Extending InterBase with User Defined Functions

by Paul McGee

Borland Developer's Conference 1995

► An overview of UDFs User Defined Functions

User Defined Functions (UDF) are host-language functions for performing customized, often-used tasks in applications. UDFs allow the programmer to modularize a database application and to enhance the functionality that can be embedded in the database itself. UDFs also are always executed on the database server. This can reduce the network traffic by allowing the data to be massaged on the server instead of retrieving it all to the client and massaging it and then passing it back to the server to update the database.

UDFs can perform routine tasks such as getting the available disk space on the server, trimming blanks from a string, computing the standard deviation for a series of values, etc. UDFs can perform any function that can be expressed in the host- language. That language is usually either C or C++.

UDFs are most effective in: providing functionality not available in the SQL language for defining database objects or querying or updating the database; or providing common functions for heterogenous client workstation environments.

As with everything, there is a cost to using UDFs. There are two primary area's of overhead associated with UDFs. The first is that UDFs are blocking AST routines on UNIX or VMS platforms. That means that while the UDF is executing, no other access to the database can occur. The caveat then should be to keep UDFs as small and efficient as possible. The second is that if the database server crashes and you need to move the database to another machine, you must first install the UDF libraries on the new server machine before you can even restore the database from a backup. This is not hard if the new server has the same operating system. But moving to another operating system can require you to at least recompile the libraries if not modify the source code.

► Sample UDFs

InterBase includes a couple of built-in SQL functions: *UPPER*, *GEN_ID*, and *CAST*. *UPPER* converts a string to all uppercase. *GEN_ID* generates a unique long integer value for a specific GENERATOR that has already been defined in the database. This is very useful for generating primary keys such as customer number or employee numbers. *CAST* will convert a column from one datatype to another.

InterBase also supplies the source to other sample UDFs in the examples directory. They are contained in the udflib.c file. The UDFs there are: *lower*, *strcat, substr, trim, trunc*, *doy, moy, dow, sysdate, add2, mul, fact, abs, maxnum, sqrt, blob_linecount, blob_bytecount,* *substr_blob*. *lower* converts a string to lowercase. *strcat* concatenates two strings together. *substr* will return the portion of a string. *trim* will trim off any *leading* blanks from a string. *trunc* will return a truncated string. *doy* will return the current julian day of the year. *moy* will return the current month number, 1 through 12. *dow* will return the current day's name, i.e. Tuesday. *sysdate* will return the current date in a string with the format of "MMM-DD-YYYY". *add2* will add two integers together. *mul* will multiply two doubles. *fact* will return the factorial of an double. *abs* will return the absolute value of a double. *maxnum* will return the max of two doubles. *sqrt* will return the square root of a double. *blob_linecount* will return the number of lines in a blob. *blob_bytecount* will return the total size in bytes of a blob. *substr_blob* will a portion of a text blob.

We are going to add a couple of new ones, *rtrim, left, right, swapcase, imonth, iday, iyear*. *rtrim* will trim off any trailing blanks on a string. *left* will return a string of the first n bytes of an input string. *right* will return a string of the last n bytes of a string. *swapcase* will convert swap upper to lower and lower to upper in a string. *imonth* will return the month number, 1 through 12, from an InterBase date field. *iday* will return the day number, 1 through 31, from an InterBase date field. *iyear* will return the full year number, i.e. 1995, from an InterBase date field.

Setting up UDFs

Once you have written the UDF in a host-language you must create the host-language library and make it available. Then you must define it to the database. We'll cover the data definition first and then cover making the libraries.

Defining UDFs in the data definition language is very simple. The basic syntax is:

```
DEFINE EXTERNAL FUNCTION name [<DATATYPE> | CSTRING (int)
             [, <DATATYPE> | CSTRING (int) ...]]
             RETURNS {<DATATYPE> [BY VALUE] | CSTRING (int)}
             ENTRY_POINT "<ENTRYNAME>"
             MODULE_NAME "<MODULENAME>" ;
```

The name is what you what to refer to the function as when using it. It can be up to 31 characters long. The first datatype field is for input parameters to the function. Datatype refers to a standard InterBase datatype of INTEGER, CHAR, VARCHAR, etc. Or you can use the optional CSTRING which is a normal C style NULL terminated array of characters. The entry_point refers to the actual function name within the source code. In the InterBase supplied examples the SQL function name is *lower* while the actual function name within udflib.c is *fn_lower_c*. The module_name refers to the library name the function is compiled into and exported from. For example, here are the SQL definitions of the functions *lower* and *substr*.

```
DEFINE EXTERNAL FUNCTION lower
             VARCHAR (256)
             RETURNS CSTRING (80)
             ENTRY_POINT "fn_lower_c" MODULE_NAME "funclib";

     DEFINE EXTERNAL FUNCTION substr
             CSTRING (256), SMALLINT, SMALLINT
             RETURNS CSTRING (80)
             ENTRY_POINT "fn_substr" MODULE_NAME "funclib";
```

Now to make this into a function library on NT using Borland C++ we should have local copies of LIB modules for any DLLs we may want to compile in with the UDFs.

```
implib mygds32.lib \interbas\bin\gds32.dll
```

Then link with the necessary options to create our new funclib.dll. This is documented in the V4 Installing and Running on NT manual on page 18 through 22.

```
bcc32 -v -a4 -DWIN32 -tWM -tWCD -efunclib.dll udf.c mygds32.lib
```

To use the DLL locally it must be in the BIN directory of the InterBase tree or in a directory in your PATH environment variable. To use it remotely, it must be in the directory on the server taht is in the PATH environment variable of the *user* running the InterBase Remote Service. By default, that is the SYSTEM account.

Using UDFs

Once compiled, linked, and defined; a UDF can be used in any SQL statement where UDFs are permitted.

They can be used in: defining *computed by* fields as part of a table definition; as column expressions in view defintions or with stored procedures and/or triggers as part of a SELECT, INSERT, UPDATE, or DELETE operation.

For example, as a computed by field in a table:

```
   CREATE TABLE name ( FIRST_NAME VARCHAR(20), LAST_NAME VARCHAR(20),
                FULL_NAME_UPPER COMPUTED BY
                (upper(FIRST_NAME) | " " | upper(LAST_NAME)));
```

As a column expression in a view:

```
   CREATE VIEW upper_names (FIRST_NAME, LAST_NAME) AS
                SELECT upper(n.first_name), upper(n.last_name) FROM name
n;
```

In a SELECT operation:

```
   SELECT substr(n.FIRST_NAME, 2, 4) FROM name n WHERE
                upper(n.LAST_NAME) = 'MOORE';
```

In an INSERT operation:

```
   INSERT INTO name (FIRST_NAME, LAST_NAME)
                VALUES (rtrim(:new_fname), rtrim(:new_lname));
```

In an UPDATE operation:

```
   UPDATE name SET LAST_NAME = rtrim(:new_lname) WHERE
                upper(n.LAST_NAME) = 'JONES';
```

In a DELETE operation:

```
   DELETE FROM name WHERE left(LAST_NAME, 3) = 'SMI';
```