



# **Guide Firebird et NULL**

Comportement et précautions au sujet de `NULL` dans Firebird SQL

Paul Vinkenoog

Traduction en français: Philippe Makowski

04 mai 2005 - Document version 0.2.1-fr

---

---

## Table des matières

Introduction .....	3
Qu'est ce que NULL? .....	3
NULL dans les expressions .....	3
Expressions retournant NULL .....	3
NULL dans des expressions booléennes .....	4
Plus de logique (ou pas) .....	5
NULL dans des fonctions d'agrégat .....	5
NULL dans les UDFs .....	6
NULL <-> non-NULL conversions non demandées .....	6
Soyez préparés aux conversions non voulues .....	7
Plus d'information sur les UDF .....	7
NULL dans des instructions if .....	8
Tester si quelque chose est NULL .....	8
Affecter NULL à une variable ou un champ .....	9
Travailler avec NULL .....	9
Tester NULL - si cela est nécessaire .....	9
Vérifier que des champs sont égaux .....	10
Substituer une valeur à NULL .....	11
Résumé .....	12
A. Historique du document .....	13
B. Licence .....	14

---

## Introduction

Régulièrement, dans les listes de support sur Firebird des questions au sujet de « choses étranges » arrivant avec `NULL` dans Firebird SQL sont posées. Ce concept semble difficile à comprendre - peut être à cause de son nom, qui semble correspondre à « rien » qui donc n'aurait aucune conséquence s'il est ajouté à un nombre ou ajouté à la fin d'une chaîne de caractères. En fait, le résultat de ces opérations renvoie toujours `NULL`.

Cet article explore le comportement de `NULL` dans Firebird SQL, pointe les chausse-trappe habituelles et vous montre comment travailler avec des expressions qui contiennent `NULL` ou peuvent avoir un résultat `NULL`.

Si vous voulez une référence rapide pour rafraîchir votre mémoire, allez directement à ce [résumé](#) (qui est vraiment condensé).

### Note

Certaines phrases et exemples dans ce guide sont empruntés au *Guide de démarrage de Firebird*, publié initialement par IBPhoenix, et maintenant partie intégrante du Projet Firebird.

## Qu'est ce que `NULL` ?

Dans SQL, `NULL` n'est pas une valeur. C'est un *état* indiquant que la valeur d'un item est inconnue ou inexistante. Ce n'est ni zéro ni blanc ni une « chaîne vide » et cela ne se comporte comme aucune de ces valeurs. Peu de choses sèment autant le trouble dans SQL que `NULL`, et pourtant son mécanisme est simple à comprendre quand on suit cette simple définition suivante : `NULL` signifie *indéterminé*.

Laisser moi le redire :

**`NULL` signifie INDÉTERMINÉ**

Gardez cette définition en tête quand vous lirez le reste de cet article, et tout ce qui vous semblait illogique dans le comportement de `NULL` s'expliquera quasiment de lui même.

## `NULL` dans les expressions

Combien d'entre nous ont appris à leurs dépens que `NULL` est contagieux: utilisez le dans une expression numérique, de chaîne de caractère ou de date/heure, et le résultat sera toujours `NULL`. Utilisez le dans une expression booléenne, et le résultat dépendra du type d'opération et des autres valeurs impliquées.

Notez au passage que dans les versions de Firebird avant 2.0, il est le plus souvent illégal d'utiliser la constante `NULL` directement dans des opérations ou comparaisons. Partout où vous verrez `NULL` dans les instructions suivantes, lisez le comme « un champ, une variable ou une autre expression renvoyant `NULL` ».

## Expressions retournant NULL

Les expressions dans cette liste renvoient *toujours* NULL:

- `1 + 2 + 3 + NULL`
- `'Home ' || 'sweet ' || NULL`
- `MyField = NULL`
- `MyField <> NULL`
- `NULL = NULL`
- `not (NULL)`

Si vous avez des difficultés à comprendre pourquoi, souvenez vous que NULL signifie « indéterminé ». Regardez aussi le tableau suivant où des explications sont données au cas par cas. Dans le tableau nous n'écrivons pas NULL dans les expressions (comme dit plus haut, c'est souvent interdit); à la place, nous utilisons deux entités A et B qui sont toutes deux NULL. A et B peuvent être des champs, des variables, ou des sous-expressions - du moment quelles sont NULL, elles ont le même comportement dans les expressions utilisées.

