External functions (UDFs)

Ability to signal SQL NULL via a null pointer

C. Valderrama

Previous to Firebird 2, UDF authors only could guess that their UDFs might return a null, but they had no way to ascertain it. This led to several problems with UDFs. It would often be assumed that a null string would be passed as an empty string, a null numeric would be equivalent to zero and a null date would mean the base date used by the engine.

For a numeric value, the author could not always assume null if the UDF was compiled for an environment where it was known that null was not normally recognized.

Several UDFs, including the ib_udf library distributed with Firebird, assumed that an empty string was more likely to signal a null parameter than a string of length zero. The trick may work with CHAR type, since the minimum declared CHAR length is one and would contain a blank character normally: hence, binary zero in the first position would have the effect of signalling NULL.

However, but it is not applicable to VARCHAR or CSTRING, where a length of zero is valid.

The other solution was to rely on raw descriptors, but this imposes a lot more things to check than they would want to tackle. The biggest problem is that the engine won't obey the declared type for a parameter; it will simply send whatever data it has for that parameter, so the UDF is left to decide whether to reject the result or to try to convert the parameter to the expected data type.

Since UDFs have no formal mechanism to signal errors, the returned value would have to be used as an indicator.

The basic problem was to keep the simplicity of the typical declarations (no descriptors) while at the same time being able to signal null.

The engine normally passed UDF parameters by reference. In practical terms, that means passing a pointer to the data to tell the UDF that we have SQL NULL. However, we could not impose the risk of crashing an unknown number of different, existing public and private UDFs that do not expect NULL. The syntax had to be enhanced to enable NULL handling to be requested explicitly.

The solution, therefore, is to restrict a request for SQL NULL signaling to UDFs that are known to be capable of dealing with the new scenario. To avoid adding more keywords, the NULL keyword is appended to the UDF parameter type and no other change is required.

Example

```
declare external function sample
int null
returns int by value...;
```

If you are already using functions from ib_udf and want to take advantage of null signaling (and null recognition) in some functions, you should connect to your desired database, run the script ../misc/upgrade/ib_udf_upgrade.sql that is in the Firebird directory, and commit afterwards.

Caution: It is recommended to do this when no other users are connected to the database.

The code in the listed functions in that script has been modified to recognize null only when NULL is signaled by the engine. Therefore, starting with FB v2, rtrim(), ltrim() and several other string functions no longer assume that an empty string means a NULL string.

The functions won't crash if you don't upgrade: they will simply be unable to detect NULL.

If you have never used ib_udf in your database and want to do so, you should connect to the database, run the script ../udf/ib_udf2.sql, preferably when no other users are connected, and commit afterwards.

Note:

- Note the “2” at the end of the name.
- The original script for FB v1.5 is still available in the same directory.

back to top of page

UDF library diagnostic messages improved

A. Peshkov

Diagnostics regarding a missing/unusable UDF module have previously made it hard to tell whether a module was missing or access to it was being denied due to the UDFAccess setting in firebird.conf. Now we have separate, understandable messages for each case.

UDFs added and changed

UDFs added or enhanced in Firebird 2.0's supplied libraries are:

IB_UDF_rand() vs IB_UDF_srand()

F. Schlottmann-Goedde

In previous versions, the external function rand() sets the random number generator's starting point based on the current time and then generates the pseudo-random value.

```
srand((unsigned) time(NULL));
return ((float) rand() / (float) RAND_MAX);
```

The problem with this algorithm is that it will return the same value for two calls done within a second.

To work around this issue, rand() was changed in Firebird 2.0 so that the starting point is not set

explicitly. This ensures that different values will always be returned.

In order to keep the legacy behaviour available in case somebody needs it, srand() has been introduced. It does exactly the same as the old rand() did.

IB_UDF_lower

The function IB_UDF_lower() in the ib_udf library might conflict with the new internal function lower(), if you try to declare it in a database using the ib_udf.sql script from a previous Firebird version.

```
/* ib_udf.sql declaration that now causes conflict */
DECLARE EXTERNAL FUNCTION lower
   CSTRING(255)
   RETURNS CSTRING(255) FREE_IT
   ENTRY_POINT 'IB_UDF_lower' MODULE_NAME 'ib_udf';
```

The problem will be resolved in the latest version of the new ib_udf2.sql script, where the old UDF is declared using a quoted identifier.

```
/* New declaration in ib_udf2.sql */
DECLARE EXTERNAL FUNCTION "LOWER"
   CSTRING(255) NULL
   RETURNS CSTRING(255) FREE_IT
   ENTRY_POINT 'IB_UDF_lower' MODULE_NAME 'ib_udf';
```

Tip: It is preferable to use the internal function LOWER() than to call the UDF.

General UDF changes

Build changes

C. Valderrama Contributors

The FBUDF library no longer depends on FBCLIENT to be built.