KURSORI

Kursor je objekt baze podataka koji omogućava dohvat pojedinog zapisa iz rezultata SELECT upita. Služi isključivo za dohvat podataka, a ne za ažuriranje, unos niti brisanje podataka. U slučajevima kada se pomoću naredbe SELECT dohvaća više od jednog zapisa, koristi se kursor za iterativnu obradu svakog zapisa zasebno. Kursori su podržani u okviru pohranjenih procedura i funkcija. MySQL omogućava slijedni dohvat od prvog do zadnjeg zapisa, pri čemu se kursor ne može koristiti za čitanje prethodnih zapisa niti skočiti na točno određeni zapis.

Kada koristiti kursor?

U pravilu se koriste kada u svakoj iteraciji dohvata n-torke postoji potreba za obradom zapisa u više tablica. Također se koriste kada se određene operacije ne moraju izvršiti nad svim dohvaćenim zapisima već samo nad onima koji zadovoljavaju određeni uvjet.

Kada ne koristiti kursor?

Svakako bi se trebalo izbjegavati korištenje kursora u slučajevima kada se može napisati SQL upit koji je u mogućnosti obraditi sve potrebne zapise odjednom. Procedura se optimizira ako se umjesto kursora koristi složeniji SQL upit, naravno u slučajevima kada je to izvedivo. Nuspojava korištenja kursora je usporavanje izvođenja procedure jer se vrti petlja unutar koje se prilikom svake iteracije izvršava niz određenih naredbi. Druga je nuspojava složenost koda i otežano debagiranje za procedure koje koriste kursor.

Deklaracija kursora

Osnovna sintaksa:

```
DECLARE ime_kursora CURSOR FOR SELECT_izraz;
```

Deklaracija kursora služi za definiranje skupa podataka kroz koji će se kursorom iterativno prolaziti. Kursor se deklarira na način da se nakon dodjeljivanja imena definira SELECT naredba koja definira skup n-torki. Za kursor nije nužno da dohvaća n-torke iz jedne tablice, već one koje su rezultat SELECT naredbe koliko god ona složena bila. Važno je da se kursor deklarira na početku procedure odnosno funkcije i to prije prve naredbe, ali nakon što su deklarirane sve lokalne varijable koje će se koristiti u samoj proceduri ili funkciji.

Primjerice potrebno je obaviti određene operacije nad zapisima u tablici *radnik*. Kursor će biti deklariran na sljedeći način:

```
DECLARE mojKursor CURSOR FOR
        SELECT sifRadnik, pbrStan FROM radnik;
```

Otvaranje kursora

Osnovna sintaksa:

```
OPEN ime_kursora;
```

Nakon deklaracije kursor je potrebno otvoriti naredbom OPEN. Time se pokreće SELECT naredba definirana u deklaraciji te se podaci dohvaćaju u kontejner zapisa, kroz kojeg će se iterativno prolaziti i čitati podatke.

Dohvat podataka iz kursora

Osnovna sintaksa:

```
FETCH ime_kursora INTO niz_varijabli;
```

Naredbom FETCH dohvaćaju se podaci iz pojedine n-torke koja je dio skupa rezultata dobivenih otvaranjem kursora. Kursor funkcionira kao pokazivač koji pokazuje na trenutnu n-torku u skupu rezultata, međutim on ne zna sam u kojem trenutku treba dohvatiti sljedeću n-torku. Mehanizam kursora funkcionira na način da nakon što se kursor otvori, zavrti se petlja te se prilikom svake iteracije petlje naredbom FETCH dohvaća podatke iz trenutne n-torke. Podaci će se tom prilikom pohraniti u lokalne varijable nad kojima će se obavljati daljnje operacije. Važno je deklarirati točno onoliko pomoćnih varijabli koliko se atributa čita SELECT naredbom definiranom u deklaraciji kursora.

Primjer:

```
FETCH mojKursor INTO sifRadnik_vrijednost, pbrStan_vrijednost;
```

U gornjem će primjeru kursor prilikom svake iteracije petlje dohvaćati šifru te poštanski broj stanovanja radnika koji se odnose na trenutnu n-torku u tablici *radnik* te ih pohraniti u lokalne varijable *sifRadnik_vrijednost* te *pbrStan_vrijednost*.

