Kurzory a záznamy

Záznamy

- **Záznam** je kompozitní datový typ/struktura, která zapouzdřuje více položek, i různých datových typů.
- Příklad deklarace záznamu v deklarační sekci:

```
TYPE zamestnanec_rec IS RECORD (
  jmeno zamestnanec.jmeno%TYPE,
  id zamestnanec.id%TYPE
);
-- promenna typu zaznam (typu zamestnanec_rec)
v_zamestnanec zamestnanec_rec;
```

- Nebo lze přímo nadeklarovat proměnnou typu záznam odpovídající řádku tabulky:

```
v_zamestnanec zamestnanec%ROWTYPE;
```

Kurzory

- **Kurzor** je privátní pracovní oblast, kterou databázový server vytvoří pro každý SELECT – je spojen s dotazem typu SELECT a určen pro průchod výsledky. Kurzor lze vytvářet:
  - **implicitně** – automaticky databázovým serverem, nestaráme se o jeho otevření / uzavření.
  - **explicitně** – nadeklarujeme a vytvoříme ho přímo my.

Implicitní kurzory

- Jednoduché implicitní kurzory jsme už používali … SELECT … INTO ...
- Oracle automaticky provádí otevírání, zavírání implicitního kurzoru, průchod záznamy.
- PL/SQL vytváří implicitní kurzor také pro každý INSERT, UPDATE, DELETE, kde není možné použít explicitní kurzor (např. kurzor identifikující updatované řádky).
- Best Practice je používat implicitní kurzory. V posledních verzích Oracle jsou rychlejší než explicitní kurzory.

SELECT … INTO ...

```
declare
  v_zamestnanec zamestnanec%rowtype;
begin
  select * into v_zamestnanec from zamestnanec
  where zamestnanec_id = 1;
  dbms_output.put_line(v_zamestnanec.jmeno
                       || ' ' ||
                       v_zamestnanec.prijmeni);
end;
```

- Poznámka: Tímto způsobem se může vracet pouze právě jeden záznam.

CURSOR FOR LOOP

```
begin
  for zamestnanec_rec in (select * from zamestnanec)
  loop
    dbms_output.put_line(zamestnanec_rec.jmeno
                         || ' ' ||
                         zamestnanec_rec.prijmeni);
  end loop;
end;
```

- Poznámka: Jedná se o implicitní kurzor, který umí pracovat s více řádky. Navíc automaticky vytváří record na základě toho, co z kurzoru získáváme. Jedná se o jednoduché a velice výkonné řešení.

CURSOR FOR LOOP & explicit cursor

- Je také možné explicitně deklarovat kurzor a použít FOR … IN … LOOP pro jeho procházení:

```
declare
  cursor employees_cur is select * from zamestnanec;
begin
  for zamestnanec_rec in employees_cur
  loop
    dbms_output.put_line(zamestnanec_rec.jmeno
                         || ' ' ||
                         zamestnanec_rec.prijmeni);
  end loop;
end;
```

Kurzory s parametry

- Pro kurzor můžeme nadefinovat parametry, za které se dosadí konkrétní hodnoty při otvírání kurzoru. Parametry můžeme využívat v SELECTu, se kterým je kurzor svázán:

```
DECLARE
  v_zamestnanec zamestnanec%ROWTYPE;
  CURSOR k1 (v_jmeno_prefix VARCHAR2) IS
    SELECT jmeno, zamestnanec_id FROM zamestnanec
    WHERE jmeno LIKE (v_jmeno_prefix || '%');
BEGIN
  FOR v_zamestnanec IN k1 ('Mi')
  LOOP
    DBMS_OUTPUT.PUT_LINE('Jméno: ' || v_zamestnanec.jmeno
      || ', Id: ' || v_zamestnanec.zamestnanec_id);
  END LOOP;
END;
```

Explicitní kurzory

- Základní kroky při práci s explicitním kurzorem:
  - Deklarace kurzoru (CURSOR - IS),
  - otevření kurzoru (OPEN),
  - výběr dat prostřednictvím kurzoru (FETCH - INTO),
  - uzavření kurzoru (CLOSE).

