antonijo	Javorina	10000	10000	1987-07-13	1307987330068	10000	Zagreb	21
1173	Gordan	Boršić	10000	20000	1980-02-29	1902980334016	10000	Zagreb	21
1177	Kristian	Klarin	10000	10000	1988-06-06	0606988330028	10000	Zagreb	21
1181	Arsenio	Minić	10000	20000	1987-09-14	1409987320508	10000	Zagreb	21
1184	Tihomir	Fabris	10000	20000	1987-09-05	0509987360058	10000	Zagreb	21
1197	Dominik	Hacek	10000	10000	1988-04-08	0804988306801	10000	Zagreb	21

Primjer (izlazni parametar):

Potrebno je napisati proceduru koja će prebrojati klijente iz zadanog mjesta te preko parametra vratiti broj klijenata.

```
DELIMITER //
CREATE PROCEDURE prebrojiKlijenteIzMjesta(IN zadanoMjesto VARCHAR(50),
                                          OUT brojKlijenata INT)
BEGIN
        SELECT COUNT(*) INTO brojKlijenata
```

```
                FROM klijent JOIN mjesto ON pbrKlijent=pbrMjesto
                WHERE nazivMjesto=zadanoMjesto;
END; //
DELIMITER ;
```

U WHERE klauzuli selektirani su samo oni klijenti koji se odnose na mjesto definirano putem ulaznog parametra (*nazivMjesto=zadanoMjesto*). Rezultat ugrađene funkcije COUNT, pomoću ključne riječi **INTO** pohranjuje se u izlaznu varijablu *brojKlijenata*, koju procedura automatski vraća kao izlaz iz procedure.

Primjer poziva procedure i ispisa rezultata:

```
CALL prebrojiKlijenteIzMjesta('Zagreb',@br);
SELECT @br AS broj_mjesta;
```

Prilikom poziva procedure potrebno je osim definiranja vrijednosti za ulazni parametar (primjerice 'Zagreb'), definirati i ime (globalne) varijable u koju će se pohraniti vrijednost izlazne varijable odnosno rezultat izvođenja procedure. Varijabla može imati proizvoljan naziv.

Pozivom procedure neće se (u ovom slučaju) ispisati rezultat već će se on nalaziti u globalnoj varijabli *@br*. Sadržaj te varijable ispisuje se narednom SELECT. Rezultat pozivanja procedure s vrijednošću ulaznog parametra 'Zagreb' je sljedeći:

broj_mjesta
51

Ovaj primjer moguće je riješiti i bez izlaznog parametra na način da procedura vrati rezultat kroz SELECT klauzulu definiranu unutar tijela procedure:

```
DELIMITER //
CREATE PROCEDURE prebrojiKlijenteIzMjesta(IN zadanoMjesto VARCHAR(50))
BEGIN
        SELECT COUNT(*)
                FROM klijent JOIN mjesto ON pbrKlijent=pbrMjesto
                WHERE nazivMjesto=zadanoMjesto;
END; //
DELIMITER ;
```

Primjer poziva procedure:

```
CALL prebrojiKlijenteIzMjesta('Zagreb');
```

Pozivanjem ove procedure direktno se ispisuje rezultat. Primjer poziva sa vrijednošću ulaznog parametra 'Zagreb' je sljedeći:

COUNT(*)
51

Treći je način vraćanje rezultata preko globalne varijable:

```
DELIMITER //
CREATE PROCEDURE prebrojiKlijenteIzMjesta(IN zadanoMjesto VARCHAR(50))
BEGIN
        SELECT COUNT(*) INTO @n
                FROM klijent JOIN mjesto ON pbrKlijent=pbrMjesto
```

```
                WHERE nazivMjesto=zadanoMjesto;
END; //
DELIMITER ;
```

Primjer poziva procedure i ispisa rezultata:

```
CALL prebrojiKlijenteIzMjesta('Zagreb');
SELECT @n;
```

@n
51

Primjer (ulazno-izlazni parametar):

Potrebno je napisati proceduru koja će za zadanog klijenta vratiti poštanski broj mjesta u kojem stanuje.

```
DELIMITER //
CREATE PROCEDURE nadiPbrKlijenta(INOUT klijentMjesto INT)
BEGIN
        SELECT pbrKlijent INTO klijentMjesto
                FROM klijent WHERE sifKlijent=klijentMjesto;
END; //
DELIMITER ;
```

Varijabla *klijentMjesto* ima ulogu prenošenja ulaznog podatka proceduri (WHERE *sifKlijent*=*klijentMjesto*) te primanja rezultata procedure (SELECT *pbrKlijent* INTO *klijentMjesto*).

Primjer poziva procedure:

```
SET @a=1426;
CALL nadiPbrKlijenta(@a);
SELECT @a AS pbr_stanovanja;
```

Prilikom poziva procedure, ulazno-izlazna varijabla komunicirat će sa sesijom preko globalne varijable *@a*. Varijabli *@a* ključnom riječju **SET** pridodana je vrijednost 1426. Procedura traži poštanski broj stanovanja klijenta sa šifrom 1426 te ga pohranjuje u *@a*. Naredbom CALL rezultat neće biti ispisan, već će biti upisan u *@a*. Sadržaj globalne varijable *@a* ispisuje se naredbom SELECT.

pbr_stanovanja
20000

4. Pohranjene funkcije

Razlika između procedura i funkcija

Pohranjena funkcija vrsta je pohranjene rutine koja direktno vraća rezultat te ne može vratiti skup rezultata. Za nju je karakteristično da s bazom podataka komunicira isključivo putem ulaznih

parametara, dok izlazne parametre ne podržava. Podatke prima preko ulaznih varijabli koje može modificirati. S obzirom da rezultat ne može biti vraćen preko parametra, prilikom deklaracije funkcije nužno je navesti tip podatka kojeg će funkcija (obavezno) vratiti.