**Tableau 1. Opérations des entités A et B null**

Si A et B sont NULL, alors:	Est:	Parce que:
<code>1 + 2 + 3 + A</code>	NULL	Si A est indéterminée, alors <code>6 + A</code> est aussi indéterminée.
<code>'Home '    'sweet '    A</code>	NULL	Si A est indéterminée, <code>'Home sweet '    A</code> est aussi indéterminée.
<code>MonChamp = A</code>	NULL	Si A est indéterminée, vous ne pouvez dire si <code>MonChamp</code> à la même valeur...
<code>MonChamp &lt;&gt; A</code>	NULL	...mais vous ne pouvez non plus dire si <code>MonChamp</code> à une valeur <i>différente</i> !
<code>A = B</code>	NULL	Avec A et B indéterminées, il est impossible de savoir si elles sont égales.
<code>not (A)</code>	NULL	Si A est indéterminée, sa négation est aussi indéterminée.

## NULL dans des expressions booléennes

Non avons déjà vu que `not (NULL)` renvoie NULL. Pour les opérateurs `and` et `or`, les choses sont un peu plus compliquées :

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

Le SQL Firebird n'a pas de type booléen; les constantes `true` et `false` n'existent pas. Dans la co-

lonne de gauche du tableau suivant, (`true`) et (`false`) représentent des expressions renvoyant `true/false`.

**Tableau 2. Opérations booléennes sur une entité A null**

Si A est NULL, alors:	Est:	Parce que:
A or ( <code>false</code> )	NULL	« A or false » a toujours la même valeur que A - qui est indéterminée.
A or ( <code>true</code> )	<code>true</code>	« A or true » est toujours <code>true</code> - la valeur de A n'a pas d'importance.
A or A	NULL	« A or A » est toujours égal à A - qui est NULL.
A and ( <code>false</code> )	<code>false</code>	« A and false » est toujours <code>false</code> - la valeur de A n'a pas d'importance.
A and ( <code>true</code> )	NULL	« A and true » a toujours la même valeur que A - qui est indéterminée.
A and A	NULL	« A and A » est toujours égal à A - qui est NULL.

Tous ces résultats sont en accord avec la logique booléenne. Le fait que, pour calculer « X or `true` » et « X and `false` », vous n'avez simplement pas *besoin* de connaître la valeur de X, fait partie des bases d'une fonctionnalité que nous connaissons dans divers langages de programmation : l'évaluation booléenne rapide.

## Plus de logique (ou pas)

Les résultats obtenus ci-dessus pourraient vous mener aux idées suivantes :

- 0 fois x égal 0 pour tout x. En conséquence, même si la valeur de x est indéterminée, `0 * x` égal 0. (Note: ceci seulement si x est d'un type n'acceptant que des nombres, pas NaN ou infini.)
- Un chaîne vide est classée lexicographiquement avant toute autre chaîne. Donc, `S >= ''` est vrai quelque soit la valeur de S.
- Chaque valeur est égale à elle même, qu'elle soit connue ou pas. Donc, bien que `A = B` renvoie NULL si A et B sont des entités NULL différentes, `A = A` devrait toujours retourner `true`, même si A est NULL.

Comment cela est-il implémenté dans le SQL de Firebird ? Et bien, je suis désolé de vous dire qu'en dépit de toute cette logique - et les analogies avec les résultats booléens discutés ci-dessus - toutes ces expressions renvoient NULL:

- `0 * NULL`
- `NULL >= ''`
- `'' <= NULL`
- `A = A` (avec A comme champ ou variable null)

Et voilà pour la logique.

