Kurzory a triggery v PL/SQL

Kurzory

Než začnete pracovat s následujícími příklady, vytvořte si tabulku OSOBY a naplňte ji daty :

```
DROP TABLE osoby;

CREATE TABLE osoby (
  os_cislo  NUMBER(5),
  prijmeni  VARCHAR2(30),
  jmeno     VARCHAR2(30)
);

INSERT INTO osoby VALUES (1, 'Otta', 'Max');
INSERT INTO osoby VALUES (2, 'Pesicka', 'Lada');
INSERT INTO osoby VALUES (3, 'Rohlik', 'Ondra');

COMMIT;

SET SERVEROUTPUT ON
```

Kurzor je abstraktní datový typ umožňující procházet záznamy vybrané dotazem, který je s kurzorem spojen. Nad kurzorem jsou definovány následující operace :

- `OPEN` - otevření kurzoru
- `FETCH` - přečtení dalšího záznamu
- `CLOSE` - zavření kurzoru

Kromě těchto operací obsahuje kurzor navíc následující stavové proměnné :

- `%FOUND` - BOOLEAN; TRUE pokud FETCH vybral další záznam, FALSE pokud FETCH nic nevybral. Před prvním FETCH je NULL.
- `%NOTFOUND` - BOOLEAN; opak %FOUND
- `%ISOPEN` - BOOLEAN; TRUE pokud byl kurzor otevřen příkazem OPEN, jinak FALSE.
- `%ROWCOUNT` - INTEGER; počet řádků dosud zpracovaných příkazem FETCH. Před otevřením kurzoru nebo před prvním FETCH je 0.

Deklaraci a typické použití kurzoru ukazuje následující příklad :

```
DECLARE
  tmp  osoby%ROWTYPE;
  CURSOR  plist IS
  SELECT * FROM osoby;

BEGIN
  OPEN plist;

  LOOP
    FETCH plist INTO tmp;
    EXIT WHEN plist%NOTFOUND;
    dbms_output.put_line(plist%ROWCOUNT||'. '||tmp.jmeno||' '||tmp.prijmeni);
  END LOOP;

  CLOSE plist;
END;
```

Kurzor s parametry

Kurzor může mít i parametry, které musí být všechny vstupního typu. Proto zápis typu parametru pomocí klíčového slova `IN` je nepovinný :

```
DECLARE
  num INTEGER;

  CURSOR osoba (name IN VARCHAR2) IS
  SELECT os_cislo
    FROM osoby
   WHERE jmeno = name;

BEGIN
  OPEN osoba('Max');

  FETCH osoba INTO num;

  IF osoba%FOUND THEN
    dbms_output.put_line('Osobni cislo : '||num);
  END IF;

  CLOSE osoba;
END;
```

S kurzory lze pracovat i jednodušším způsobem tak, že je spojíme s příkazem `FOR - LOOP`. V cyklu jsou postupně vybrány všechny záznamy kurzoru. Všimněte si, že proměnná cyklu není deklarována a její struktura odpovídá struktuře řádky vybrané kurzorem :

```
DECLARE
  CURSOR plist(num IN INTEGER) IS
  SELECT *
    FROM osoby
   WHERE os_cislo > num;

BEGIN
  FOR p IN plist(1) LOOP
    dbms_output.put_line(p.jmeno||' '||p.prijmeni);
  END LOOP;
END;
```

Možné je i následující použití, kde dotaz kurzoru reprezentuje rozsah cyklu `FOR` :

```
BEGIN
  FOR p IN (SELECT * FROM osoby) LOOP
    dbms_output.put_line(p.jmeno||' '||p.prijmeni);
  END LOOP;
END;
```

Kontrolní úloha

Vytvořte proceduru `kopiruj_interval` se vstupními parametry (*dolní a horní mez*), která po zadání mezí zkopíruje takové hodnoty z tabulky `POKUS` (viz předchozí příklad), kde je rozsah klíčové položky v zadaném intervalu.

