Firebird SQL

Every database management system has its own idiosyncrasies in the ways it implements SQL. Firebird adheres to the SQL standard more rigorously than any other RDBMS except possibly its 'cousin', InterBase®. Developers migrating from products that are less standards-compliant often wrongly suppose that Firebird is quirky, which is really not true at all.

The following excerpts have been taken from the Firebird 2 Quick Start Guide, ©IBPhoenix Publications 2008.

Division of an integer by an integer

Firebird accords with the SQL standard by truncating the result (quotient) of an integer/integer calculation to the next lower integer. This can have bizarre results unless you are aware of it. For example, this calculation is correct in SQL:

```
1 / 3 = 0
```

If you are upgrading from an RDBMS which resolves integer/integer division to a float quotient, you will need to alter any affected expressions to use a float or scaled numeric type for either dividend, divisor, or both. For example, the calculation above could be modified thus in order to produce a non-zero result:

```
1.000 / 3 = 0.333
```

Things to know about strings

String delimiter symbol

Strings in Firebird are delimited by a pair of single quote (apostrophe) symbols: 'I am a string' (ASCII code 39, not 96). If you used earlier versions of Firebird's relative, InterBase®, you might recall that double and single quotes were interchangeable as string delimiters. Double quotes cannot be used as string delimiters in Firebird SQL statements.

Apostrophes in strings

If you need to use an apostrophe inside a Firebird string, you can “escape” the apostrophe character by preceding it with another apostrophe. For example, this string will give an error:

```
'Joe´s Emporium'
```

because the parser encounters the apostrophe and interprets the string as 'Joe' followed by some unknown keywords. To make it a legal string, double the apostrophe character:

```
'Joes´´ Emporium'
```

Notice that this is TWO single quotes, not one double-quote.

Concatenation of strings

The concatenation symbol in SQL is two “pipe” symbols (ASCII 124, in a pair with no space between). In SQL, the “+” symbol is an arithmetic operator and it will cause an error if you attempt to use it for concatenating strings. The following expression prefixes a character column value with the string “Reported by: ”:

```
'Reported by: ' || LastName
```

Firebird will raise an error if the result of a string concatenation exceeds the maximum (var)char size of 32 Kb.

If only the potential result – based on variable or field size – is too long you'll get a warning, but the operation will be completed successfully. (In pre-2.0 Firebird, this too would cause an error and halt execution.)

See also the section below, Expressions involving NULL, about concatenating in expressions involving NULL.

Double-quoted identifiers

Before the SQL-92 standard, it was not legal to have object names (identifiers) in a database that duplicated keywords in the language, were case-sensitive or contained spaces. SQL-92 introduced a single new standard to make any of them legal, provided that the identifiers were defined within pairs of double-quote symbols (ASCII 34) and were always referred to using double-quote delimiters.

The purpose of this “gift” was to make it easier to migrate metadata from non-standard RDBMSes to standards-compliant ones. The down-side is that, if you choose to define an identifier in double quotes, its case-sensitivity and the enforced double-quoting will remain mandatory.

Firebird does permit a slight relaxation under a very limited set of conditions. If the identifier which was defined in double-quotes:

1. was defined as all upper-case,
2. is not a keyword, and
3. does not contain any spaces,

…then it can be used in SQL unquoted and case-insensitively. (But as soon as you put double-quotes around it, you must match the case again!)

Warning: Don't get too smart with this! For instance, if you have tables “TESTTABLE” and “TestTable”, both defined within double-quotes, and you issue the command:

```
SQL>select * from TestTable;
```

…you will get the records from “TESTTABLE”, not “TestTable”!

Unless you have a compelling reason to define quoted identifiers, it is usually recommended that you avoid them. Firebird happily accepts a mix of quoted and unquoted identifiers – so there is no problem including that keyword which you inherited from a legacy database, if you need to.

Warning: Some database admin tools enforce double-quoting of all identifiers by default. Try to choose a tool which makes double-quoting optional.

Expressions involving NULL

In SQL, NULL is not a value. It is a condition, or *state*, of a data item, in which its value is unknown. Because it is unknown, NULL cannot behave like a value. When you try to perform arithmetic on NULL, or involve it with values in other expressions, the result of the operation will almost always be NULL. It is not zero or blank or an “empty string” and it does not behave like any of these values.

Below are some examples of the types of surprises you will get if you try to perform calculations and comparisons with NULL.

The following expressions all return NULL:

- 1 + 2 + 3 + NULL
- not (NULL)
- 'Home ' || 'sweet ' || NULL

You might have expected 6 from the first expression and “Home sweet ” from the third, but as we just said, NULL is not like the number 0 or an empty string – it's far more destructive!

The following expression:

- FirstName || ' ' || LastName

will return NULL if either FirstName or LastName is NULL. Otherwise it will nicely concatenate the two names with a space in between – even if any one of the variables is an empty string.

Tip: Think of NULL as *UNKNOWN* and these strange results suddenly start to make sense! If the value of Number is unknown, the outcome of '1 + 2 + 3 + Number' is also unknown (and therefore NULL). If the content of MyString is unknown, then so is 'MyString || YourString' (even if YourString is non-NULL). Etcetera.

Now let's examine some PSQL (Procedural SQL) examples with if-constructs:

- if (a = b) then

```
MyVariable = 'Equal';
else
MyVariable = 'Not equal';
```

After executing this code, MyVariable will be 'Not equal' if both a and b are NULL. The reason is that 'a = b' yields NULL if at least one of them is NULL. If the test expression of an “if” statement is NULL, it behaves like false: the 'then' block is skipped, and the 'else' block executed.

Warning: Although the expression may behave like false in this case, it's still NULL. If you try to invert it using not(), what you get is another NULL – not “true”.

- if (a <> b) then

```
MyVariable = 'Not equal';
else
MyVariable = 'Equal';
```

Here, MyVariable will be 'Equal' if a is NULL and b isn't, or vice versa. The explanation is analogous to that of the previous example.

The DISTINCT keyword comes to the rescue!

Firebird 2 implements a new use of the DISTINCT keyword allowing you to perform (in)equality tests that take NULL into account. The semantics are as follows:

- Two expressions are DISTINCT if they have different values or if one is NULL and the other isn't;
- They are NOT DISTINCT if they have the same value or if both are NULL.

Notice that if neither operand is NULL, DISTINCT works exactly like the “<>” operator, and NOT DISTINCT like the “=” operator.

DISTINCT and NOT DISTINCT always return true or false, never NULL.

Using DISTINCT, you can rewrite the first PSQL example as follows:

```
if (a is not distinct from b) then
MyVariable = 'Equal';
else
MyVariable = 'Not equal';
```

And the second as:

```
if (a is distinct from b) then
MyVariable = 'Not equal';
else
MyVariable = 'Equal';
```

These versions will give you the results that a normal human being (untouched by SQL standards) would expect, whether there are NULLs involved or not.