Embedded SQL

Antonín Steinhauser

Obsah referátu

- Základní myšlenka embedded SQL
- Úvod, historie, přehled jazyků a databází
- Oracle Pro* C/C++
- Základy syntaxe
- Statické embedded SQL
- Hostitelské proměnné a indikátory
- Kurzory, ošetřování chyb
- Dynamické embedded SQL, SQLDA

Úvod a motivace

- Vkládání SQL příkazů do hostitelského programovacího jazyka
- Kombinace výpočetní síly programovacího jazyka s efektivním přístupem k databázi
- De-facto rozšíření SQL
- Hostitelský jazyk může být procedurální i neprocedurální, důležitá je jeho výpočetní síla

Embedded SQL versus SQL API

- Embedded SQL není jedinou možností, jak přistupovat z programu k databázi
- Zvláštní API na rozdíl od embedded SQL nepotřebuje prekompilaci
- Je tudíž lépe přenositelné ale obvykle méně přehledné než embedded SQL
- Například ODBC standard
- Preprocesor převadí embedded SQL na volání API funkcí

Historie embedded SQL

- První implementace IBM DB2 v první polovině 80. let
- Ta byla základem pro ANSI SQL v roce 1986, aktualizovaného 1989
- Oba tyto standardy obsahovaly pouze statické embedded SQL
- De-facto standardem dynamického embedded SQL se stala opět jeho implementace v DB2

Konkrétní varianty

- DB2 – C/C++, Cobol, Fortran, REXX
- Oracle – Ada, C/C++, Cobol, Fortran, Pascal, PL/1
- Microsoft SQL Server – od verze 2008 už nepodporuje embedded SQL
- MySQL – nepodporuje embedded SQL

Oracle Preprocesor

- Převádí embedded SQL pro Oracle do OCI (Oracle call interface)
- Společný pro všechny hostitelské jazyky
- Pro* C/C++, Pro* Cobol, Pro* Fortran

Pro* C/C++ versus OCI

- Větší kód
- Menší kontrola přístupu aplikace k databázi
- Nemožnost ručních optimalizací
- Lépe čitelný kód
- Jednodušší konstrukce programu

Pro* C/C++ - vytvoření programu

- *.pc – zdrojový kód v Pro* C/C++
- Oracle precompiler převede *.pc na *.c nebo *.cpp
- *.c, *.cpp – zdrojový kód v C nebo C++
- Kompilátor převede zdrojový kód na object *.o resp. *.obj
- Linker slinkuje objecty do přímo spustitelného binárního kódu

Syntaxe Pro* C/C++

- Základem je C/C++, Pro* C/C++ je pouze jeho rozšířením (stejně jako u ostatních hostitelských jazyků)
- Přidané příkazy zpracovávané Oracle precompilerem
- EXEC SQL ... ;

Syntaxe Pro* C/C++ - změny

- Nejsou přípustné jednořádkové komentáře dvojitým lomítkem, je nutno používat /* */
- Pro inkluzi jiného Pro* C/C++ zdrojového souboru je nutno použít příkaz EXEC SQL INCLUDE

Deklarativní a výkonné příkazy

- Syntaxe EXEC SQL dovoluje příkazy, které přímo pracují s databází – SELECT, INSERT, UPDATE, DELETE, ...
- Zahrnuje ovšem i další konstrukce, které se v SQL jinak nevyskytují – DECLARE, OPEN, FETCH, DESCRIBE, ...

Statické embedded SQL

- Starší a slabší varianta embedded SQL
- Je ovšem jednodušší a obvykle i rychlejší
- Odolná vůči SQL injection
- Pokud ke splnění účelu stačí, měla by být preferována před dynamickou formou

Hostitelské proměnné

- Výstupní – pouze příkaz SELECT (a později FETCH)
- Vstupní – všechny ostatní příkazy i SELECT
- Slouží programu pro předávání vstupních dat databázi a databázi pro vracení výsledků programu
- Nová klauzule SELECT INTO

Hostitelské proměnné II

- Mohou v SQL dotazu vyjadřovat pouze hodnoty nebo výrazy
- Nemohou vyjadřovat klauzule, názvy příkazů ani jména databázových objektů (tabulky, indexy, ...)
- Proměnná je v OCI převedena na hodnotu, proto nemá smysl používat hostitelské proměnné například v klauzuli ORDER

Hostitelské proměnné III

- Musí být deklarovány ve speciální sekci

EXEC SQL BEGIN DECLARE SECTION

...Hostitelské proměnné...