## NULL dans des fonctions d'agrégat

Dans des fonctions d'agrégat comme COUNT, SUM, AVG, MAX, et MIN, NULL est géré différemment: pour calculer le résultat, seuls les champs non-NULL sont pris en considération. C'est à dire que, si vous avez cette table:

### MyTable

ID	Name	Amount
1	John	37
2	Jack	<NULL>
3	Joe	5
4	Josh	12
5	Jay	<NULL>

...l'instruction `select sum(Amount) from MyTable` renvoie 54, qui est  $37 + 5 + 12$ . Si les cinq champs avaient été additionnés, le résultat aurait été NULL. Pour AVG, les champs non-NULL sont additionnés et la somme est divisée par le nombre de champs non-NULL.

Il y a seulement une exception à cette règle: `COUNT(*)` renvoie le dénombrement de toutes les lignes, même celles pour lesquelles les champs sont NULL. Mais `COUNT(FieldName)` se comporte comme les autres fonctions d'agrégat et ne compte que les enregistrements pour lesquels le champ spécifié est non NULL.

Une autre chose à savoir est que `COUNT(*)` et `COUNT(FieldName)` ne renvoient jamais NULL: s'il n'y a pas d'enregistrement dans l'ensemble de données les deux fonctions renvoient 0. Ainsi, `COUNT(FieldName)` renvoie 0 si tous les champs `FieldName` dans l'ensemble de données sont NULL. Les autres fonctions d'agrégat renvoient NULL dans ce cas. Faites attention que même SUM renvoie NULL s'il est utilisé sur un ensemble vide, ce qui est contraire à la logique commune.

## NULL dans les UDFs

UDF (*User Defined Functions*) sont des fonctions qui ne sont pas internes au moteur, mais définies dans des modules séparés. Firebird est livré avec deux bibliothèques UDF: `ib_udf` (héritée d'InterBase) et `fbudf`. Vous pouvez ajouter d'autres bibliothèques, e.g. en les achetant ou les téléchargeant, ou en les écrivant vous même. Les UDF ne peuvent être utilisées directement; elles doivent être « déclarées » dans la base d'abord. Ceci est aussi vrai pour les UDF livrées avec Firebird.

## NULL <-> non-NULL conversions non demandées

Vous apprendrez à déclarer, utiliser, et écrire des UDFs est hors du champ de ce guide. Toutefois, nous devons vous prévenir que les UDFs peuvent quelques fois effectuer des conversions de NULL non désirées. Cela va quelques fois convertir des entrées NULL en des valeurs régulières, et quelques fois rendre NULL des entrées valides comme '' (une chaîne vide).

La cause principale de ce problème est qu'avec l'« ancien style » d'appel UDF, il n'est pas possible de passer NULL comme entrée de fonction. Quand une UDF comme LTRIM (left trim) est appelée avec un argument NULL, l'argument est passé à la fonction comme une chaîne vide. A l'intérieur de la fonction, il n'y a aucune manière de savoir si cet argument représente réellement une chaîne vide ou un NULL. Donc que fait le créateur de la fonction? Il doit choisir: soit prendre l'argument comme étant une valeur, ou considérer qu'à l'origine s'était un NULL et faire le traitement en conséquence.

En fonction du type de résultat, renvoyer NULL peut être possible même si recevoir NULL ne l'est pas. Ainsi, les choses non attendues suivantes peuvent arriver :

- Vous appelez une UDF avec un argument NULL. Il est passé comme une valeur, e.g. 0 ou ' '. Dans la fonction, cet argument n'est pas remis à NULL; un résultat non-NULL est renvoyé.
- Vous appelle une UDF avec un argument valide comme 0 ou ' '. Il es passé tel quel (manifestement). Mais le code de la fonction suppose que cette valeur en réalité représente un NULL, le traite comme tel, et renvoie NULL à l'appelant.

Ces deux conversion sont en général non désirées, mais la seconde certainement plus que la première (on préfère valider NULL que détruire quelque chose de valide). Pour revenir à notre fonction LTRIM par exemple: jusqu'à la version Firebird 1.0.3 incluse, cette fonction renvoyait NULL si vous lui donniez une chaîne vide; et depuis la version 1.5, elle ne renvoie jamais NULL. Dans ces versions récentes, les chaînes NULL sont « transformées » en chaînes vides. Ce n'est pas vraiment juste, mais c'est considéré comme la moins mauvaise solution: dans l'ancienne situation , les chaînes valides (mais vides) étaient sans merci ramenées à NULL.

