„Vnitřní“ programovací konstrukce (Embedded SQL) - procedurální prostředky v rámci jazyka SQL

Wednesday, May 29, 2013 4:49 PM

Embedded SQL

Přímý přístup programovacího jazyka do databázových struktur – **jazyk (překladač) je obohacený o konstrukce pro SQL**

Je metoda vkládání řádkových SQL příkazů do kódu programovacího jazyka, protože tento programovací jazyk neumí parsovat SQL. Vkládané SQL je parsováno v SQL preprocesoru.

- **Query Language**
  - http://people.ku.edu/~nkinners/LangList/Extras/classif.htm
- SQL dotazy jsou psány přímo ve zdrojovém kódu. Syntaxe SQL je přizpůsobena syntaxi daného programovacího jazyka.
- Před kompilací programu se musí zdrojový kód zpracovat preprocesorem Embedded SQL, kdy SQL dotazy jsou nahrazeny odpovídajícím kódem programovacího jazyka.
- Nejčastěji v kombinaci s jazyky C/C++.

Podpora v databázích

Podporují klasicky velké databázové systémy

- IBM DB2 (C/C++, Java, Cobol)
- Oracle (Cobol, *Pro*C* - embedded SQL Oraclu pro C/C++)
- PostgreSQL (C/C++)

Nepodporují

- MySQL
- Microsoft SQL Server (starší verze podporovali C)

Příklad syntaxe

Oracle Embedded SQL v jazyce C:

```
    {
        int a;
        /* ... */
        EXEC SQL SELECT salary INTO :a
                 FROM Employee
                 WHERE SSN=876543210;
        /* ... */
        printf("The salary is %d\n", a);
        /* ... */
    }
```

Embedded SQL

The first technique for sending SQL statements to the DBMS is embedded SQL. Because SQL does not use variables and control-of-flow statements, it is often used as a database sublanguage that can be added to a program written in a conventional programming language, such as C or COBOL. This is a central idea of embedded SQL: placing SQL statements in a program written in a host programming language. Briefly, the following techniques are used to embed SQL statements in a host language:

- Embedded SQL statements are processed by a special SQL precompiler. All SQL statements begin with an introducer and end with a terminator, both of which flag the SQL statement for the precompiler. The introducer and terminator vary with the host language. For example, the

introducer is "EXEC SQL" in C and "&SQL(" in MUMPS, and the terminator is a semicolon (;) in C and a right parenthesis in MUMPS.

- Variables from the application program, called host variables, can be used in embedded SQL statements wherever constants are allowed. These can be used on input to tailor an SQL statement to a particular situation and on output to receive the results of a query.
- Queries that return a single row of data are handled with a singleton SELECT statement; this statement specifies both the query and the host variables in which to return data.
- Queries that return multiple rows of data are handled with cursors. A cursor keeps track of the current row within a result set. The DECLARE CURSOR statement defines the query, the OPEN statement begins the query processing, the FETCH statement retrieves successive rows of data, and the CLOSE statement ends query processing.
- While a cursor is open, positioned update and positioned delete statements can be used to update or delete the row currently selected by the cursor.

From <http://msdn.microsoft.com/en-us/library/windows/desktop/ms713573(v=vs.85).aspx>

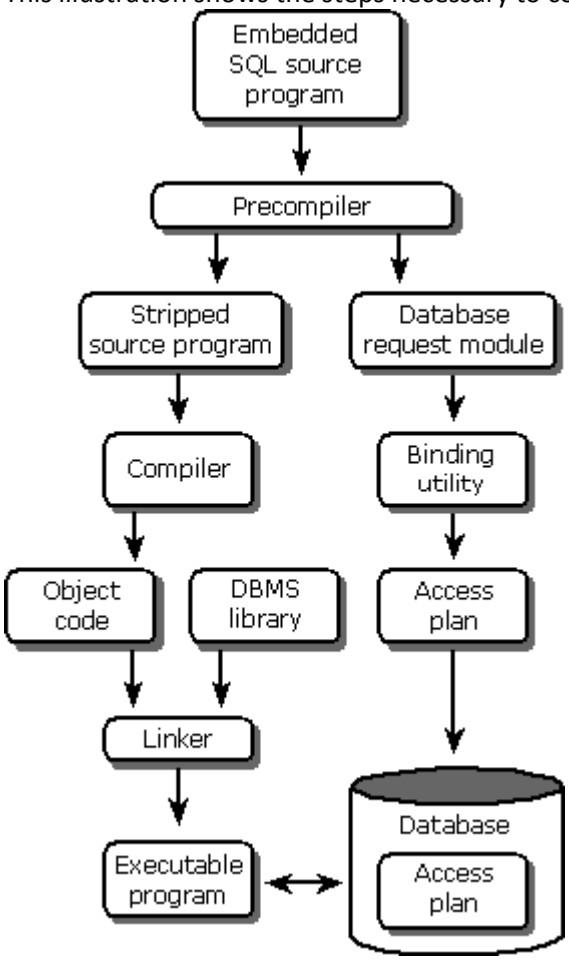
The following code is a simple embedded SQL program, written in C. The program illustrates many, but not all, of the embedded SQL techniques. The program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen.

From <http://msdn.microsoft.com/en-us/library/windows/desktop/ms714570(v=vs.85).aspx>

```
int main() {
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    int OrderID;       /* Employee ID (from user)       */
    int CustID;        /* Retrieved customer ID         */
    char SalesPerson[10]  /* Retrieved salesperson name  */
    char Status[6]       /* Retrieved order status       */
  EXEC SQL END DECLARE SECTION;
/* Set up error processing */
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
/* Prompt the user for order number */
  printf ("Enter order number: ");
  scanf_s("%d", &OrderID);
/* Execute the SQL query */
  EXEC SQL SELECT CustID, SalesPerson, Status
    FROM Orders
    WHERE OrderID = :OrderID
    INTO :CustID, :SalesPerson, :Status;
/* Display the results */
  printf ("Customer number:  %d\n", CustID);
  printf ("Salesperson: %s\n", SalesPerson);
  printf ("Status: %s\n", Status);
  exit();
query_error:
  printf ("SQL error: %ld\n", sqlca->sqlcode);
  exit();
bad_number:
  printf ("Invalid order number.\n");
  exit();
}
```

From <http://msdn.microsoft.com/en-us/library/windows/desktop/ms714570(v=vs.85).aspx>

Because an embedded SQL program contains a mix of SQL and host language statements, it cannot be submitted directly to a compiler for the host language. Instead, it is compiled through a multistep process. Although this process differs from product to product, the steps are roughly the same for all products.
This illustration shows the steps necessary to compile an embedded SQL program.



Five steps are involved in compiling an embedded SQL program:

1. The embedded SQL program is submitted to the SQL precompiler, a programming tool. The precompiler scans the program, finds the embedded SQL statements, and processes them. A different precompiler is required for each programming language supported by the DBMS. DBMS products typically offer precompilers for one or more languages, including C, Pascal, COBOL, Fortran, Ada, PL/I, and various assembly languages.
2. The precompiler produces two output files. The first file is the source file, stripped of its embedded SQL statements. In their place, the precompiler substitutes calls to proprietary DBMS routines that provide the run-time link between the program and the DBMS. Typically, the names and the calling sequences of these routines are known only to the precompiler and the DBMS; they are not a public interface to the DBMS. The second file is a copy of all the embedded SQL statements used in the program. This file is sometimes called a database request module, or DBRM.
3. The source file output from the precompiler is submitted to the standard compiler for the host programming language (such as a C or COBOL compiler). The compiler processes the source code and produces object code as its output. Note that this step has nothing to do with the DBMS or with SQL.
4. The linker accepts the object modules generated by the compiler, links them with various library routines, and produces an executable program. The library routines linked into the executable program include the proprietary DBMS routines described in step 2.
5. The database request module generated by the precompiler is submitted to a special binding utility. This utility examines the SQL statements, parses, validates, and optimizes them, and then produces an access plan for each statement. The result is a combined access plan for the

entire program, representing an executable version of the embedded SQL statements. The binding utility stores the plan in the database, usually assigning it the name of the application program that will use it. Whether this step takes place at compile time or run time depends on the DBMS.

Notice that the steps used to compile an embedded SQL program correlate very closely with the steps described earlier in Processing an SQL Statement. In particular, notice that the precompiler separates the SQL statements from the host language code, and the binding utility parses and validates the SQL statements and creates the access plans. In DBMSs where step 5 takes place at compile time, the first four steps of processing an SQL statement take place at compile time, while the last step (execution) takes place at run time. This has the effect of making query execution in such DBMSs very fast.

From <http://msdn.microsoft.com/en-us/library/windows/desktop/ms713968(v=vs.85).aspx>

Procedurální prostředky SQL (procedurální rozšíření SQL)

- SQL bylo původně navrženo pro získávání dat z relačních databází. SQL je deklarativní jazyk a ne imperativní, jako například C nebo Java.
- Dodavatelům DB systémů možnosti SQL nedostačovaly a tak si začali implementovat vlastní procedurální rozšíření SQL.
- příklad: kursory, bloky, proměnné

Motivace

- Šetření komunikačního kanálu
- Menší množství odesílaných povelů
- v jednom povelu je větší množství příkazů
- Podstatně menší objem přenesených dat - Data se zpracují na serveru bez přenosu na klienta
- Odlehčení klienta - Možnost ukládat a vykonávat kód na serveru

Na co si dát při návrhu pozor

- Hlídat na úrovni databáze všechny manipulace s daty, které ohlídat jdou
  - Cokoli jde zadat uživatelem špatně, bude zadáno špatně
- Integritní omezení, triggery
  - Později je čištění nekonzistentních dat namáhavé a opravy často nemožné
  - Lépe ohlídat vše

Procedurální rozšíření v SQL:1999

SQL:1999 standardizuje procedurální rozšíření. Rozšíření se jmenuje *SQL/PSM - SQL/Persisted Stored Modules*. **Nikdo to moc nedodržuje a všichni si dělají vlastní standardy/implementace**

- funkce a procedury - lze zapsat v SQL i v hostitelském programovacím jazyce
- řídící konstrukce - cykly, větvení, přiřazení

Podpora v databázích

SQL:1999 (SQL/PSM)

- IBM DB2
- MySQL
- PostgreSQL

Vlastní (proprietární) rozšíření

- Oracle (PL/SQL)
- PostgreSQL (PL/pgSQL)
- Microsoft (T-SQL)
- Sybase (T-SQL)