嵌入式 SQL 中动态 SQL 技术的研究*)

晏 庆[1] 阳国贵[1] 陈宏盛[2] 景 宁[2]

(国防科技大学计算机学院 长沙 410073)[1] (国防科技大学电子科学与工程学院 长沙 410073)[2]

摘 要 对动态 SQL 技术进行了研究,实现了一个嵌入式 SQL 系统中动态 SQL 语句处理的 3 种方法。首先介绍了整个嵌入式 SQL 系统 GKD-ESQL 系统,然后详细描述了 3 种动态 SQL 语句的处理方法,重点讲述了方法 3 的设计与实现,最后讲述了动态 SQL 语句的错误处理机制。

关键词 嵌入式 SQL,动态 SQL,错误处理,DBMS

The Research of Dynamic SQL Technology in an Embedded SQL System

YAN Qing[1] YANG Guo-Gui[1] CHEN Hong-Sheng[2] JING Ning[2]

(College of Computer, National University of Defense Technology, Changsha 410073)[1]

(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073)[2]

Abstract The technology of dynamic SQL is researched and three methods for using dynamic SQL statements in an embedded SQL system are implemented. The whole embedded SQL system GKD-ESQL is introduced, and then three methods for using dynamic SQL statements are proposed. The design and implementation of the Method 3 is mainly discussed. At last, the mechanism of the error handler of the dynamic SQL is described.

Keywords Embedded SQL system, Dynamic SQL, Error handler, DBMS

1 引言

SQL 语言与过程式语言相结合,已成为数据库发展的一个趋势。大多数主要的 DBMS 支持嵌入式 SQL。嵌入式 SQL[1,2]就是将 SQL 语句嵌入到高级语言中,如 C,FORTRAN,COBOL 等。嵌入式 SQL 语言综合了 SQL 语言的数据处理能力和高级语言的过程处理能力,是一种高性能、可移植的事务处理语言,正受到越来越广泛的重视和应用。

用户在编写嵌入式 SQL 的应用程序时,可以在程序中插入完整的 SQL 语句。但是在很多应用中,在编译时,SQL 语句还不能完整地知道,而要在程序运行时才可以构造出来。这就产生了动态 SQL 技术。动态 SQL 技术是一种先进的程序设计技术,它是相对于静态而言。静态 SQL 语句在编码时就能把它们完整地写出来。但是在某些情况下,在编码时 SQL 语句还不能完整地写出来,而是在程序执行时才能构造出来,这种在程序执行过程中临时生成的 SQL 语句叫动态 SQL 语句。利用动态 SQL 语句来编写程序的方法叫动态 SQL 技术。

使用动态 SQL 语句有很多优点,比如包含动态 SQL 语句的程序比一般的嵌入式 SQL 程序更加通用、功能更强、更加精炼、编码更灵活,用户可在运行时通过交互地输入来构造 SQL 语句。

当前流行的各种数据库管理系统都支持动态 SQL 技术,如 Oracle 的 Pro * C/C++[3],PostgreSQL 的 ECPG[4]等。本文描述的动态 SQL 技术可以在 C 语句中嵌入动态 SQL 语句,在 GKD-Base 中的 GKD-ESQL 系统中正确运行。

GKD-Base 是我国自主研究开发的军用安全数据库管理系统,是兼容 Oracle、具有自主知识产权的关系型数据库管理系统。GKD-Base 的应用非常广泛,不但可以满足用户的一般需求,并且具有高度的安全性和保密性。GKD-ESQL 系统是关系数据库管理系统 GKD-Base 的重要组成部分,它支持将 SQL 语句嵌入到 C 语言中,这样可以把 SQL 对数据库的操作和 C 语言的过程控制的优点结合在一起,方便用户对数据库的操作。动态 SQL 语句的处理方法是嵌入式 SQL 的一个重要部分,本文将重点介绍 GKD-ESQL 系统中动态 SQL 语句的处理方法。