## Soyez préparés aux conversions non voulues

Les conversions nous voulues décrites ci-dessus arrivent normalement qu'avec les UDFs héritées d'Interbase, mais il en existe beaucoup (essentiellement dans `ib_udf`). De même, rien n'empêchera un autre développeur de faire la même chose dans une nouvelle fonction. Donc la règle est: si vous utilisez une UDF et que vous savez pas comment elle se comporte avec NULL:

1. Regardez sa déclaration pour voir comment les valeurs sont passées et retournées. S'il est écrit « by descriptor », cela devrait aller (même si on ne peut en être certain). Dans tous les autres cas, suivez les instructions suivantes.
2. Si vous avez les sources et que vous savez les lire, regardez le code de la fonction.
3. Testez la fonction avec des entrées NULL et des entrées comme 0 (pour les arguments numériques) et/ou ' ' (pour les chaînes de caractères).
4. Si la fonction effectue une conversion non désirée de NULL <-> non-NULL , vous devrez en tenir compte dans votre code avant d'utiliser l'UDF (voir aussi [Tester si quelque chose est NULL](#) , dans ce guide).

Les déclarations pour les bibliothèques UDF livrées avec Firebird se trouvent dans le sous répertoire de Firebird `bin/examples` (v. 1.0) ou `bin/UDF` (v. 1.5 et suivantes). Regardez les fichiers avec l'extension `.sql`

## Plus d'information sur les UDF

Pour en savoir plus sur les UDF, consultez *InterBase 6.0 Developer's Guide* (disponible gratuitement à <http://www.ibphoenix.com/downloads/60DevGuide.zip>), *Using Firebird* et le *Firebird Reference Guide* (les deux sur CD), ou le *Firebird Book*. Les CD et le livre peuvent être achetés chez <http://www.ibphoenix.com>.

## NULL dans des instructions `if`

Si l'expression à tester dans l'instruction `if` est évaluée à `NULL`, la clause `then` est ignorée et la clause `else` (si elle est présente) exécutée. Attention! Cette expression semble se *comporter* comme `false` dans ce cas, mais elle n'a pas la *valeur* `false`. Elle est toujours `NULL`, et des choses curieuses peuvent arriver si vous oubliez cela. Les exemples suivants explorent quelques diaboliques comportements de `NULL` dans des instructions `if` :

- ```
if (a = b) then
  MyVariable = 'Egal';
else
  MyVariable = 'Different';
```

Si `a` et `b` sont tous deux `NULL`, `MyVariable` sera « `Different` » après l'exécution de ce code. Cela vient de ce que l'expression « `a = b` » renvoie `NULL` si au moins un des termes est `NULL`. Avec l'expression évaluée à `NULL`, le bloc `then` est ignoré, et le bloc `else` exécuté.

- ```
if (a <> b) then
  MyVariable = 'Different';
else
  MyVariable = 'Egal';
```

Ici, `MyVariable` sera « `Egal` » si `a` est `NULL` et `b` ne l'est pas, et vice versa. L'explication est analogue à celle de l'exemple précédent.

- ```
if (not (a <> b)) then
  MyVariable = 'Egal';
else
  MyVariable = 'Different';
```

Il semble que cet exemple devrait donner le même résultat que l'exemple précédent, n'est ce pas? Après tout, nous avons inversé l'expression de test et interverti les clauses `then` et `else`. Et de fait, tant qu'aucune variable est `NULL`, les deux codes sont équivalents. Mais dès que `a` ou `b` est `NULL`, alors toute l'expression l'est aussi, la clause `else` est exécutée, et le résultat est « `Different` ».

### Note

Bien sûr, nous savons que ce troisième exemple est équivalent au premier. Nous ne vous l'avons proposé que pour insister sur le fait que `not(NULL)` est `NULL`. Donc, dans les situations où l'expression testée est `NULL`, `not()` n'inverse pas le résultat.