Za razliku od procedura, funkcija mora vratiti rezultat i ne može ispisivati podatke unutar BEGIN … END bloka.

Definiranje funkcije

Funkcija se definira prema sljedećim koracima:

1. Izradi funkciju (ključna riječ **CREATE FUNCION**)
2. Definiraj ime funkcije
3. Definiraj ulazne i izlazne parametre -> svi su parametri ulaznog tipa (drugi tipovi nisu dozvoljeni) te se iz tog razloga **ne navodi** ključna riječ IN prije imena parametra
4. Ključna riječ **RETURNS** i **tip parametra koji funkcija vraća**
5. Definirati je li funkcija **DETERMINISTIC** ili **NOT DETERMINISTIC**
6. Definirati tijelo procedure (ako se procedura sastoji od više naredbi, potrebno ih je definirati unutar **BEGIN … END** bloka)
7. Unutar tijela funkcije naredbom **RETURN** definirati izlaz iz funkcije

Pravila koja su vrijedila za procedure, a tiču se definiranja graničnika, izmjene i brisanja procedura vrijede i za pohranjene funkcije.

Osnovna sintaksa:

```
CREATE FUNCTION ime_funkcije ([parametri[,...]]) RETURNS tip_podatka
        [NOT] DETERMINISTIC [karakteristike]
        tijelo_funkcije
```

DETERMINISTIC i NOT DETERMINISTIC

Prilikom deklaracije funkcije nužno je definirati je li funkcija deterministička ili je nedeterministička. Ako funkcija ima determinističko obilježje, tada se kaže da je strogo određena. Za iste će ulazne parametre uvijek vratiti isti rezultat. Navedeno se podrazumijeva u slučaju replikacije i u situacijama kada se unutar funkcije pozivaju ugrađene funkcije tipa RAND() i NOW().

Definiranje ostalih karakteristika funkcije je opcionalno i služi isključivo kao komentar (MySQL ne provjerava istinitost navedenih karakteristika) . Moguće je sljedeće:

- MODIFIES SQL DATA – funkcija mijenja podatke (naredbama INSERT, UPDATE i DELETE)
- READS SQL DATA – funkcija sadrži isključivo naredbe čitanja podataka (SELECT)
- CONTAINS SQL – funkcija ne sadrži naredbe za čitanje ili pisanje u bazu (primjerice može sadržavati CURDATE() ugrađenu funkciju)
- NO SQL – funkcija ne sadrži SQL upite

Rad s funkcijom

i. Kreiranje funkcije

Primjer:

Napisati funkciju koja ispisuje najveću šifru radnika:

```
DELIMITER //
CREATE FUNCTION vratiNajvecuSifru() RETURNS INT
DETERMINISTIC
BEGIN
        DECLARE vrati INT DEFAULT NULL;
        SELECT MAX(sifKlijent) INTO vrati FROM klijent;
        RETURN vrati;
END; //
DELIMITER ;
```

Gornjim kodom funkcija je kreirana i pohranjena, ali ne i pozvana. Obavezno je nakon navođenja imena i parametara definirati tip podatka kojeg funkcija vraća.

ii. Pozivanje funkcije

Funkcija se poziva ključnom riječi SELECT nakon čega slijedi ime funkcije te navođenje vrijednosti parametara koje se funkciji predaje.

Osnovna sintaksa:

```
SELECT ime_funkcije ([parametri[,...]])
```

Primjer poziva gornje funkcije:

```
SELECT vratiNajvecuSifru();
```

Pozivanje gornje funkcije rezultirat će sljedećim:

vratiNajvecuSifru()
1519

iii. Izmjena funkcije

MySQL omogućava naredbom ALTER FUNCTION izmijeniti samo određene karakteristike funkcije. Ako postoji potreba za izmjenom tijela funkcije, s obzirom da DBMS ne dozvoljava više funkcija istog imena, potrebno je funkciju obrisati te je nakon toga ponovno izraditi s izmijenjenim tijelom.

iv. Brisanje funkcije

Osnovna sintaksa:

```
DROP FUNCTION [IF EXISTS] ime_funkcije;
```

v. Definiranje parametara

Osnovna sintaksa:

```
CREATE PROCEDURE imeFunkcije (x tipPodatka)
        ...tijelo procedure...
```

Funkcija podržava isključivo ulazne parametre. Pri tome se ne smije navoditi IN kao oznaka tipa parametra jer druga opcija ne postoji.

Primjer:

Napisati funkciju koja za zadanog radnika vraća broj naloga na kojima je radio.

```
DELIMITER //

CREATE FUNCTION brojiNaloge(zadaniRadnik INT) RETURNS INT
DETERMINISTIC
BEGIN
        RETURN (SELECT COUNT(*) FROM nalog WHERE sifRadnik=zadaniRadnik);
END; //
DELIMITER ;
```

Primjer poziva funkcije;

```
SELECT brojiNaloge(456);
```