Guía de NULL en Firebird

Comportamiento y dificultades de `NULL` en Firebird

Paul Vinkenoog

Traducción al castellano de Víctor Zaragoza

22 de julio de 2005 Versión de documento 0.2-es

Introducción

Frecuentemente, preguntas de soporte en las listas de correo de Firebird hablan de "cosas raras" que pasan con los valores NULL en el SQL de Firebird. El concepto parece difícil de entender – quizá en parte por el nombre, que sugiere una "nada" que no puede hacer ningún daño si se lo sumas a un número o lo concatenas al final de un string. En realidad, realizar esas operaciones devolverán también la expresión NULL.

Este artículo explora el comportamiento de NULL en el SQL de Firebird, apunta fallos comunes y te enseña como manejar de manera segura expresiones que contengan NULL o resulten en NULL.

> Nota:
>
> Algunas sentencias y ejemplos en esta guía han sido tomados de *Firebird Quick Start Guide*, inicialmente publicado por IBPhoenix, ahora parte de Firebird Project.

Qué es `NULL`?

En SQL, `NULL` no es un valor. Es un *estado* que indica que el valor de ese item es desconocido o no existente. No es cero o blanco o una "cadena vacía" y no se comporta como ninguno de esos valores. Pocas cosas en SQL llevan a tanta confusión como `NULL`, y será difícil de entender mientras no entiendas la siguiente simple definición: `NULL` significa *desconocido*.

Déjame repetirlo:

`NULL` significa *desconocido*.

Retén esta línea en tu mente mientras leas el resto de este artículo y verás como muchos de los resultados que parecen absolutamente ilógicos que obtengas con `NULL`, prácticamente se autoexplicarán.

`NULL` en expresiones

Como muchos de nosotros hemos encontrado, para nuestro disgusto, `NULL` es contagioso: úsalo en una expresión numérica, texto o fecha/hora y el resultado siempre es `NULL`. Úsalo en una expresión lógica y el resultado depende del tipo de operación y el resto de valores implicados.

Por cierto, nota que en versiones anteriores a Firebird 2.0 es normalmente ilegal usar la constante `NULL` directamente en operaciones o comparaciones. Cuando veas `NULL` en las expresiones siguientes, léelas como "un campo, variable u otra expresión que resuelve en `NULL`".

Expresiones que devuelven NULL

Las expresiones en esta lista *siempre* devuelven NULL.

- `1 + 2 +3 + NULL`
- `'Hogar ' || 'dulce ' || NULL`
- `MiCampo = NULL`
- `MiCampo <> NULL`
- `NULL = NULL`
- `not (NULL)`

Si tienes dificultades en entender por qué, recuerda que NULL significa "desconocido". Además mira en la siguiente tabla donde hay explicaciones por caso. En la tabla, no hemos escrito NULL en las expresiones (como ya hemos dicho, es, a menudo, ilegal); en vez de ello, hemos usado dos entidades A y B que son ambas NULL. A y B pueden ser campos, variables o subexpresiones enteras en su derecho – como son NULL, se comportan de la misma manera que expresiones cerradas.

Tabla 1. Operaciones sobre entidades NULL A y B

Si A y B son NULL, entonces:	**Es:**	**Porque:**
`1 + 2 + 3 + A`	NULL	Si A es desconocido, entonces 6+A también es desconocido
`'Hogar ' || 'dulce ' || NULL`	NULL	Si A es desconocido, entonces 'Dulce hogar' || A es también desconocido
`MiCampo = A`	NULL	Si A es desconocido, no puedes decir que MiCampo tenga el mismo valor…
`MiCampo <> A`	NULL	...ni puedes decir que MiCampo tenga distinto valor
`A = B`	NULL	Con A y B desconocidos, es imposible saber si son iguales
`Not (A)`	NULL	Si A es desconocido, su inverso también

NULL en expresiones booleanas

Hemos visto que `not(NULL)` devuelve NULL. Para los operadores `and` y `or`, las cosas son un poco más complicadas:

- `NULL or false = NULL`
- `NULL or true = true`
- `NULL or NULL = NULL`
- `NULL and false = false`
- `NULL and true = NULL`
- `NULL and NULL = NULL`

El SQL de Firebird, no tiene un dato de tipo booleano (lógico); no hay unas constantes true o false existentes. En la columna de la izquierda de la siguiente tabla `(true)` and `(false)` representan expresiones que devuelven true/false;

Tabla 2. Operaciones lógicas (booleanas) sobre una entidad NULL A

Si A es NULL, entonces:	Es:	Porque:
`A or false`	`NULL`	"A or false" siempre tiene el valor de A, que es desconocido
`A or true`	`True`	"A or true" siempre es true. El valor de A no importa
`A or A`	`NULL`	"A or A" siempre equivale a A, que es NULL
`A and false`	`False`	"A y false" es siempre false. El valor de A no importa
`A and true`	`NULL`	"A y true" siempre tiene el valor de A el cuál es desconocido
`A and A`	`NULL`	"A and A" siempre equivale a A, que es NULL

Todas estas expresiones están en concordancia con la lógica booleana. El hecho de que, para calcular "`X or true`" y "`X and false`", simplemente no *necesites* saber el valor de X, es también la base de una característica que conocemos en varios lenguajes de programación: evaluación de circuitos cortos booleanos.

Más lógica (o no)

Los resultados de circuitos cortos obtenidos arriba, pueden llevarte a las siguientes ideas:

- 0 veces X equivale a 0 para cada X. Por tanto, igual que el valor de X es desconocido, 0 * X es 0. (Nota: esto sólo sucede si el tipo de dato de X sólo contiene números, no NaN o infinitos).
- La cadena vacía está ordenada lexicográficamente antes de cada otra cadena. Por tanto S >= '' es verdad siempre independientemente del valor de S.
- Cada valor equivale a sí mismo, independientemente de si es desconocido o no. Por tanto, aunque A = B justificadamente se devuelve NULL si A y B son entidades NULL diferentes, A = A siempre devuelve true, igual que A es NULL.

¿Cómo está esto implementado en el SQL de Firebird? Bueno, siento informarte que, a pesar de esta convincente lógica – y la analogía con los resultados explicados arriba – las siguientes expresiones se resuelven todas con NULL:

- `0 * NULL`
- `NULL >= ''`
- `'' <= NULL`
- `A = A` (con A como un campo o variable NULL)

Demasiado para la coherencia.

`NULL` en funciones agregadas

En funciones agregadas como `COUNT`, `SUM`, `AVG`, `MAX`, y `MIN`, `NULL` se maneja de manera diferente: para calcular el resultado, sólo se tienen en consideración los campos con valores no-`NULL`. Esto es, si tienes esta tabla:

```
MiTabla
ID          Nombre          Sueldo
1           Juan            37
2           Perico          <NULL>
3           Andrés          5
4           Roberto         12
5           Antonio         <NULL>
```

… la sentencia `select sum(Sueldo) from MiTabla` devuelve 54, que es 37+5+12. Si sumáramos todos los valores, el resultado debería haber sido `NULL`. Para `AVG`, los campos no-`NULL` son sumados y la suma dividida entre el número de campos no-`NULL`.

Hay una excepción en esta regla: `COUNT(*)` devuelve el número de todas las filas, incluidas todas aquellas con campos en `NULL`. Pero `COUNT(NombreDeCampo)` se comporta como las otras funciones agregadas y cuenta aquellas filas que tienen campos con contenido no-`NULL`.

Otra cosa a tener en cuenta: `COUNT(*)` y `COUNT(NombreDeCampo)` jamás devuelven `NULL`: si no hay filas en el grupo, ambas funciones devuelven 0. Además, `COUNT(NombreDeCampo)` devuelve 0 si todos los `NombreDeCampo` del grupo son `NULL`. Las otras funciones agregadas devuelven `NULL` en tales casos. Ten en cuenta que `SUM` devuelve `NULL` si se usa en un grupo de registros vacío, lo que es contrario a la lógica común.

Manejo de `NULL` en UDF's

Las UDF's (User Defined Functions o Funciones Definidas por el Usuario) son funciones no internas en el motor, por tanto definidas en módulos separados. Firebird viene con dos librerías UDF: ib_udf (heredada de Interbase) y fbudf. Puedes agregar más librerías, por ejemplo comprándolas o descargándotelas o bien escribiéndolas tú mismo. Las UDF no pueden ser usadas "fuera de la caja", deben ser "declaradas" en la base de datos primero. Esto es cierto incluso con las UDF que vienen con Firebird.

Conversiones `NULL` <-> `no-NULL` no deseadas

Enseñarte como declarar, escribir y usar UDF's está fuera del ámbito de esta guía. No obstante, debemos avisarte de que las UDF's pueden efectuar, ocasionalmente, conversiones `NULL` no esperadas. Esto puede, en ocasiones, resultar de introducir un valor `NULL` y su conversión a un valor normal y en otra ocasiones en la "nulificación" de un valor válido de entrada '' (una cadena vacía).

La principal causa de este problema es la llamada "a la antigua usanza" de la UDF. No es posible pasar un valor `NULL` como entrada a la función. Cuando una UDF como `LTRIM` (recortar por la izquierda) se llama con un argumento `NULL`, el argumento es pasado a la función como una cadena vacía. Desde dentro de la función no hay manera de decir si el argumento pasado es una cadena vacía real o un valor `NULL`. ¿Entonces

qué hace el implementador de la función? Tiene que tomar una decisión: o bien tomar el argumento como entró o asumir que es un NULL y tratarlo acorde a ello.
Dependiendo del tipo de dato del resultado, devolver NULL es posible igual que recibirlo como argumento no lo es. De este modo, estas situaciones inesperadas pueden suceder:

- Puedes llamar a una UDF con un argumento NULL, ésta lo recibe como un 0 o como una cadena vacía ‘’. Por medio de la función, este valor no vuelve a ser NULL: el resultado es un valor no-NULL.
- Puedes llamar a una UDF con un argumento válido como 0 o ‘’ (cadena vacía). Se pasa el argumento “como es” (obviamente), pero la función considera que el argumento realmente representa un NULL, lo trata como un agujero negro y devuelve un NULL.

Ambas conversiones no son deseadas, pero la segunda probablemente más que la primera (mejor validar algo que es NULL que “nulificar” un valor válido). Volviendo a nuestro ejemplo con LTRIM: hasta e incluyendo Firebird 1.0.3, esta función devolvía NULL si se le pasaba como argumento una cadena vacía. Desde la versión 1.5, nunca devuelve NULL. En estas recientes versiones, las cadenas NULL son “TRIMeadas” a cadenas vacías – lo cual es incorrecto, pero considerado el menos malo de los dos casos; en el caso anterior, cadenas vacías válidas eran graciosamente convertidas a cadenas NULL.

Preparándose para conversiones no deseadas

Las conversiones no deseadas descritas anteriormente, sólo suceden en UDF’s heredadas, pero hay muchas de ellas (principalmente en ib_udf). Además, nada puede evitar que un desarrollador haga lo mismo en funciones de nuevo estilo. Por tanto, si usas una UDF y no sabes cómo se comporta respecto a los valores NULL:

1. Mira su declaración para ver cómo los valores son pasados y devueltos. Si dice “by descriptor”, debe ser segura (aunque nada se puede decir que sea 100% seguro). En otro caso, sigue el resto de pasos.

2. Si tienes el código fuente y sabes leer C/C++, inspecciona el código.

3. Prueba la función con argumentos de entrada NULL y valores 0 (para numéricos) y ‘’ para (cadenas).

4. Si la función devuelve una conversión NULL <-> no-NULL no deseada, tienes que trabajar en tu código antes de llamar a la UDF (mira también Comprobando si algo es NULL, también en esta guía).

Las declaraciones de las UDF proporcionadas pueden encontrarse en la carpeta bin/examples(v.1.0) o bin/UDF(v.1.5 y superiores) de la instalación de Firebird. Mira en los archivos con extensión .sql

Más sobre UDF's

Para aprender más sobre UDF's, consulta la *InterBase 6.0 Developer's Guide* (libre en http://ibphoenix.com/downloads/60DevGuide.zip), *Using Firebird* y la *Firebird Referente Guide* (ambas en CD), o el *Firebird book*. El libro y el CD se pueden comprar vía http://www.ibphoenix.com.

`NULL` en sentencias `if`

Si la condición de una sentencia if devuelve un `NULL`, la cláusula `then` se salta y (si existe) se ejecuta la cláusula `else`. Pero ¡cuidado! La condición debe *comportarse* como `false` en este caso, pero no tiene el *valor* `false`. Es todavía `NULL`, y pueden suceder cosas extrañas si lo olvidas. Los siguientes ejemplos exploran algunas de las obras diabólicas de `NULL` en las sentencias `if`.

- ```
  if (a = b) then
    MiVariable = 'Igual';
  Else
    MiVariable = 'No Igual';
  ```

Si a y b son ambas `NULL`, `MiVariable` será `'No Igual'` al ejecutar este código. La razón es que la expresión "a = b" devuelve `NULL` si, al menos, uno de ellos es `NULL`. Con esta condición `NULL`, el bloque `then` es ignorado y el `else` es ejecutado.

- ```
  if (a <> b) then
    MiVariable = 'No Igual';
  Else
    MiVariable = 'Igual';
  ```

Aquí, `MiVariable` será `'Igual'` si `a` es `NULL` y `b` no o viceversa. La explicación es análoga al ejemplo anterior.

- ```
  if (not (a <> b)) then
    MiVariable = 'Igual';
  Else
    MiVariable = 'No Igual';
  ```

Esta otra parece que devuelve el mismo resultado que la anterior, ¿Es así? Después de todo, hemos invertido la condición e intercambiado las cláusulas `then` y `else`. Y, de hecho, siempre que ninguna variable es `NULL`, ambos códigos son equivalentes. Pero siempre que a o b son `NULL`, lo es la expresión entera y se ejecuta la cláusula else y el resultado es `'No Igual'`.

> Nota:
>
> Por supuesto, somos conscientes de que este tercer ejemplo es totalmente equivalente al primero. Simplemente lo hemos incluido para hacer hincapié en que not(NULL) es NULL. Por tanto, en condiciones que resulten en NULL, not () no las invierte.
  ```

Comprobando si algo es NULL

Para resolver los estragos que NULL puede causar, a menudo tienes que saber si algo es NULL antes de usarlo en una expresión. Para ello, lo obvio sería esto:

```
If (A = NULL) then
```

De hecho, hay sistemas de control de bases de datos que soportan esta sintaxis para determinarlo. Pero el SQL estándar no lo soporta y Firebird tampoco. En versiones anteriores a la 2.0, la expresión completa es ilegal. A partir de la 2.0 está permitida, pero la comparación siempre devuelve NULL, independientemente del valor de A o su estado. Es, por tanto, inútil; lo que necesitamos es un resultado limpio `true` o `false`.

El modo correcto para testear NULL es:

```
… is null/ … is not null
```

Estas comprobaciones devolverán siempre true o false sin errores. Ejemplos:

- `if (MiCampo is null) then …`
- `select * from Alumnos where Telefono is not null`
- `select * from Alumnos where not (Telefono is null)`
  `/* hace lo mismo que el ejemplo anterior*/`
- `update Numero set Total = A + B + C where A + B + C is not null`

Se puede decir que mientras **"="** (cuando se usa como un operador de igualdad) puede comparar sólo valores, **"is"** compara estados.

Colocando un campo o variable a NULL

Los campos y variables pueden ser puestos a null usando la misma sintaxis que el resto de valores normales:

- `insert into MiTabla values (1, 'cadena', NULL, '1/2/2005')`
- `update MiTabla set MiCampo = NULL where TuCampo = -1`
- `if (Numero=0) then MiVariable = NULL;`

– "Un momento… ¡¡antes dijiste que MiCampo = NULL era illegal!!"

Es correcto… para el *operador de comparación "="* (como mínimo en las versiones pre-2.0 de Firebird). Pero aquí estamos hablando de "=" como un *operador de asignación*. Desafortunadamente, ambos operadores comparten el mismo símbolo en SQL. En asignaciones, si están hechas con "=" o con una lista de inserción, puedes tratar tanto con NULL como con cualquier otro valor – no se necesita una sintaxis especial (o de hecho, es posible).

Trabajando con NULLs

Esta sección contiene trucos prácticos y ejemplos que pueden ser usados por ti en tu trabajo diario con NULLs.

Comprobando NULL – Si es necesario

El mayor número de ocasiones, no tendrás que tomar especiales medidas para campos o variables que puedan ser NULL. Por ejemplo, si haces esto:

```
Select * from Clientes where Ciudad = 'Valencia'
```

probablemente no quieras ver los clients cuya ciudad no está especificada. Igualmente:

```
if (Edad >= 18) then PuedeVotar = 'Si';
```

no incluye gente que tenga la edad desconocida, que es también defendible. Pero:

```
if (Edad >= 18) then PuedeVotar = 'Yes';
else PuedeVotar = 'No';
```

no parece correcto: si no sabes la edad de una persona, no puedes explícitamente denegarle el derecho a votar. Peor es esto:

```
if (Edad < 18) then PuedeVotar = 'No';
else PuedeVotar = 'Si';
```

no tiene el mismo efecto que la anterior. Si algunos de las edades NULL son, en realidad, menores de 18, ¡les estás dando derecho a votar!

La manera correcta es comprobar el valor NULL específicamente:

```
If (Edad is null) then PuedeVotar = 'No definido';
Else
  If (Edad >= 18) then Puedevotar = 'Si;
  Else PuedeVotar = 'No';
```

> Nota:
>
> `Else` siempre se refiere al último `if` en el mismo bloque. Pero es a menudo bueno evitar confusiones poniendo las palabras clave `begin … end` en los grupos de líneas. No lo he hecho aquí para escribir el menor número de líneas posible. Y entonces he compensado esa economización escribiendo esta nota ;-)

Comprobando que los campos son iguales

A veces quieres saber si dos campos o dos variables son lo mismo y quieres considerarlas lo mismo si ambas son `NULL`. La manera correcta de comprobarlo es esta:

```
If (A = B or A is null and B is null) then …
```

O, si quieres situar la preferencia de operaciones explícitamente:

```
If ((A = B) or (A is null and B is null)) then …
```

¡Atención!: Si solo una de A o B es NULL, la comparación resulta NULL, no false. Es correcto en una sentencia if, pero podemos añadir una cláusula else que se ejecutará si A y B no son iguales (incluyendo cuando una es NULL y la otra no).

```
If (A = B or A is null and B is null)
  Then … código a añadir si A es igual a B
  Else … código a añadir si A y B son diferentes
```

Pero no tengas la brillante idea de invertir la expresión y usarla en una comparación de no igualdad (como yo hice una vez):

```
/* ¡¡¡No hagas esto!!!*/
If (not(A = B or A is null and B is null))
  Then … código a añadir si A es distinto a B
```

El código anterior funciona correctamente si A y B son ambas NULL o ambas no-NULL. Pero falla al ejecutar la cláusula then si sólo una de ellas es NULL.

Si sólo quieres que se haga algo si A y B son diferentes, usa cualquiera de las expresiones anteriores y escribe una instrucción muerta (que no causará ningún efecto) en la cláusula then, o usa esta expresión:

```
/* Esta es una expresión correcta de no igualdad */
If (A <> B)
   Or A is null and B is not null
   Or A is not null and B is null) then …
```

Comprobando que un campo se ha cambiado

En triggers, es muy útil saber si cierto campo ha cambiado (incluyendo que se haya convertido de NULL a no-NULL o viceversa) o sigue siendo igual. Esto es sólo un caso especial de comprobar la (des)igualdad de dos campos. Usa new.nombrecampo y old.nombrecampo para A y B.

```
If (new.trabajo = old.trabajo or new.trabajo is null
and old.trabajo is null)
  Then … El campo trabajo es igual
  Else … El campo trabajo ha cambiado
```

Substituyendo NULL por un valor

La función COALESCE

Firebird 1.5 tiene una función que puede convertir NULL a cualquier otra cosa. Esto te habilita para hacer conversiones "on-the-fly" y usar el resultado en un tratamiento posterior, sin la necesidad de utilizar la construcción "if (MiExpresion is null) then". La función se llama COALESCE y la puedes llamar así:

```
COALESCE (Expr1, Expr2, Expr3, …)
```

COALESCE devuelve el primer valor no-NULL de la lista de argumentos. Si todas las expresiones son NULL, devuelve NULL.

Así es como puedes usar COALESCE para construir el nombre completo de una persona desde el nombre, segundo nombre y primer apellido, asumiendo que algunos datos puedan estar en NULL:

```
Select Nombre || coalesce(' ' || Nombre2, '') || ' ' ||
Apellido from Personas
```

O para crear un nombre tan informal como sea posible de una tabla que incluya también apodos y asumiendo que tanto el apodo como el primer nombre pueden ser NULL:

```
SELECT COALESCE (Apodo, nombre, 'Sr./Sra.') || ' ' ||
Apellido from Personas
```

COALESCE te ayudará sólo en situaciones donde NULL puede ser tratado como otros valores posibles del tipo de dato. Si NULL necesita un manejo especial, como en el ejemplo de "derecho a votar" usado anteriormente, tu única opción es usar "if (MiExpresion is null) then".

Firebird 1.0: las funciones *NVL

Firebird 1.0 no tiene COALESCE. Sin embargo, puedes usar cuatro UDF's que proveen una buena parte de su funcionalidad. Estas UDF's residen en la librería fbudf y son:

- iNVL, para argumentos enteros
- i64NVL, para argumentos enteros largos
- dNVL, para argumentos de doble precisión
- sNVL, para argumentos de cadena

Las funciones *NVL tienen exactamente dos argumentos. Como COALESCE, devuelven el primero argumento si es no-NULL; en otro caso, devuelven el segundo. Recuerda por favor, que la librería fbudf – y por tanto las funciones *NVL – sólo están disponibles para Windows.

Sumario

NULL en pocas palabras:

- NULL significa *desconocido*
- Si NULL figura en una expresión, la mayoría de las veces el resultado es NULL.
- En funciones agregadas sólo los valores no-NULL se involucran en la operación. Excepción: COUNT(*).
- A veces, las UDF's convierten NULL <-> no-NULL de manera que parece aleatoria.
- Si la comprobación de una expresión de una sentencia IF es NULL, el bloque then se ignora y se ejecuta el bloque else.
- Para saber si A es NULL, use "A is (not) null".
- Las funciones COALESCE y *NVL pueden convertir un NULL en un valor.
- Asignar NULL se hace igual que asignar valores: con "A=NULL" o una lista de inserción.

Recuerda, esto es como funciona NULL en *Firebird SQL*. Hay diferencias con otras RDBMS's.

Apéndice A: Histórico de documentos

El archivo exacto con el histórico está en el módulo de manual en nuestro servidor CVS; mira en **http://sourceforge.net/cvs/?group_id=9028**

Histórico de revisiones

0.1	8 Abril 2005	PV	Primera Edición
0.2	15 Abril 2005	PV	Menciona que Fb 2.0 legaliza las comparaciones "`A = NULL`". Texto cambiado en "*Comprobando si algo es `NULL`*". Ligeramente cambiada la sección "*Manejando `NULL's`*".
0.2-es	22 Julio 2005	VZ	Traducción al castellano por Víctor Zaragoza.

Apéndice B: Licencia

La documentación original se llama *Firebird Null Guide*.

El escritor inicial de la documentación original es: Paul Vinkenoog.

La traducción inicial se llama *Guía de NULL en Firebird*.

El traductor inicial de la documentación en castellano es: Víctor Zaragoza.

Contacto con el escritor inicial: paulvink en users dot sourceforge dot net

Contacto con el traductor inicial: victorxata en users dot sourceforge dot net