Vnořené SQL

Vnořené SQL

Upozornění!

Než začnete pracovat na příkladech, vytvořte v databázi potřebné objekty pomocí skriptu EMBEDDED.SQL :

Princip vnořeného SQL spočívá v tom, že umožňuje do některého z vyšších programovacích jazyků vkládat příkazy SQL označené standardním prefixem. Ty jsou pak pomocí prekompilátoru přeloženy na volání funkcí knihovny, která realizuje spojení s databázovým serverem, překlad požadavků na server a odpovědí serveru. Soubor vytvořený prekompilátorem lze již přeložit překladačem konkrétního programovacího jazyka a připojit potřebné knihovny.

Následující příklady ukazují SQL vnořené do jazyka C. SQL příkazy jsou označeny prefixem **EXEC SQL**. Na začátku je nutné do zdrojového textu zahrnout hlavičkový soubor s prototypy knihovních funkcí pro komunikaci s databází a potřebné datové struktury příkazem :

```
EXEC SQL INCLUDE sqlca.h;
```

Tento soubor obsahuje (mimo jiné) definici datové struktury SQL Communiactions Area (SQLCA) přes níž probíhá komunikace s databází. Stejně jako příkazy vnořeného SQL, i tuto strukturu definuje standard SQL. Proměnné, které jsou používány jak v hostitelském jazyce (zde C), tak v SQL se nazývají **vazební proměnné** a musí se deklarovat ve zvláštní deklarační sekci ohraničené příkazy :

```
EXEC SQL BEGIN DECLARE SECTION
```

a

```
EXEC SQL END DECLARE SECTION
```

Přihlášení k databázi je umožněno příkazem :

```
EXEC SQL CONNECT login/heslo
```

Odhlášení z databáze s potvrzením, respektive odvoláním transakcí příkazy :

```
EXEC SQL COMMIT WORK RELEASE
```

respektive

```
EXEC SQL ROLLBACK WORK RELEASE
```

Celkovou strukturu programu ukazuje následující obrázek :

```
#include
EXEC SQL INCLUDE sqlca.h;
EXEC SQL BEGIN DECLARE SECTION;
  char uzivatel[30];
EXEC SQL END DECLARE SECTION;
```

```
int main(int argc, char **argv) {

  EXEC SQL CONNECT :uzivatel;

  EXEC SQL COMMIT WORK RELEASE;

  return OK;
}
```

Co se týče obsluhy výjimek, které mohou nastat při zpracování vnořeného SQL příkazu, je možné nastavit implicitní chování programu při chybě takto :

- program pokračuje ve zpracování instrukcí :

  ```
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  ```

- program pokračuje skokem na dané návěští :

  ```
  EXEC SQL WHENEVER SQLERROR GOTO navesti;
  ```

- program opustí právě zpracovávanou smyčku :

  ```
  EXEC SQL WHENEVER SQLERROR DO BREAK;
  ```

Jelikož se jedná o direktivu prekompilátoru, nesouvisí volání uvedených tří příkazů nijak se skutečným tokem instrukcí programu ! Prekompilátor postupně prochází program a pokud narazí na jednu z těchto direktiv, pak za každý vnořený SQL příkaz vkládá odpovídající konstrukci :

```
if (sqlca.sqlcode < 0) goto navesti;
```

nebo

```
if (sqlca.sqlcode < 0) break;
```

dokud nenarazí na další direktivu, která definuje jiné chování.

Kurzory

Práce s kurzory je ve vnořeném SQL prakticky stejná jako v PL/SQL. Následující příklad ukazuje použití jednoduchého kurzoru s parametrem : **Příklad 1**.

Stažený soubor musí být nejdříve editován (nutno doplnit login a heslo pro přihlášení do Oracle), dále přenesen (např. prostřednictvím WinSCP) na stroj, kde je databázový server instalován (tj. ARES) a poté :

- zpracován precompilerem jazyka C

  ```
  $proc example1.esql
  ```

  který vytvoří soubor *example1.c* v programovacím jazyce C
- soubor *example1.c* přeložen překladačem jazyka C

  ```
  $gcc -c example1.c
  ```

- sestavení spustitelného souboru linkerem překladače kazyka C

  ```
  $-L$ORACLE_HOME/lib -lclntsh example1.o -o example1
  ```

- a otestování funkčnosti programu, např.

```
$./example1 Jiricka
$./example1 Machacek
```

Manipulace s daty

Vkládání a mazání záznamů ukazuje **Příklad 2**.

Poměrně častý je případ, kdy potřebujeme vybrat určité záznamy a postupně provádět jejich aktualizaci. Vzhledem ke konkurenčnímu přístupu ke sdíleným datům, je nutné zmíněné záznamy nejprve uzamknout pro zápis (ostatní transakce je mohou číst) a pak teprve provádět aktualizaci. Záznamy jsou odemknuty po potvrzení nebo odvolání transakce.

Kurzor, který vybere a zamkne záznamy nadeklarujeme např. takto :

```
EXEC SQL DECLARE crsr_osoby CURSOR FOR
  SELECT  prijmeni, jmeno, plat, funkce
    FROM  osoby
   WHERE  plat < 2000
  FOR UPDATE;
```

Klauzule **FOR UPDATE** říká, že záznamy, které dotaz vybere, mají být uzamknuty pro zápis. Některé databázové systémy (např. Oracle) umožňují zakmnout i pouze některé položky vybraných záznamů. Budeme-li aktualizovat pouze položky *plat* a *funkce*, napíšeme :