EXEC SQL END DECLARE SECTION

- V hostitelském jazyce se používají přímo v EXEC SQL sekcích s dvojtečkou - :proměnná

Datové typy v Pro* C/C++

- char - jeden znak
- char [n] – pole znaků
- int
- float
- VARCHAR [n] – řetezec proměnlivé délky
- Je ve skutečnosti strukturou o položkách arr a len

Indikátorové proměnné

- Nekompatibilita datových struktur programovacího jazyka a databáze
- Je řešena pomocnou hostitelskou proměnnou
- Může být přiřazena nejvýše jedna ke každé hostitelské proměnné
- Může být vstupní i výstupní
- Zapisuje se za další dvojtečku za hostitelskou proměnnou - :proměnná:indikátor

Indikátorové proměnné II

- Indikátorová proměnná je dvoubytové číslo – short
- Řeší především nekompatibilitu ternární logiky databáze s binární logikou C/C++
- Dále řeší přetečení velikosti datového typu hostitelské proměnné

Hodnoty indikátorů na vstupu

- -1 – Hodnota příslušné hostitelské proměnné je NULL, její skutečná hodnota bude databází ignorována
- 0 – Bude použita skutečná hodnota hostitelské proměnné

- NULL lze ovšem vložit do databáze i přímo konstantou NULL

Hodnoty indikátorů na výstupu

- -1 – Hodnota hostitelské proměnné je NULL, skutečná hodnota může být libovolná
- 0 – Hodnota hostitelské proměnné je platná
- >0 – Hodnota hostitelské proměnné je platná, ale byla zkrácena na velikost jejího datového typu, kladné číslo označuje její originální velikost
- -2 – Hodnota hostitelské proměnné byla zkrácena, její skutečná velikost je neznámá

Začátek a konec spojení

- Spojení musí v každém programu navázáno i ukončeno
- Pro navázání je nutno použít vstupní hostitelské proměnné – jsou dvě varianty
- EXEC SQL CONNECT :auth;, kde auth obsahuje username/password
- EXEC SQL CONNECT :username IDENTIFIED by :password

Začátek a konec spojení II

- Hostitelské proměnné nelze nahradit konstantními řetězci z bezpečnostních důvodů
- Konec spojení
- EXEC SQL COMMIT WORK RELEASE
- EXEC SQL ROLLBACK WORK RELEASE

Detekce chyb

- Má smysl v zásadě u každé
- Implicitně klauzulí WHENEVER
- WHENEVER <CONDITION> <ACTION>
- Podmínky:

SQLERROR – SQLCA.sqlcode is negative

SQLWARNING – SQLCA.sqlwarn[0] is set

NOT FOUND – SQLCA.sqlcode is positive

Detekce chyb II

- WHENEVER akce:

STOP – ukončí program

CONTINUE – program se pokusí přeskočit chybující příkaz a pokračovat

DO <function> - program vykoná obslužnou rutinu a pokračuje dalším příkazem

GOTO <label> - program skočí na návěští <label>

SQLCA

- Příznaková struktura Oracle OCI – program jí ale musí naincludovat, aby jí mohl používat
- EXEC SQL INCLUDE SQLCA.H
- WHENEVER je jen zapouzdřující klauzule pro čtení z této struktury
- Její přímé čtení umožňuje přesnější identifikaci chyb, varování a příznaků

SQLCA II

```
struct sqlca {
        char sqlcaid[8]; - obsahuje konstantu SQLCA
        long sqlabc; - délka struktury sqlca
        long sqlcode; - status kód posledního příkazu
        .....
        long sqlerrd[6];
        char sqlwarn[8];
        .....
};
```

Kursory

- Výstupní hostitelské proměnné nelze použít, pokud dotaz vrací více řádků
- Jednou možností je využití hostitelských polí
- Jejich omezením je maximální pevně definovaná velikost
- Univerzálnějším a silnějším prostředkem jsou kurzory

Kursory II

- Umožní vykonat na databázi dotaz vracející více sloupců, zafixovat jeho výsledek a procházet jej po jednotlivých řádcích
- Samotná hodnota kurzoru určuje offset ve výsledku
- Konec množiny výsledků se obvykle detekuje pomocí klauzule WHENEVER NOT FOUND
- Jména kursorů musí být unikátní, ale nejsou to proměnné

Kursory III

- DECLARE CURSOR cursorname FOR (QUERY) – deklaruje kursor pro určitý dotaz
- OPEN cursorname – provede dotaz definovaný kursorem, otevře množinu výsledků a nastaví kursor před první výsledek
- FETCH cursorname INTO :a, :b, ... - posune hodnotu o jeden výsledek a načte výsledek do výstupních hostitelských proměnných
- CLOSE cursorname – uzavře cursor

Scrollable cursors

- Normální kurzory umožňují pouze sekvenční přístup do množiny výsledků
- Pro návrat je nutné kursor uzavřít, znovu otevřít a doiterovat na požadovanou pozici
- Scrollable cursors umožňují náhodný přístup
- Příkazy OPEN a CLOSE jsou stejné jako u normálních kursorů
- Scrollable cursor se deklaruje pomocí DECLARE SCROLLABLE cursorname FOR

Scrollable cursors II

- FETCH příkaz mám několik variací podle různých požadavků na pohyb
- FETCH FIRST, FETCH LAST, FETCH PRIOR, FETCH CURRENT, FETCH NEXT – vždy posunou kursor a načtou prvek
- FETCH RELATIVE n, FETCH ABSOLUTE n – určují lokální resp. globální offset požadované položky

Zápis do databáze přes kursory

- Kursory lze využít i k manipulacím s nalezenými daty – příkazy UPDATE a DELETE
- Přidávají novou podmínku za WHERE – CURRENT OF cursorname
- V Oracle se změny automaticky nepromítají do aktuální množiny výsledků kursoru

Dynamické embedded SQL

- Ve statickém embedded SQL se prováděné příkazy, klauzule i databázové objekty určují již v době prekompilace a za runtimu není možné je modifikovat s výjimkou dosazování hodnot hostitelských proměnných
- Komplexnějším aplikacím, které neprovádějí jen omezenou předem specifikovanou množinu příkazů, to nestačí

Dynamické embedded SQL II

- Hodí se v případech, kdy se za jeho běhu dynamicky mění používané tabulky, sloupce, indexy, pohledy a další databázové objekty, klauzule nebo samotné dotazy
- Jeho použití je také nevyhnutelné v případě, kdy není předem znám počet nebo datové typy hostitelských proměnných

Dynamické embedded SQL III

- Dynamické SQL poskytuje oproti statickému možnost sestavovat celý dotaz až za runtime
- Nelze v něm provádět kompilační optimalizace, proto je obecně pomalejší a méně výkonné než statické SQL
- Je také náchylnější k chybám (např. SQL injection), složitější a méně přehledné

4 varianty dynamického SQL

- Existují 4 postupně se komplikující varianty použití dynamického SQL
- Je vhodné variantu s nejnižším číslem, která splňuje požadovaný úkol
- Čím vyšší číslo varianty, tím větší dynamiku varianta a tím je složitější – a tudíž méně přehledná a náchylnější k chybám

1. varianta dynamického SQL

- EXECUTE IMMEDIATE příkaz;
- Neumožňuje žádné hostitelské proměnné ani výstup příkazu
- Nelze jí tudíž použít pro dotaz SELECT
- Příkaz se najednou sestaví, zkompiluje a spustí
- Poměrně neefektivní metoda – hodí se pouze pro příkazy, které se neopakují

2. varianta dynamického SQL

- PREPARE statement FROM příkaz
- EXECUTE statement USING :var1, :var2
- Neumožňuje žádný výstup příkazu – je tudíž nepoužitelná pro SELECT
- Umožňuje použití vstupních hostitelských příkazů
- Dotaz se zkompiluje pouze jednou příkazem SELECT z řetězce reprezentujícího dotaz včetně hostitelských proměnných

2. varianta dynamického SQL II

- Nezáleží na jménech takovýchto hostitelských proměnných – rozhodující jsou proměnné v USING klauzuli příkazu EXECUTE, ale musí být unikátní
- Příkazem EXECUTE je možno jednou zkompilovaný příkaz spustit několikrát, volitelně s různými hodnotami proměnných
- Je potřeba předem znát počet a typ hostitelských proměnných

3. varianta dynamického SQL

- PREPARE statement FROM příkaz
- DECLARE c CURSOR FOR statement
- OPEN c USING :invar1, :invar2, ...
- FETCH c INTO :outvar1, :outvar2, ...
- CLOSE c

- Umožňuje vstupní i výstupní hostitelské proměnné

3. varianta dynamického SQL II

- Příkazem PREPARE se zkompiluje a zoptimalizuje příkaz jako ve 2. variantě
- Pak se na takto vytvořeném deklaruje kurzor a otevře se s příslušnými parametry příkazem OPEN
- Z otevřeného kurzoru se čte pomocí příkazů FETCH a kontrolou konce výsledku nejlépe přes detektor WHENEVER NOT FOUND
- Kurzor se uzavře příkazem CLOSE

3. varianta dynamického SQL III

- Stejně jako u předchozích variant musí být předem známý počet i datový typ vstupních i výstupních hostitelských proměnných
- Pokud jejich počty nebo datové typy předem známé nejsou, musí být použita nejsložitější 4. varianta dynamického embedded SQL

SQLDA

- SQL Descriptor Area
- Používá se ve 4. variantě dynamického embedded SQL
- Je strukturou pro komunikaci s databází, na rozdíl od SQLCA do ní databáze nejen zapisuje, ale také z ní čte hodnoty dosazené programem
- Položky jsou vyplňovány při PREPARE, DESCRIBE a FETCH příkazech

SQLDA II

- Každou instanci je potřeba naalokovat funkcí SQLSQLDAAlloc()
- Položky N (maximální počet placeholders nebo sloupců SELECT) a M (pole maximálních velikostí výstupních bufferů) vyplní SQLSQLDAAlloc()
- Položky L, T, F, S, C vyplní příkaz DESCRIBE
- Položky V a I vyplní fetch

SQLDA – DESCRIBE ITEMS

- L – pole délek vstupních proměnných nebo sloupců SELECT dotazu
- T – pole kódů datových typů
- F – skutečný počet placeholders
- S – pole ukazatelů na řetězce obsahující jména vstupních proměnných nebo sloupců
- C – pole délek řetězců obsahujících jména vstupních proměnných a SELECT sloupců

SQLDA – FETCH ITEMS

- V – pole ukazatelů na hodnoty hostitelských proměnných a výstupních sloupců
- I – pole ukazatelů na hodnoty indikátorů

Celý postup 4. varianty

- PREPARE stmt FROM query
- DECLARE c CURSOR FOR stmt
- DESCRIBE BIND VARIABLES FOR stmt INTO bd
- OPEN c USING bd
- DESCRIBE SELECT LIST FOR stmt INTO sd
- FETCH c USING DESCRIPTOR sd
- CLOSE c

Zdroje

- Pro* C/C++ Precompiler Programmer's Guide – 6 Embedded SQL
- Pro* C/C++ Precompiler Programmer's Guide – 13 Dynamic SQL
- Doplňkově i studijní materiály pro předmět Administrace Oracle