Procedura dále doplní hodnoty uživatele, tj. kdo ji spustil (pseudoproměnná `USER`) a aktuální datum (pseudoproměnná `SYSDATE`) kopírování. Kopírované hodnoty se budou ukládat do tabulky `POKUS_INTERVAL`, která má následující strukturu:

- `kod` (celé číslo, primární klíč)
- `hodnota` (textová položka max. délky 50 znaků)
- `datum_kopirovani` (datum)
- `uzivatel` (textová položka max. délky 50 znaků)

Triggery

Trigger (česky spínač) lze definovat příkazem `CREATE TRIGGER`, který má následující syntaxi :

```
CREATE [OR REPLACE] TRIGGER jméno triggeru
BEFORE | AFTER
DELETE | INSERT | < UPDATE [OF jméno sloupce]>
ON jméno tabulky
[REFERENCING < OLD AS jméno | NEW AS jméno >]
[FOR EACH ROW]
[WHEN podmínka]
BEGIN
 < ... PL/SQL blok ... >
END;
```

Triggery jsou vlastně procedury, které automaticky volá systém řízení báze dat při definované události. Touto událostí může být buď vložení (`INSERT`), rušení (`DELETE`) nebo aktualizace (`UPDATE`) záznamu v tabulce. Triggery bývají obvykle volány buď :

- před specifikovanou událostí - `BEFORE` nebo
- po specifikované události - `AFTER`

Nadefinovaný trigger může být buď spuštěn jednou při vkládání do tabulky nebo pro každý vkládaný záznam znovu. Například při :

```
INSERT INTO tab1 SELECT * FROM tab2;
```

se trigger spustí

- jen jednou, pokud je označen jako *tabulkový*
- pro každý vkládaný záznam znovu je označen jako *řádkový* - `FOR EACH ROW`

U triggeru lze rovněž specifikovat podmínku kdy má být vykonáno jeho tělo (PL/SQL blok) a to použitím klauzule `WHEN`.

Vlastní obsluhu události lze nadefinovat v PL/SQL bloku. Uvnitř PL/SQL bloku (a také v klauzuli `WHEN`) *řádkového triggeru* se lze odkazovat na původní a nový záznam pomocí pseudoproměnných `:new` a `:old`, kde `:new` obsahuje vkládaný záznam a `:old` původní záznam (tedy např. :new.plat < 2000). Je zřejmé, že při vkládání nového záznamu není definována pseudoproměnná `:old` a analogicky při mazání `:new`. Jejich jména lze předefinovat v klauzuli `REFERENCING OLD AS`, popř. `REFERENCING NEW AS`.

Tabulkové triggery

Chceme logovat jednotlivé změny prováděné v databázi, která obsahuje tabulky `PRACOVISTE` a `ZAMESTNANCI`, do tabulky `LOGY`. Při každé operaci (`INSERT`, `UPDATE`, `DELETE`) s tabulkami `PRACOVISTE` a `ZAMESTNANCI` se bude do tabulky `LOGY` ukládat název tabulky a typ operace (`I`, `U`, `D`) :

```
CREATE TABLE logy (
  tab  VARCHAR2(20),
```

```
  op   CHAR(1)
);

CREATE TRIGGER tai_pracoviste
AFTER INSERT ON pracoviste
BEGIN
  INSERT INTO logy VALUES ( 'PRACOVISTE', 'I');
END;
/

CREATE TRIGGER tau_pracoviste
AFTER UPDATE ON pracoviste
BEGIN
  INSERT INTO logy VALUES ( 'PRACOVISTE', 'U');
END;
/

CREATE TRIGGER tad_pracoviste
AFTER DELETE ON pracoviste
BEGIN
  INSERT INTO logy VALUES ( 'PRACOVISTE', 'D');
END;
/
```

Pro tabulku `ZAMESTNANCI` vytvoříme odpovídající triggery analogicky. Z názvu (identifikátoru) triggeru musí být patrné, pro jakou operaci, nad kterou tabulkou a kdy je spouštěn.

Řádkové triggery

Z předchozího příkladu chceme mít u každého záznamu informaci, kdy byl zadán a jaký uživatel jej zadal. K tomuto účelu rozšíříme tabulky `PRACOVISTE` a `ZAMESTNANCI` o sloupce `ZADAL` a `DATUM` a vytvoříme odpovídající **řádkové triggery** :

```
CREATE TRIGGER trbi_pracoviste
BEFORE INSERT ON pracoviste
FOR EACH ROW
BEGIN
  :new.zadal := user;
  :new.datum := sysdate;
END;
/

CREATE TRIGGER trbi_zamestnanci
BEFORE INSERT ON zamestnanci
FOR EACH ROW
BEGIN
  :new.zadal := user;
  :new.datum := sysdate;
END;
/
```

Poznámka: v SŘBD Oracle jsou definovány pseudoproměnné `user` a `sysdate`, ze kterých lze získat aktuální uživatelské jméno a čas.

Business rules triggery

Triggery jsou také často používány při realizaci tzv. *"business rules"*, tj. integritních omezení specifických pro danou oblast použití. Následující příklad ukazuje část pomyslné studijní agendy, ve

které si mohou studenti zapsat maximálně 20 kreditů za semestr. Protože tento požadavek se může měnit, je zavedena tabulka OMEZENI, ve které jsou uloženy všechny potřebné hraniční hodnoty.

Nejdříve založíme tabulku OMEZENI a naplníme požadovanými daty ...

```
CREATE TABLE omezeni (
  typ          VARCHAR2(30) NOT NULL,
  min_hodnota  NUMBER(5)    NOT NULL,
  max_hodnota  NUMBER(5)    NOT NULL
);

INSERT INTO omezeni VALUES ('KRED_SEMESTR', 10, 20);
INSERT INTO omezeni VALUES ('KRED_STUDIUM', 300, 320);

COMMIT;
```

... potom vytvoříme tabulku ZAPIS ...

```
CREATE TABLE zapis (
  os_cislo VARCHAR2(10) NOT NULL,
  predmet  VARCHAR2(10) NOT NULL,
  kredity  NUMBER(2)    NOT NULL
);
```

a nakonec na tabulku ZAPIS "pověsíme" business trigger :

```
CREATE OR REPLACE TRIGGER trbi_zapis
BEFORE INSERT
    ON zapis
   FOR EACH ROW

DECLARE
  suma    INTEGER;
  maximum INTEGER;
  errno   INTEGER;
  errmsg  VARCHAR2(256);

BEGIN
  SELECT SUM(kredity) INTO suma
    FROM zapis
   WHERE os_cislo = :new.os_cislo;

  SELECT max_hodnota INTO maximum
    FROM omezeni
   WHERE typ = 'KRED_SEMESTR';

  IF ((suma + :new.kredity) > maximum) THEN
    errno  := -20001;
    errmsg := 'Prekroceno maximum '||TO_CHAR(maximum)||' kreditu za semestr !';
    raise_application_error(errno, errmsg);
  END IF;
END;
```

Pokud při vkládání dat do tabulky zapis překročíme maximální povolenou hodnotu, dostaneme následující chybové hlášení :

```
ORA-20001: Prekroceno maximum 20 kreditu za semestr !
ORA-06512: at "ZIMA.TRBI_ZAPIS", line 19
ORA-04088: error during execution of trigger 'ZIMA.TRBI_ZAPIS'
```

Aktivace a deaktivace triggeru

Trigger je možno deaktivovat (zakázat) :

```
ALTER TRIGGER jméno DISABLE;
```

a opět aktivovat (povolit) :

```
ALTER TRIGGER jméno ENABLE;
```

Zrušení triggeru

Zrušení triggeru je realizováno příkazem `DROP TRIGGER`.

```
DROP TRIGGER jméno;
```

Sekvence

Často potřebujeme automaticky generovat hodnotu primárního klíče (identifikátor objektu, číslo pracoviště nebo zaměstnance). K tomuto účelu jsou určeny tzv. *sekvence*, které si lze představit jako sdílené globální čítače. Souběžný přístup k sekvenci je řešen automaticky. Syntaxe příkazu pro vytvoření sekvence je následující :

```
CREATE SEQUENCE jméno
   [INCREMENT BY] <celé číslo>   /* default 1 */
   [START WITH]   <celé číslo>   /* default 1 */
   [MAXVALUE]     <celé číslo>   /* default 10^27 */
   [MINVALUE]     <celé číslo>   /* default 1 */
   [CYCLE | NOCYCLE]             /* default NOCYCLE */
```

Význam jednotlivých klauzulí je následující :

- `INCREMENT BY` - hodnota o kterou má být čítač zvětšen (může být i záporná)
- `START WITH` - hodnota od které má čítač začít (hodí se chceme-li sekvenci použít pro generování klíče do tabulky, ve které jsou již data)
- `MAXVALUE` - nejvyšší povolená hodnota čítače
- `NOCYCLE` - při překročení `MAXVALUE` nastane chyba
- `CYCLE` - při překročení `MAXVALUE` začne čítač znova od `MINVALUE`

Každá sekvence obsahuje dva pseudosloupce, `CURRVAL` a `NEXTVAL`. Pseudosloupec `CURRVAL` poskytuje aktuální hodnotu čítače sekvence, `NEXTVAL` aktuální hodnotu čítače zvedne o požadovaný přírůstek a tuto hodnotu poskytuje. Vše je demonstrováno na následujících příkladech :

```
CREATE SEQUENCE test_seq MAXVALUE 10 NOCYCLE NOCACHE;

SELECT test_seq.NEXTVAL FROM DUAL;
SELECT test_seq.NEXTVAL FROM DUAL;

SELECT test_seq.CURRVAL FROM DUAL;

SELECT test_seq.NEXTVAL FROM DUAL;
```

V případě, že je sekvence nadefinována jako `NOCYCLE` a je překročena hodnota `MAXVALUE`, je vypsána chybová hláška :

```
ORA-08004: sequence TEST_SEQ.NEXTVAL exceeds MAXVALUE and cannot be instantiated
```

V tom případě nám pomůže jeden z následujících příkazů :

```
ALTER SEQUENCE test_seq CYCLE;
ALTER SEQUENCE test_seq MAXVALUE 1000;
ALTER SEQUENCE test_seq NOMAXVALUE;
```

Automatické číslování

Pomocí sekvence je automaticky generován identifikátor pracoviště (primární klíč) a vložen do záznamu řádkovým triggerem :

```
CREATE TABLE pracoviste (
  id_prac  NUMBER(5)    PRIMARY KEY,
  nazev    VARCHAR2(30) NOT NULL
);

CREATE SEQUENCE seq_id_prac MAXVALUE 99999;

CREATE TRIGGER trbi_pracoviste
BEFORE INSERT
    ON pracoviste
   FOR EACH ROW
BEGIN
   SELECT seq_id_prac.NEXTVAL INTO :new.id_prac FROM DUAL;
END;
/
```

Nyní můžeme otestovat funkčnost automatického číslování :

```
INSERT INTO pracoviste (nazev) VALUES ('KIV');
INSERT INTO pracoviste VALUES (NULL, 'CIV');
INSERT INTO pracoviste VALUES (10, 'FAV');
```

Kontrolní úloha

Vytvořte tabulky `DRUH_ZBOZI` a `ZBOZI`, které mají následující strukturu:

`DRUH_ZBOZI`:

- `druh` (textová položka max. délky 10, primární klíč)
- `min_cena` (povinná číselná položka)
- `max_cena` (povinná číselná položka)

`ZBOZI`:

- `id_zbozi` (celé číslo, primární klíč)
- `popis` (textová položka max. délky 50 znaků)
- `druh_zbozi` (textová položka max. délky 10, cizí klíč na tabulku `DRUH_ZBOZI`)
- `kupni_cena` (povinná číselná položka)

Vytvořte trigger, který v tabulce `ZBOZI`:

1. zajistí automatické číslování sloupce `id_zbozi` (použijte sekvenci)
2. kontroluje, zda cena zadávaného zboží je v intervalu (min_cena, max_cena) s ohledem na druh zboží