```
EXEC SQL DECLARE crsr_osoby CURSOR FOR
  SELECT  prijmeni, jmeno, plat, funkce
    FROM  osoby
   WHERE  plat < 2000
  FOR UPDATE OF plat, funkce;
```

Aktualizaci záznamu, který je v daném kurzoru aktuální, provedeme takto :

```
EXEC SQL UPDATE osoby SET plat = ..., funkce = ...
         WHERE CURRENT OF crsr_osoby;
```

Detaily této úlohy ukazuje **Příklad 3**.

Dynamické SQL

Následující příklady ukazují, jak začlenit do hostitelského jazyka příkazy DDL (vytváření tabulek, pohledů, atd.) a jak dynamicky za běhu programu sestavit dotaz. Obě zmíněné úlohy mají společné to, že využívají tzv. **dynamické SQL**.

1. Pokud SQL příkaz není dotaz a neobsahuje vazební proměnné, lze použít konstrukce **EXEC SQL EXECUTE IMMEDIATE** :

   ```
   EXEC SQL EXECUTE IMMEDIATE
            CREATE TABLE osoby (
              os_cislo   NUMBER(5) PRIMARY KEY,
              prijmeni   VARCHAR2(30) NOT NULL,
              jmeno      VARCHAR2(30) NOT NULL
            );
   ```

   Lze spustit i SQL příkaz uložený jako řetězec ve vazební proměnné typu **VARCHAR** :

   ```
   EXEC SQL EXECUTE IMMEDIATE :command_string;
   ```

   Obě tyto možnosti jsou ukázány na **Příkladě 4**.
2. Pokud SQL příkaz není dotaz a obsahuje vazební proměnné (parametry), je postup následující :

```
prikaz := "INSERT INTO osoby (jmeno, prijmeni, plat) VALUES (:par1, :par2, 10000.00)"
EXEC SQL PREPARE p_vloz FROM :prikaz;

jmeno := "Honza"; prijmeni := "Cervenka"
EXEC SQL EXECUTE p_vloz USING :jmeno, :prijmeni;

jmeno := "Honza"; prijmeni := "Zelenka"
EXEC SQL EXECUTE p_vloz USING :jmeno, :prijmeni;
```

Tento postup najdete v **Příkladě 5**.

3. Pokud SQL příkaz je dotaz a obsahuje vazební proměnné, je nutné jej spojit s kurzorem :

```
prikaz := "SELECT plat FROM osoby WHERE jmeno = :par1 AND prijmeni = :par2";

EXEC SQL PREPARE p_vyber FROM :prikaz;
EXEC SQL DECLARE c_vyber CURSOR FOR :prikaz;

EXEC SQL OPEN c_vyber USING :jmeno, :prijmeni;
EXEC SQL WHENEVER NOT FOUND DO BREAK;

for (;;) {
  EXEC SQL FETCH c_vyber INTO :osoba;
  ...
}

EXEC SQL CLOSEEXEC SQL COMMIT WORK RELEASE;
```

Tato metoda je zřejmá z **Příkladu 6**.

PL/SQL

Vnořený PL/SQL blok je do hostitelského jazyka včleněn mezi direktivy

```
EXEC SQL EXECUTE
```

a

```
END-EXEC
```

jak je ukázáno na **Příkladu 7**.

Pozor ! U vnořeného PL/SQL prekompilátor vyžaduje sématickou kontrolu PL/SQL kódu a potřebuje se připojit do databáze (využívá její PL/SQL kompilátor a zároveň se tak již při překladu odhalí případné chyby v kódu, popř. odkazy na neexistující objekty v databázi). K tomu potřebuje uživatelské jméno a heslo, které předáte jako parametr příkazové řádky :

```
$proc sqlcheck=semantics userid=uzivatel/heslo soubor.esql
```

Současné připojení do několika databází

V praxi můžeme narazit na situace, kdy je nutné, aby jeden program současně přistupoval do několika databází najednou. Ve vnořeném SQL se tato situace řeší tak, že se nadeklaruje symbolické jméno databáze :

```
EXEC SQL DECLARE jmeno_db DATABASE;
```

a u každého vnořeného SQL příkazu se uvádí v jaké databázi má být proveden :

```
EXEC SQL AT jmeno_db CONNECT :uzivatel;

EXEC SQL AT jmeno_db SELECT count(*) INTO :pocet FROM osoby;
```

```
EXEC SQL AT jmeno_db COMMIT WORK RELEASE;
```

Metodika současné práce se dvěma databázemi je ukázána na **Příkladě 8**.

Uvedený způsob je použitelný, jsou-li předem známy databáze, se kterými bude program pracovat, stejně tak jako jejich počet. Následující příklad ukazuje "dynamickou" verzi předchozího. Rozdíl spočívá v tom, že nejsou deklarována symbolická jména databází, ale jsou uložena jako řetězce ve vazebních proměnných. Pak se mohou libovolně měnit databáze i jejich počet, viz **Příklad 9**.

Shrnutí

Vnořené SQL umožňuje poměrně snadné propojení vyššího programovacího jazyka s SQL, stejně jako rozšíření neprocedurálního SQL o procedurální rysy (data jsou uchována v databázi, algoritmus, který nelze vyjádřit pomocí SQL příkazů, je implementován ve vyšším programovacím jazyce). Nevýhodou je **neshoda datových typů** (*impedance mismatch*) SQL a hostitelského jazyka a fakt, že programátor se musí učit programovací jazyk navíc (popř. SQL).

Mnohem více informací o vnořeném SQL v jazyce C se dočtete v knize Pro*C/C++ Programmer's Guide .