## Tester si quelque chose est `NULL`

Puisque `NULL` peut vous poser des problèmes, vous aurez souvent à tester si quelque chose est `NULL` avant de l'utiliser dans une expression. La plupart du temps, on pense que le test le plus indiqué serait

```
if (A = NULL) then...
```

et de fait certains systèmes de gestion de base de données supportent cette syntaxe pour déterminer l'état `NULL`. Mais le standard SQL ne le permet pas, et Firebird non plus. Dans les version avant 2.0 cette syntaxe est même illégale. Depuis la 2.0 c'est permis, mais la comparaison retournera toujours

NULL, quels que soient l'état et la valeur de A. Et cela n'est pas très utile comme test - ce dont nous avons besoin est un résultat clair `true` ou `false` .

La façon correcte de tester NULL est:

```
...is null/...is not null
```

Ces tests retourneront toujours `true` ou `false` - sans tourner autour du pot. Exemples:

- `if (MyField is null) then...`
- `select * from Pupils where PhoneNumber is not null`
- `select * from Pupils where not (PhoneNumber is null)`  
/\* exemple identique au précédent \*/
- `update Numbers set Total = A + B + C where A + B + C is not null`

On peut dire que « `=` » (utilisé comme opérateur d'égalité) peut seulement comparer des valeurs, « `is` » teste un état.

## Affecter NULL à une variable ou un champ

On peut affecter NULL aux champs et variables en utilisant la même syntaxe que pour les valeurs :

- `insert into MyTable values (1, 'teststring', NULL, '8-May-2004')`
- `update MyTable set MyField = null where YourField = -1`
- `if (Number = 0) then MyVariable = null;`

- « Attendez une minute... vous avez dit que `MyField = NULL` n'était pas permis! »

C'est vrai... pour l'*opérateur de comparaison* « `=` » (au moins pour les versions pre-2.0 de Firebird). Mais ici nous parlons de « `=` » en tant qu'*opérateur d'affectation* . Malheureusement, les deux opérateurs utilisent le même symbole en SQL. Pour les affectations, faites soit avec « `=` » soit avec une liste d'insertion, vous pouvez traiter NULL comme n'importe quelle valeur - cela ne nécessite pas de syntaxe particulière (il n'en existe d'ailleurs pas).

## Travailler avec NULL

Cette section contient des trucs et astuces et exemples qui peuvent être utilisés dans votre travail avec NULLs.

### Tester NULL - si cela est nécessaire

La plupart du temps, vous n'avez pas à prendre de précaution particulière pour les champs ou variables qui peuvent être NULL. Par exemple, si vous faites ceci :

```
select * from Clients where Ville = 'Ralston'
```

vous ne voulez certainement pas voir les clients pour lesquels la ville n'a pas été spécifiée. De même :

```
if (Age >= 18) then CanVote = 'Yes'
```

n'inclue pas les gens dont l'âge est inconnu, ce qui semble correct. Mais :

```
if (Age >= 18) then CanVote = 'Yes';  
else CanVote = 'No';
```

semble moins vrai : si vous ne connaissez pas l'âge d'une personne, vous ne pouvez expressément lui refuser le droit de vote. Pire, cela :

```
if (Age < 18) then CanVote = 'No';  
else CanVote = 'Yes';
```

n'a pas les mêmes conséquences. Si certain des âges NULL sont en réalité inférieurs à 18, vous allez laisser des mineurs voter!

La bonne approche ici est de tester NULL expressément:

```
if (Age is null) then CanVote = 'Unsure';  
else  
  if (Age >= 18) then CanVote = 'Yes';  
  else CanVote = 'No';
```

#### Note

else se réfère toujours au dernier if dans le même bloc. Mais il est préférable souvent d'éviter les confusions en utilisant les mots clés begin...end autour des groupes de lignes. Je ne l'ai pas fait ici - je voulais écrire un faible nombre de lignes. Mais du coup j'ai compensé en ajoutant cette note ;-)

## Vérifier que des champs sont égaux

Quelques fois vous devez vérifier que deux champs ou variables sont égaux et vous voulez les considérer égaux s'ils sont tous deux NULL. Le test correct pour cela est :

```
if (A = B or A is null and B is null) then...
```

ou, si vous préférez :

```
if ((A = B) or (A is null and B is null)) then...
```

Attention tout de même: si seulement un des deux (A ou B) est NULL, l'expression de test devient NULL, pas false! C'est correct dans une instruction if, et nous pouvons même ajouter une clause else qui sera exécutée si A et B ne sont pas égaux (incluant le cas où un est NULL et l'autre ne l'est pas):

```
if (A = B or A is null and B is null)  
  then ...travail à faire si A égal B...  
  else ...travail à faire si A et B sont différents...
```

Mais n'ayez pas la brillante idée d'inverser l'expression et de l'utiliser comme un test d'inégalité (comme je l'ai déjà fait dans le passé):

```
/* Ne faites pas cela! */  
if (not(A = B or A is null and B is null))  
  then ...travail à faire si A différent de B...
```

Le code ci-dessus fonctionnera correctement si A et B sont tous deux NULL ou tous deux non-NULL. Mais la clause `then` ne s'exécutera pas si un des deux seulement est NULL.

Si vous voulez que quelque chose soit fait seulement si A et B sont différents, utilisez soit une des expressions correctes ci-dessus et mettez une expression muette dans la clause `then`, ou utilisez cette expression de test suivante :

```
/* Ceci est un test correct d'inégalité: */
if (A <> B
    or A is null and B is not null
    or A is not null and B is null) then...
```

## Vérifier qu'une valeur de champ a changée

Dans les triggers, il est souvent utile de savoir si une valeur de champ a changée (y compris: passer de NULL à non-NULL ou vice versa) ou est restée la même. Ce n'est rien d'autre qu'un cas particulier du test de l'(in)égalité de deux champs. Utilisez juste `New.FieldName` et `Old.FieldName` pour A et B:

```
if (New.Job = Old.Job or New.Job is null and Old.Job is null)
then ...le champ Job est resté le même...
else ...le champ Job a changé...
```

## Substituer une valeur à NULL

### La fonction COALESCE

Il existe une fonction dans Firebird 1.5 qui convertit NULL en quasiment tout ce que l'on veut. Cela permet de faire une conversion à la volée et utiliser le résultat dans le processus, sans utiliser la construction « `if (MyExpression is null) then` ». Cette fonction s'appelle `COALESCE` et s'utilise comme suit :

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

`COALESCE` retourne la première expression non-NULL dans la liste d'arguments. Si toutes les expressions sont NULL, elle renvoie NULL.

Voici comment utiliser `COALESCE` pour écrire le nom complet d'une personne avec ses prénom, surnom et nom, en supposant que certains surnoms sont NULL:

```
select Prenom
       || coalesce(' ' || Surnom, '')
       || ' ' || Nom
from Personnes
```

Ou bien encore en considérant que le surnom et le prénom peuvent être NULL:

```
select coalesce (Surnom, Prenom, 'Mr/Mme.')
       || ' ' || Nom
from AutresPersonnes
```

`COALESCE` ne vous aidera que dans les situations où NULL peut être traité de la même manière qu'une valeur permise pour le type de données. Si NULL a besoin d'un traitement particulier, comme dans l'exemple « droit de vote » utilisé précédemment, votre seule option est d'utiliser « `if (MonExpression is null) then` ».

## Firebird 1.0: les fonctions \*NVL

Firebird 1.0 ne connaît pas COALESCE. Toutefois, vous pouvez utiliser quatre UDFs qui procurent une bonne partie de ses fonctionnalités. Ces UDFs sont dans la bibliothèque `fbudf` et sont :

- `iNVL`, pour les integers
- `i64NVL`, pour les bigint
- `dNVL`, pour les double precision
- `sNVL`, pour les chaînes de caractères

Les fonctions \*NVL prennent deux arguments. Comme COALESCE, elles renvoient le premier argument s'il n'est pas NULL; sinon, elles renvoient le second. Notez que la bibliothèque de Firebird 1.0 `fbudf` - et par conséquent, les fonctions \*NVL - n'est disponible que pour Windows.

## Résumé

NULL en bref :

- NULL signifie *indéterminé*.
- