EMBEDED SQL

I just wandered whether there is a method of combining the computing power of a high-level language like C/C++ and the database manipulation capabilities of SQL a few days ago. But now I find the answer, embedded SQL. We can use embedded SQL to execute any SQL statement from an application.

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, C++ or Fortran. This is a central idea of embedded SQL: placing SQL statements in a program written in a host programming language.

Two types of SQL can be used in an application program -- static SQL and dynamic SQL-- and each have their own advantages and disadvantages.

Static SQL

Static SQL is called static because the SQL statements in the program do not change each time the program is run.

Static SQL statements are hardcoded in an application program when the source code is written. The source code is then processed using a SQL precompiler before it can be compiled and executed.

The SQL precompiler evaluates the declared data types of all host defined variables and determines which data conversion methods need to be used when data is moved to and from the database. Additionally, the SQL precompiler performs error checking on each coded SQL statement and ensures that appropriate host variable data types are used for their respective table column values.

Static SQL works well in many situations. In fact, it can be used in any application for which the data access can be determined at program design time. For example, an order entry program always uses the same statement to insert a new order, and an airline reservation system always uses the same statement to change the status of a seat from available to reserved. Each of these statements would be generalized through the use of host variables. Different values can be inserted in a sales order and different seats can be reserved. Because such statements can be hard-coded in the program, such programs have the advantage that statements need to be parsed, validated, and optimized only once, at compile time. This results in relatively fast code.

We can easily use static SQL as follows:

```
Bool print_employee_info()
{
    int age=0;   char name[20];   char address[80];
```

```
//Build the relationship between C and SQL
#SQL BIND (AGE, age);
#SQL BIND (NAME, name);
#SQL BIND (Address, address);
// Running SQL
#SQL SELECT AGE, NAME, ADDRESS FROM EMPLOYEES;
//Running C
if (age==NULL)
{
        return false;
}
while (age!=NULL)
{
        printf(“ AGE=%d\n”, age);
        printf(“ NAME=%d\n”, name);
printf(“ Address=%d\n”, address);
}
return true;
}
```

From the example we can know that although Static SQL statements are easy to use, they are limited because their format must be known in advance by the precompiler and because they can only work with host variables. There are many situations in which the content of an SQL statement is not known by the programmer in advance. For example, suppose a spreadsheet allows a user to enter a query, which the spreadsheet then sends to the DBMS to retrieve data. The contents of this query obviously cannot be known to the programmer when the spreadsheet program is written.

Also, if you made some mistakes in your query process, the compiler will help to check and point them out. However, static SQL cannot be changed during the running process of the program, thus a lot of redundancy codes have to be added. What is worse, this means it is impossible to transplant these codes into other programs, which may decrease the charm of static SQL.

Dynamic SQL

To solve this problem, the spreadsheet uses a form of embedded SQL called Dynamic SQL. Unlike static SQL statements, which are hard-coded in the program, Dynamic SQL statements can be built at run time and placed in a string host variable. They are then sent to the DBMS for processing. Because the DBMS must generate an access plan at run time for dynamic SQL statements, dynamic SQL is generally slower than static SQL. When a program containing dynamic

SQL statements is compiled, the dynamic SQL statements are not stripped from the program, as in static SQL. Instead, they are replaced by a function call that passes the statement to the DBMS.

Since the actual creation of Dynamic SQL statements is based upon the flow of programming logic at application run time, they are more powerful than Static SQL statements. However, because the DBMS must go through each of the five steps of processing a SQL statement for each dynamic request, Dynamic SQL tends to be slower than Static SQL.

Source: http://toyhouse.cc/profiles/blogs/embeded-sql