Explicitní kurzory – syntaxe základních kroků

```
-- V deklarační sekci (za DECLARE):
CURSOR <název kurzoru> IS <příkaz SELECT>;
-- Dále ve výkonné sekci (za BEGIN):
OPEN <název kurzoru>; -- provedení SELECTu
-- Pro postupné procházení záznamů se v cyklu volá:
FETCH <název kurzoru> INTO <seznam proměnných>;
-- Nakonec uzavření kurzoru:
CLOSE <název kurzoru>;
```

Pozor! Nezapomínejte kurzory uzavírat! Když se pokusíte znovu-otevřít již otevřený kurzor, dostanete chybu! Navíc je v databázi maximální hodnota počtu otevřených kurzorů per session (OPEN_CURSORS). A v úplně nejhorším případě je možné při neuzavírání kurzorů a zároveň nastavené vysoké hodnotě OPEN_CURSORS přetížit paměť serveru: http://www.asktheoracle.net/what-will-happen-if-we-do-not-close-a-cursor-in-plsql.html

Stav kurzoru

- Kdykoliv můžeme otestovat stav kurzoru pomocí atributů kurzoru:
  - **<název kurzoru>%ROWCOUNT** – pořadové číslo aktuálního procházeného záznamu (od 1); pokud zatím nebyl vybrán žádný záznam, má hodnotu 0,
  - **<název kurzoru>%FOUND** – pokud poslední příkaz FETCH načetl nějaká data, má hodnotu TRUE; jinak FALSE; používá se pro zjištění konce cyklu, ve kterém se iteruje přes záznamy; obdobně funguje **<název kurzoru>%NOTFOUND**,
  - **<název kurzoru>%ISOPEN** – vrací TRUE, pokud je kurzor otevřen.

Explicitní kurzory – příklad

```
DECLARE
  v_jmeno zamestnanec.jmeno%TYPE;
  v_id zamestnanec.zamestnanec_id%TYPE;
  CURSOR k1 IS SELECT jmeno, zamestnanec_id FROM zamestnanec;
BEGIN
  OPEN k1; -- otevreni kurzoru - provedeni SELECTu
  LOOP -- cyklus pro pruchod vyslednych zaznamu
    FETCH k1 INTO v_jmeno, v_id;
    DBMS_OUTPUT.PUT_LINE('Jméno: ' || v_jmeno || ', Id: '
      || v_id);
    EXIT WHEN k1%NOTFOUND;
  END LOOP;
  CLOSE k1;
END;
```

Parsování

- Parsování je interpretování SQL dotazu / příkazu a vytváření Execution plánu. Tento proces má mnoho fází (kontrolu syntaxe, oprávnění, generování Execution plánu, ukládání do shared pool atd.). Jsou dva typy parsování:
  - **Hard parsing**
    - SQL dotaz / příkaz je zavolán poprvé, není ve shared pool. Je nutné vyvolat všechny fáze.
  - **Soft parsing**
    - SQL dotaz / příkaz je nalezen ve shared poolu. Není nutné vyvolat všechny fáze, tudíž je výkonnější než hard parsing.

Kurzor

- Kurzory je možné zpřístupnit pomocí `v$sql`:

```
select executions, sql_text from v$sql
order by last_load_time desc;
```

- Existují i další způsoby zpřístupnění dalších informací:
  - http://smahamed.blogspot.cz/2011/06/difference-between-vsql-and-vsqlarea.html

Rozdílné SELECTy

- Vykonejte tyto SELECTy:

```
select * from hr.employees;
select * from hr.Employees;
select * from hr.EMPLOYEES;
```

- A poté zavolejte:

```
select executions, sql_text from v$sql
order by last_load_time desc;
```

- Vidíte, že se pro Oracle jedná o kompletně rozdílné SELECTy!
- Do této situace se můžete dostat když je SQL dotaz dynamicky generován, nebo když se nepoužívají proměnné.

Proměnné

- Důsledek: V SQL příkazu / dotazu byste nikdy neměli hard-kódovat proměnné.
  - http://www.akadia.com/services/ora_bind_variables.html

SELECT FOR UPDATE

- Příkaz SELECT FOR UPDATE umožňuje uzamknout záznamy načtené do result setu (množiny záznamů) kurzoru, aby je nemohla modifikovat jiná transakce (uživatel).
  - Následně můžeme chtít na zamčených záznamech provádět UPDATE, DELETE, aniž by záznamy mezi výběrem a UPDATEm/DELETEm mohl modifikovat někdo jiný.
  - Zamykání se používá např. při práci s BLOBy, kdy je potřeba nejdříve získat lokátor (adresu) BLOBu (uzamčený pro zápis, aby do BLOBu nemohl zapisovat nikdo jiný) a následně pomocí dalšího SQL příkazu plnit BLOB binárními daty.
- Záznamy jsou odemčeny při následujícím commitu nebo rollbacku.

```
CURSOR <jméno kurzoru> IS
   <příkaz SELECT>
   FOR UPDATE [of <seznam sloupců>] [NOWAIT];
```

WHERE CURRENT OF

- Slouží k syntakticky snadnému mazání/aktualizaci záznamů, které byly vybrány kurzorem pomocí příkazu SELECT FOR UPDATE (a uzamknuty pro zápis právě pro následnou změnu).
- Speciální klauzuli WHERE CURRENT OF můžeme použít v příkazu UPDATE nebo DELETE:

```
UPDATE <jméno tabulky>
SET <nastavení atributů>
WHERE CURRENT OF <jméno kurzoru>;

DELETE FROM <jméno tabulky>
WHERE CURRENT OF <jméno kurzoru>;
```

WHERE CURRENT OF – příklad

```
CREATE OR REPLACE FUNCTION FindCourse(course_name_in IN VARCHAR2)
RETURN NUMBER IS
  cnumber NUMBER; -- cislo kurzu vracene z funkce
  CURSOR c1 IS
    SELECT course_number
    FROM courses_tbl
    WHERE course_name = course_name_in
    FOR UPDATE OF instructor;
BEGIN
  -- Spusteni SELECTu pro dany kurzor, uzamknuti vyslednych radku
  OPEN c1;
  FETCH c1 INTO cnumber;
  IF c1%NOTFOUND THEN
    cnumber := 9999; -- cislo vracene, pokud kurz nebyl nalezen
  ELSE
    -- Kurz se zadanym jmenem byl nalezen
    -- Aktualizace radku vybraneho kurzorem
    UPDATE courses_tbl SET instructor = 'SMITH' WHERE CURRENT OF c1;
    -- Potvrzeni zmeny, uvolneni zamku
    COMMIT;
  END IF;
  CLOSE c1;
  RETURN cnumber;
END;
```

REF CURSOR I.

- Z procedury můžete vrátit kurzor pomocí SYS_REFCURSOR. Toto využijete vždy, když budete chtít vracet výsledek SELECTu z databáze.
- Příklad procedury:

```
CREATE OR REPLACE PROCEDURE find_by_prijmeni

(  res OUT SYS_REFCURSOR,

   vLastName IN zamestnanec.prijmeni%type  ) AS

BEGIN

  OPEN res FOR

    SELECT * FROM zamestnanec WHERE prijmeni = vLastName;

END find_by_prijmeni;
```

REF CURSOR II.

- Použití v PL/SQL:

```
declare
  l_cursor SYS_REFCURSOR;
  v_zamestnanec zamestnanec%ROWTYPE;
begin
  find_by_prijmeni(l_cursor, 'Boss');
  LOOP
    FETCH l_cursor INTO  v_zamestnanec;
    EXIT WHEN l_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_zamestnanec.jmeno);
  END LOOP;
  CLOSE l_cursor;
end;
```

REF CURSOR III.

- Použití v Javě spolu s Hibernate:
  - http://timezra.blogspot.cz/2008/10/spring-hibernate-and-oracle-stored.html
- Použití obecně:
  - http://www.oracle-base.com/articles/misc/using-ref-cursors-to-return-recordsets.php