Primjer:

```
DECLARE mojKursor CURSOR FOR
        SELECT * FROM radnik;
```

Deklaracija kursora na ovakav način jest moguća ali nije preporučljiva jer zvjezdicom nismo točno definirali broj atributa koje dohvaćamo te ne možemo biti sigurni koliko je lokalnih varijabli potrebno koristiti prilikom FETCH naredbe.

Zatvaranje kursora

Osnovna sintaksa:

```
CLOSE ime_kursora;
```

Po završetku rada s kursorom potrebno ga je pomoću naredbe CLOSE zatvoriti čime će se osloboditi resursi zauzeti kursorom. Iako *engine* garantira da će prilikom završetka procedure odnosno funkcije resursi automatski biti oslobođeni, preporuka je ipak eksplicitno zatvoriti kursor naredbom CLOSE.

Primjer:

Potrebno je napisati proceduru koja će dohvatiti svaki kvar zasebno te 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 za provjeru izvođenja procedure*/
END; $$
DELIMITER ;
```

Primjer poziva procedure:

```
CALL etiketiraj();
```

Deklaracijom kursora odnosno naredbom OPEN u privremeni kontejner bit će povučeni sljedeći podaci (36 n-torki):

nazivKvar	satiKvar
Zamjena prednjeg fara	2
Popravak radilice	3
Zamjena klipa	4
Izmjena amortizera	2
Zamjena pragova	4
Zamjena rashladne tekućine	1
Zamjena cerade	3
Uštimavanje pokretnog krova	2
Popravak felgi	2
Popravak oštećenih mjesta na limu	4
Zamjena blatobrana	2
Ugradnja sportskog pojasa	2

Prilikom prve iteracije, kursor pokazuje na prvu n-torku

Sadržaji atributa nazivKvar i satiKvar u svakoj se iteraciji upisjuju u varijable nazivKvar_vrijednost i satiKvar_vrijednost

…

Petlja LOOP izvodit će se beskonačno puta odnosno dok ne dođe do posljednjeg zapisa u dohvaćenom skupu podataka (u ovom primjeru tablici *kvar*). Iteriranjem će se 36 puta ispisati naziv

kvara te oznaka 'mali' odnosno 'veliki'. Primijetiti da s obzirom da je SELECT naredba pozvana 36 puta, dobit ćemo i 36 skupova rezultata. Rezultat izvođenja poziva procedure je u nastavku:

ime	velicina
Zamjena prednjeg fara	mali

ime	velicina
Popravak radilice	mali

ime	velicina
Zamjena klipa	mali

ime	velicina
Izmjena amortizera	mali

...

Kada započne iteracija s rednim brojem 37, naredba FETCH neće uspjeti dohvatiti podatak te će izvođenje rezultirati ispisom sljedeće poruke:

```
Query: CALL etiketiraj()

Error Code: 1329
No data - zero rows fetched, selected, or processed

Execution Time : 0.005 sec
Transfer Time  : 0.057 sec
Total Time     : 0.062 sec
```

S obzirom da procedura nije uspjela dohvatiti sljedeći podatak, nasilno se prekinulo njezino izvođenje. Dokaz za ovo je neizvršavanje naredbe SELECT 'kraj' – posljednje što je procedura vratila jest 'mali' kvar naziva 'Štimanje ventila'.

ime	velicina
Štimanje ventila	mali

Ovaj je problem moguće riješiti tako da se prvo prebroji koliko je n-torki dohvaćeno prilikom otvaranja kursora te se nakon toga zavrti petlja točno toliko puta (primjerice REPEAT…UNTIL, ili WHILE…DO).