2 GKD-ESQL 的系统结构

GKD-ESQL 系统可以分为两部分实现:一部分是预编译器的实现;另一部分是 ESQL 运行库的实现。预编译器的功能是将 GKD-ESQL 语言中的 GKD-ESQL 语句和 C 语句识别出来,对 GKD-ESQL 语句进行词法分析和语法分析,并将 GKD-ESQL 语句转换成 C 语言的编译器能够识别的函数调用。它包括词法分析模块和语法分析模块。ESQL 运行库的功能是实现函数调用,并与服务器交互。

预编译器的工作是读入用 GKD-ESQL 语言编写的源程序,识别其中的 GKD-ESQL 语句和 C 语句,对 C 语句不做任何处理,而对 GKD-ESQL 语句要进行语法检查和语义检查,然后将其转换成具有相应功能的 C 语言的函数调用。

ESQL 运行库的工作是实现应用程序与数据库服务器的交互,也就是 ESQL 运行库实现了由预编译器生成的与 SQL 语句相对应的 C 语言的函数。ESQL 运行库中最重要的一个

*)本课题得到国家"八六三"高技术研究发展计划基金项目(2003AA5110)资助。晏 庆 硕士研究生,主要研究方向为数据库系统技术;阳国贵 副教授,主要研究方向为数据库系统技术;陈宏盛 副教授,主要研究方向为 GIS 与空间数据库技术;景 宁 教授,主要研究方向为 GIS 与空间数据库技术。

函数就是 ESQLdo(),预编译器将大部分的 SQL 语句都转换成这个函数,它实现了大部分的数据操作功能。还有一些与 GKD-ESQL 系统相关的函数,如初始化 SQL 通信区的函数、构造纯 SQL 语句的函数等、与服务器交互的函数等。

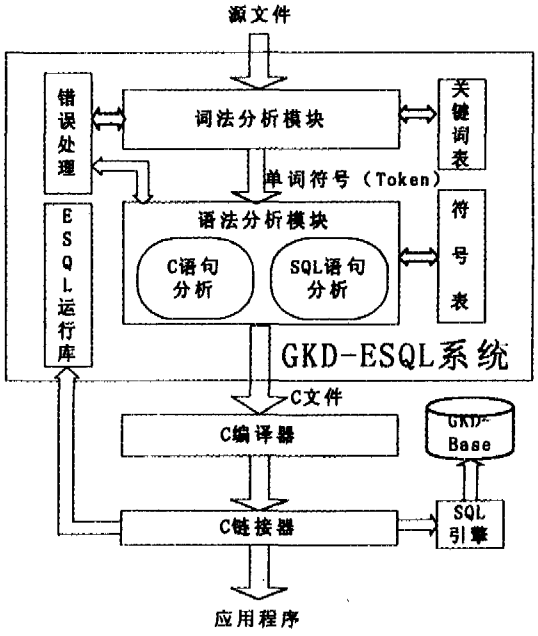

图 1　嵌入式 SQL 语言的执行流程图

图 1 是整个嵌入式 SQL 语言的执行流程图。源文件经过词法分析模块和语法分析模块之后,转换成 C 语言编译器能够识别的 C 文件,经过 C 语言编译器编译后,变成目标代码。然后链接器再调用 ESQL 运行库,并且要通过 SQL 引擎对 GKD-Base 数据库访问,最后变成可执行程序。

3　动态 SQL 语句的处理方法

GKD-Base 中嵌入式 SQL 对动态 SQL 语句的处理步骤是:

(1)构造 SQL 语句文本。动态 SQL 语句文本由字符串常量、变量和宿主变量占位符组成,但其中不包含 EXEC SQL 关键字和语句结束符(即分号";"),以及其他嵌入式 SQL 命令,如 OPEN、CLOSE、EXECUTE、FETCH、INCLUDE 等。

例如,下面两个字符串常量 stmt1 和 stmt2 所记录的内容为两个动态 SQL 语句,其中第一个语句中不包含宿主变量,而第二个动态 SQL 语句中则有三个宿主变量:

char stmt1 [128] = "INSERT INTO dept (deptno, dname,loc) VALUES(90,'demo','loc1')";

char stmt2 [128] = "INSERT INTO dept (deptno, dname,loc) VALUES(:no,:name,:loc)";

(2)分析解释 SQL 语句文本。GKD-Base 中嵌入式 SQL 对动态 SQL 语句文本进行语法检查,检查在动态 SQL 语句中引用的宿主变量是否已经在声明段(以 EXEC SQL BEGIN DECLARE SECTION 开始,以 EXEC SQL END DECLARE SECTION 结束)中声明等。

(3)为 SQL 语句结合宿主变量。在执行语句过程中,GKD-Base 读取宿主变量地址,并从这些变量中读取数据,或将数据写入这些变量中。

(4)执行 SQL 语句,实现对数据库的操作。

GKD-Base 中嵌入式 SQL 支持 3 种动态 SQL 语句的处理方法。下面分别讲述这 3 种方法的设计与实现。

3.1　方法 1

这种方法只能执行非查询 SQL 语句,并且在非查询动态 SQL 语句内不能包含输入宿主变量占位符。方法 1 使用 EXECUTE IMMEDIATE 命令立即执行动态 SQL 语句,所以在每次执行时,都需要对语句进行重新解释。方法 1 是动态 SQL 语句处理方法中最简单的一种方法,所以它的实现也比较简单。它的语法格式如下:

exec sql execute immediate :sql_statement;

sql_statement 是以宿主变量或字符串形式说明所执行的动态 SQL 语句。它的使用方法是以 exec sql execute immediate 命令开始立即执行 SQL 语句的。

如果 SQL 语句中含有宿主变量,那么这些宿主变量首先要在声明段以 EXEC SQL BEGIN DECLARE SECTION 开始,以 EXE SQL END DECLARE SECTION 中声明。预编译器首先处理在声明段中的宿主变量,将其有关信息加入到符号表中,这在以后的执行中需要读取的。预编译器将大部分的动态 SQL 语句都转换成一个 ESQLdo()函数,它实现了大部分的数据操作功能。经过预编译器后,源文件可以转换成被 C 语言的编译器识别的 C 语言源文件。转换后生成的 C 语言源文件经过 C 语言编译器编译后,生成目标文件,然后再由 C 语言的链接器调用 ESQL 运行库中的 ESQLdo()函数,执行对数据库的操作。

3.2　方法 2

方法 2 也只能执行非查询语句,并且允许非查询动态 SQL 语句内包含输入宿主变量占位符,但它要求在预编译时刻必须确定动态 SQL 语句内的占位符数量及输入宿主变量的数据类型。方法 2 按照以下两个步骤执行动态 SQL 语句:

(1)准备阶段。调用 PREPARE 语句,准备需要执行的动态 SQL 语句,这时 GKD-Base 对动态 SQL 语句进行语法分析、检查它所参照引用的数据库对象是否存在等;

(2)执行阶段。调用 EXECUTE 命令,执行准备好的动态 SQL 语句。

它的语法格式如下:

exec sql prepare statement_name from :sql_statement;

exec sql execute statement_name using :host_variable1,:host_variable2……

其中,statement_name 为应用程序定义的语句名称标识;sql_statement 为宿主变量或字符串常量形式,说明所执行的动态 SQL 语句;host_variable1、host_variable2……为宿主变量。当动态 SQL 语句中包含宿主变量占位符时,在执行时应用程序要使用 using 子句为每个占位符提供宿主变量。

预编译器在执行阶段通过 statement_name 标识确定所执行的动态 SQL 语句。所以,应用程序为不同动态 SQL 语句所赋名称标识不能相同,否则将导致程序运行时刻错误。

经过预编译器后,源文件可以转换成被 C 语言的编译器识别的 C 语言源文件,exec sql prepare 语句转换成 ESQLprepare()函数。exec sql execute 语句转换成 ESQLdo()函数,将 ESQLprepare()函数准备好的 sql 语句作为一个参数,传递给 ESQLdo()函数。

3.3　方法 3

方法 3 采用游标方式处理动态 SQL 语句,所以它可以执行查询语句。方法 3 要求在预编译阶段必须确定动态 SQL 语句中的选择列表项数以及输入宿主变量数量。动态 SQL

语句中的数据库对象名称和列名可以在运行时刻指定，但此时不能以宿主变量形式指定对象名称或列名。

方法3对动态SQL语句的处理步骤也分为准备和执行阶段。准备阶段由PREPARE命令完成，其操作与方法2相同。而在执行阶段，方法3使用游标方式，它包括游标声明、打开、提取和关闭几个步骤，各步所使用的语句分别为DECLARE、OPEN、FETCH和CLOSE。因为方法3使用游标方式，所以实现这种方法比较复杂。

它的语法格式如下：

exec sql declare cursor_name cursor for select_statement;

exec sql open cursor_name;

exec sql fetch cursor_name into :host_variable1,:host_variable2,:host_variable3,……;

exec sql close cursor_name;

其中，cursor_name为PREPARE命令准备好的SQL语句，host_variable1、host_variable2、host_variable3……为宿主变量。

方法3的设计与实现的难点主要是在游标这一部分，因为GKD-Base的SQL引擎不支持游标。但是它提供了一套API函数访问接口，称之为GKD-Base API函数（以下简称XAPI函数）。在游标的设计与实现中，我们通过调用XAPI函数对SQL语句进行处理。

XAPI接口函数根据功能主要分为连接数据库（apilon）、准备SQL命令请求（apiopn、apiosq、apibrn、apibnn）、提交执行SQL命令请求（apiexe、apihmi）、获取有关结果集的信息和结果集中的记录数据（apidsc、apidfn、apifch、apifcn、apilgt）、数据库事务控制（apicom、apirol、apicon、apicof）以及其它XAPI函数（apigem、apicco、apicls、apilof）。所有的XAPI接口函数都有一个（STRUCT HOST *）类型的结构指针变量，需要事先声明一个STRUCT HOST类型的变量；此外，使用SQL命令操作的XAPI接口函数还需要一个unsigned int类型数据库游标变量。在函数返回值方面，除apigem函数外，其它API接口函数的返回值含义是一致的。如果函数返回0，则说明函数执行成功；如果函数返回非0值，则说明函数执行失败，返回值为特定于GKD-Base数据库的错误代码，这时可调用apigem函数获取错误描述信息，错误代码也将以文本形式包含在错误描述信息中。

方法3中游标的使用分为包括声明游标、为查询打开游标、取得结果放入宿主变量中、关闭游标4个步骤。我们把游标的实现分为两部分来完成：首先是游标的编译，然后才是游标的解释执行。

游标声明操作定义游标的名字，并把游标名与其中的SELECT语句关联。编译时，首先调用SQL引擎，对SQL语句部分进行语法分析，检查是否有语法错误；如果检查通过，则转换为SQL引擎可以识别的格式，把分析产生的中间代码保存在游标结构中的一个域中；最后把游标的索引信息加入符号表中。对游标的打开语句进行编译时，首先在符号表中进行查找，获取游标在声明阶段保存在符号表中的详细游标信息；最后把这些信息作为一个分支添加到语法树的当前节点上。对检索赋值语句编译同样需要在符号表中获取游标的详细信息，并提取出SELECT语句中将被赋值的变量保存在一个链表中；然后对游标结构中SELECT语句查询列的信息和链表中宿主变量的信息进行比较，检查游标中变量的数目与游标引用列数目是否一致；最后把游标的信息和赋值变量的信息作为一个分支添加到语法树的当前节点上。游标的检索操作完成以后，关闭游标。游标的关闭操作比较简单，主要是释放资源并把游标状态初始化。

游标在解释执行器中的动作，依据它的操作步骤也是分为打开、检索赋值和关闭操作来完成的。对游标执行打开和检索赋值操作时，要动态地记录游标的当前状态信息，并根据这些信息进行相应的操作。游标OPEN语句的解释执行如图2。

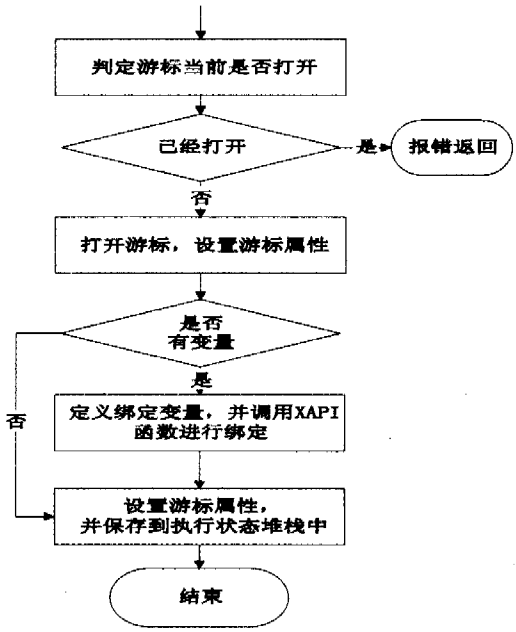

图2 游标OPEN操作

一个游标在关闭之前是不允许重复打开的。在遇到OPEN语句执行打开游标的操作时：

1）首先要判定游标的当前状态。先检查这个游标的%ISOPEN属性，判断游标是否已经打开；如果游标已经打开，则向用户报错；如果没有打开，把%ISOPEN赋值为打开状态；

2）确定了游标状态后，从语法树中获取游标中的SELECT语句，调用SQL引擎，对其进行解释执行的准备工作：先用XAPI函数对SELECT语句进行分析；如果SELECT语句中有变量绑定，则根据变量的类型、精度、规模等信息调用XAPI函数，把变量与对应的列绑定；

3）在SQL语句准备完之后，到目前为止还没有执行SQL语句生成结果集，无法确定是否有返回结果以及能够返回到多少结果。因此在这里要把游标的属性%FOUND、%NOTFOUND置为未知（NULL），%ROWCOUNT置为0，连同%ISOPEN属性、游标号等一同保存在执行状态中，完成对游标的打开操作。

游标打开之后，用FETCH语句把查询的结果赋予变量，如图3。

1）首先调用XAPI函数，得到游标中SELECT语句的查询列的个数；

2）然后调用XAPI函数，描述查询结果集中每一列的列定义信息。根据这些信息调用XAPI函数，为每一个SELECT列表中的列定义输出缓存区，用于存储列的描述信息；

3）处理完毕后，调用XAPI，执行这个SELECT语句，并调用XAPI函数，从查询中返回单个数据行，并把结果赋于赋

值变量；

4）如果有返回结果，则把%FOUND置为1，%NOTFOUND置为0，每一个查询的列都暂时存放在由先前执行XAPI调用时所标识的缓冲区中，同时把游标指针指向下一行数据并再次调用XAPI函数，行计数加1；

5）把所有的结果集都检索结束后，把%FOUND置为0，%NOTFOUND置为1；最后把结果保存在执行状态中，完成对FETCH语句的解释执行。

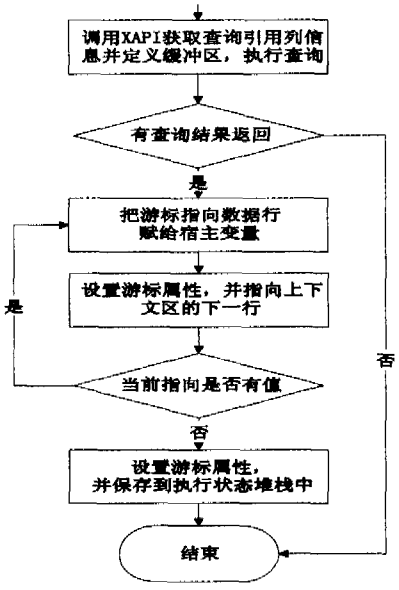

图3 游标FETCH操作

完成检索赋值操作后，执行游标的关闭操作。游标的关闭操作首先调用XAPI函数，关闭SELECT语句打开的游标，然后把当前的显式游标状态初始化，%ISOPEN置为关闭，%FOUND、%NOTFOUND、%ROWCOUNT置为无效。整个游标的解释执行到此结束。

4 错误处理机制

GKD-ESQL系统具备了一个精巧的错误处理机制，可以对动态SQL语句进行过程性的控制。根据执行SQL语句返回的SQLCODE和SQLSTATE，应用程序可以按照用户需求设置不同的控制流。该机制需要预处理器和ESQL运行库协同工作实现。

预处理器识别一个WHENEVER语句时，记录错误条件和相应的错误处理动作信息。在输出语句时，对那些根据执行结果的错误代码进行处理的语句，设置特殊的输出模式。于是，在输出ESQLdo()函数到目标文件时，预处理器会在该语句后附加判断错误码的语句。若符合相关错误条件，就执行相应的错误处理动作。WHENEVER语句的作用域从该语句出现之处到下一次WHENEVER语句用相同的错误条件设置不同的错误处理动作之处。在其作用域内，所有相关SQL语句后都会附加错误码判断和错误处理语句。

WHENEVER语句的语法格式为：

EXEC SQL WHENEVER〈condition〉〈action〉；

其中，condition指出GKD-Base检测到的错误、警告等条件，它有以下几种取值：SQLWARNING、NOT FOUND和SQLERROR。

action参数说明当GKD-Base检测到指定的错误类型后，应用程序所执行的错误处理操作，其值可为：

CONTINUE：应用程序继续执行下一条语句，这也是默认情况下GKD-ESQL应用程序所执行的操作；

DO：应用程序将控制权转移到一个错误处理函数，当错误处理函数执行完成后，又将控制权转回到发生错误的SQL语句之后的下一条语句；

DO BREAK：处理循环结构所执行的SQL语句产生的运行错误，WHENEVER语句在检测到指定的错误后，执行C语言语句的break语句，跳出当前循环；

DO CONTINUE：处理循环结构所执行的SQL语句产生的运行错误，WHENEVER语句在检测到指定的错误后，执行C语言语句的continue语句，立即执行下一次循环操作；

GOTO lable_name：应用程序跳转到lable_name指定标号处执行；

STOP：这时应用程序立即停止运行，并回滚未提交的事务。预编译器在处理STOP操作时，它实际上是调用C语言的exit()函数来结束程序的运行。

在ESQL运行库中定义了一个SQL通信区（SQLCA），用来存放动态SQL语句执行的结果信息，如错误码、错误信息、处理元组标识、处理元组数等。在语句执行出现异常时，根据SQL语句执行结果设置SQLCA的相应域，并记录SQLCODE和SQLSTATE。SQLCA的具体定义如下：

```
struct sqlca
{
    char sqlcaid[8];//SQLCA标识字符串，它被初始化为"SQLCA"
    long sqlabc;//SQLCA结构的长度，其值为sizeof(struct sqlca)
    long sqlcode;//存储SQL操作状态代码
    struct
    {
        int sqlerrml;//存储错误消息长度
        char sqlerrmc[70];//错误存储消息文本
    }sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlwarn[8];
    char sqlext[8];
}sqlca;
```

小结 本文对嵌入式SQL中动态SQL技术进行了研究，实现了动态SQL语句的几种处理方法。动态SQL语句可以很好地把SQL语句对数据操作的优点和C语句的过程控制优点结合起来。动态SQL语句的实现可以有助于GKD-Base的推广使用。

参考文献

1 萨师煊，王珊. 数据库系统概论（第三版）[M]. 北京：高等教育出版社，2000

2 Date C J. An Introduction to Database Systems (7th Edition) [M]. USA: Addison Wesley, 2000

3 Oracle Corporation. Pro * C/C++ Procomplier Programmer's Guide [M]. USA: Oracle Corporation, 2002

4 The PostgreSQL Global Development Group. PostgreSQL8. 0 Developer's Guide [M]. USA: postgresql. org, 2005

5 Aho A V, Sethi R, Ullman J D. Compilers: Principles, Techniques, and Tools [M]. USA: Addsion-Wesley Publishing Company, 2001