```
DELIMITER $$
DROP PROCEDURE IF EXISTS etiketiraj$$
CREATE PROCEDURE etiketiraj()
BEGIN
        DECLARE naziv_vrijednost VARCHAR(255);
        DECLARE sati_vrijednost, n INT;
        DECLARE i int;
        DECLARE kur CURSOR FOR SELECT nazivKvar, satiKvar FROM kvar;
        SET i=0;
        SELECT COUNT(*) INTO n from kvar;
        OPEN kur;
        WHILE i<n do
                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;
                set i=i+1;
        END WHILE;
        CLOSE kur;
        SELECT 'kraj';
END; $$
DELIMITER ;
```

Primjer poziva procedure:

```
CALL etiketiraj();
```

U ovom će se primjeru petlja iterirati točno 36 puta te nakon izvršavanja petlje neće biti nikakve pogreške niti prekida, nego će procedura nastaviti s izvođenjem te ispisati 'kraj'.

Drugi način izbjegavanja nasilnog prekida procedure i prilikom poruke o pogreški za nemogućnost dohvata iduće n-torke je hvatanjem oznake o pogreški te reagiranjem na odgovarajući način uz pomoć *handler-a*.

CONDITIONI I HANDLERI

Handler je objekt u bazi podataka koji služi za hvatanje poruke o određenoj pogreški nakon čega reagira na definirani način. Prilikom deklaracije *handler-a* obavezno je definirati na koju će se pogrešku *handler* odnositi, što se može učiniti na dva načina. Prvi je način definiranje uvjeta pomoću CONDITION klauzule. Drugi je način umetanje oznake o pogreški u sam *handler*, što će biti pokazano kasnije. Uvjet u kojem se *handler* aktivira preporuča se koristiti zbog dodjeljivanja imena uvjetu što će rezultirati kasnijom boljom preglednošću procedure, pogotovo u slučajevima kada je potrebno definirati više *handler-a* za više različitih uvjeta.

Deklaracija condition-a

Osnovna sintaksa:

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value | mysql_error_code
```

CONDITION je potrebno deklarirati prije deklaracije kursora i *handler-a*. Deklarira se za odrađeni uvjet koji može biti SQLSTATE vrijednost ili *mysql_error_code*. SQLSTATE vrijednost je string od pet znamenaka koji je ANSI SQL standardiziran. SQLSTATE vrijednosti mogu se grupirati na sljedeći način:

- Kod za uspješno izvršavanje – '00000' odnosno '0' (ne preporuča se koristiti)
- SQLWARNING = SQLSTATE koji počinje sa '01'

- NOT FOUND = SQLSTATE koji počinje sa '02' (relevantno u kontekstu kursora, definira se ponašanje za trenutak kada kursor dođe do posljednje n-toke u skupu podataka)
- SQLEXEPTION = sve što ne počinje sa '00', '01' ili '02'

Kodovi svih MySQL pogrešaka mogu se pronaći na http://dev.mysql.com/doc/refman/5.6/en/error-messages-server.html . Većina pogrešaka može se definirati na oba navedena načina. U teoriji, korištenje SQLSTATE vrijednosti koja je univerzalna za SQL jezik, rezultirat će da je sam kod lakše prenijeti na druge platforme i samim time bit će bolji izbor. S druge je strane u praksi mala vjerojatnost da će se pohranjeni zadaci prenositi na drugi RDBMS. Jezik pohranjenih procedura kod Oracle i SQL Servera u potpunosti je nekompatibilan s MySQL-om, dok je DB2 djelomično kompatibilan. Također je činjenica da svi MySQL kodovi pogreške nemaju odgovarajuću SQLSTATE vrijednost pa je preporuka da se *condition-i* radije deklariraju za određeni MySQL kod za pogrešku.

Primjer:

U nastavku su navedene dvije moguće deklaracije koje će se odnositi na istu pogrešku, prvi put deklarirano pomoću koda MySQL pogreške, a drugi put pomoću SQLSTATE vrijednosti.

```
DECLARE err_table_exists CONDITION FOR 1050;
```

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

Deklaracija handler-a

Nakon što je definiran uvjet u kojem se *handler-a* pokreće, potrebno je deklarirati i sam *handler-a* za zadani tip pogreške. *Handler* je potrebno deklarirati nakon deklaracije varijabli i *condition-a*.

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action:
    CONTINUE
  | EXIT
  | UNDO

condition_value:
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

U *handler_action* dijelu definirana je akcija koja se obavlja prilikom pokretanja *handler-a*:

- CONTINUE – nastavlja se izvođenje trenutnog programa
- EXIT – prekida se izvođenje svih naredbi definiranih unutar BEGIN … END bloka
- UNDO – bit će implementiran u nekim idućim verzijama MySQL-a

U nastavku je definiran *handler* koji reagira na pogrešku definiranu prethodnim *err_table_exists* uvjetom.

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

U slučaju događaja pogreške *err_table_exists*, globalna se varijabla @table postavlja na vrijednost -1, odnosno čitanjem vrijednosti te varijable u samoj proceduri može se utvrditi je li došlo do pogreške 1050. Operacije koje je potrebno izvršiti prilikom aktiviranja *handler-a* definirane su unutar BEGIN … END bloka. Ako će se izvršiti samo jedna operacija tada je moguće izostaviti BEGIN … END oznake.

Osim što je moguće za svaki uvjet definirati *condition* i dodijeliti mu odgovarajuće ime, deklaraciju *condition-a* i *handler-a* moguće je objediniti u samo jednu deklaraciju. Gornji primjeri deklaracije *condition-a* i *handler-a* tada će izgledati kao u nastavku:

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

Primjeri deklaracije i upotrebe handlera:

1. Ako se pojavi bilo koja pogreška (osim NOT FOUND), nastavlja se s izvođenjem pohranjenog zadatka uz postavljanje varijable *l_error* na vrijednost 1:

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SET l_error=1;
```

2. Ako se pojavi bilo koja pogreška (osim NOT FOUND), prekida izvršavanje trenutnog bloka pohranjenog zadatka izvođenjem ROLLBACK naredbe te ispisivanjem poruke o pogreški:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Dogodila se pogreška – završavam s radom';
    END;
```

3. Ako se pojavila pogreška 1062 (duplicirana vrijednost ključa), nastavlja se s izvođenjem pohranjenog zadatka uz ispis odgovarajuće pogreške:

```
DECLARE CONTINUE HANDLER FOR 1062
        SELECT 'Pogreška dupliciranog unosa';
```

4. Ako se pojavila SQLSTATE pogreška 23000 (duplicirana vrijednost ključa), nastavlja se s izvođenjem pohranjenog zadatka uz ispis odgovarajuće pogreške:

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
        SELECT 'Pogreška dupliciranog unosa';
```

5. Ako kursor naredbom FETCH nije uspio dohvatiti podatak, nastavlja se s izvođenjem ostatka pohranjenog zadatka uz postavljanje varijable *l_error* na vrijednost 1:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_done=1;
```

Ili

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
        SET l_done=1;
```

Ili

```
DECLARE CONTINUE HANDLER FOR 1329
        SET l_done=1;
```

Primjer:

Koristeći *condition-e* i *handler-e*, procedura iz prethodnog poglavlja izgledat će kao u nastavku (procedura će dohvatiti svaki kvar zasebno te 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, n INT;
        DECLARE error BOOL DEFAULT FALSE;
        DECLARE kur CURSOR FOR SELECT nazivKvar, satiKvar FROM kvar;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET error=TRUE;
        OPEN kur;
        petlja:LOOP
                FETCH kur INTO naziv_vrijednost, sati_vrijednost;
                IF error=TRUE THEN /*provjerava pojavu pogreške NOT FOUND*/
                        LEAVE petlja;
                END IF;
                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';
END; $$
DELIMITER ;
```

Primjer poziva procedure:

```
CALL etiketiraj();
```

U slučaju pogreške NOT FOUND koja će se pojaviti nakon što je kursor obradio posljednju n-torku iz skupa podataka, odnosno prilikom dohvata iduće nepostojeće n-toke, aktivirat će se continue *handler* postavljajući zastavicu *error* na vrijednost TRUE. Ako je s procedurom sve u redu, *error* ima pretpostavljenu vrijednost FALSE. Unutar same petlje preporuča se provjeravanje zastavice *error* odmah nakon dohvata n-torke pomoću naredbe FETCH kako bi *handler* reagirao u odgovarajućem trenutku. U primjeru je to učinjeno pomoću IF uvjeta, kojim se definira da ako je došlo do pogreške NOT FOUND (error=TRUE), tada se izlazi iz petlje i nastavlja s ostalim naredbama u samoj proceduri.

Primjer:

Potrebno je napisati proceduru koja svim radnicima koji stanuju u mjestu sa zadanim poštanskim brojem povećava koeficijent plaće za 10%. Procedura mora vratiti broj dohvaćenih te broj obrađenih n-torki.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS povecajKoef $$
CREATE PROCEDURE povecajKoef(IN zadanoMjesto INT)
BEGIN
        DECLARE zapis_nije_pronaden, dohvaceno, obradeno INT DEFAULT 0;
        DECLARE trenutniRadnik, trenutniPbrStan INT DEFAULT NULL;
        DECLARE mojKursor CURSOR FOR SELECT sifRadnik, pbrStan FROM radnik;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET zapis_nije_pronaden=1;
        OPEN mojKursor;
        SELECT FOUND_ROWS()INTO dohvaceno; /*odnosi se naprethodnu naredu  OPEN,
broji n-torke koje su u skupu podataka nastalom prilikom otvaranja kursora*/
        ponavljaj:LOOP
                FETCH mojKursor INTO trenutniRadnik, trenutniPbrStan;
                IF zapis_nije_pronaden THEN
                        LEAVE ponavljaj;
                END IF;
                IF trenutniPbrStan=zadanoMjesto
                        THEN UPDATE radnik SET koefPlaca=koefPlaca*1.1
                                WHERE sifRadnik=trenutniRadnik;
                        SET obradeno=obradeno+1; /*broji koliko je puta izvršena
naredba update*/
                END IF;
        END LOOP ponavljaj;
        CLOSE mojKursor;
        SELECT dohvaceno AS 'dohvaceno zapisa', obradeno AS 'obradeno zapisa';
END; $$
DELIMITER ;
```

Primjer poziva procedure:

```
CALL povecajKoef(10000);
```

Uvedena je funkcija FOUND_ROWS() koja dohvaća broj n-torki koje su rezultat prethodnog SQL upita. U ovom će slučaju dohvatiti broj n-torki vraćenih naredbom OPEN mojKursor te ga pohraniti u varijablu *dohvaceno*. Točno će se toliko iteracija petlje izvšiti koliko je zapisano u varijabli *dohvaceno*. Ako je uspješno izvršen select upit iz deklaracije kursora, taj bi broj trebao odgovarati ukupnom broju radnika.

U varijabli *obradeno* koja se inkrementira prilikom izvršavanja UPDATE naredbe nalazit će se podatak o tome kolikom je broju radnika plaća povećana. Sobzirom da kursor dohvaća sve radnike, a neće se svim radnicima povećati plaća već samo onima iz zadanog mjesta, varijable *dohvaceno* i *obradeno* imat će različite vrijednosti.

Rezultat izvođenja procedure za ulazni parametar 10000:

dohvaceno zapisa	obradeno zapisa
98	51

Sama se procedura može optimizirati na način da se prilikom deklaracije kursora definira da on odmah prolazi isključivo kroz one radnike koji žive u zadanom mjestu. Time će se smanjiti broj iteracija petlje i ubrzati izvođenje procedure.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS povecajKoef $$
CREATE PROCEDURE povecajKoef(IN zadanoMjesto INT)
BEGIN
        DECLARE zapis_nije_pronaden, dohvaceno, obradeno INT DEFAULT 0;
        DECLARE trenutniRadnik INT DEFAULT NULL;
        DECLARE mojKursor CURSOR FOR SELECT sifRadnik FROM radnik WHERE
pbrStan=zadanoMjesto;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET zapis_nije_pronaden=1;
        OPEN mojKursor;
        SELECT FOUND_ROWS() INTO dohvaceno;
        ponavljaj:LOOP
                FETCH mojKursor INTO trenutniRadnik;
                IF zapis_nije_pronaden THEN
                        LEAVE ponavljaj;
                END IF;
                UPDATE radnik SET koefPlaca=koefPlaca*1.1 WHERE
sifRadnik=trenutniRadnik;
                SET obradeno=obradeno+1;
        END LOOP ponavljaj;
        CLOSE mojKursor;
        SELECT dohvaceno AS 'dohvaceno zapisa', obradeno AS 'obradeno zapisa';
END; $$
DELIMITER ;
```

Primjer poziva procedure:

```
CALL povecajKoef(10000);
```

Rezultat izvođenja procedure za ulazni parametar 10000:

dohvaceno zapisa	obradeno zapisa
51	51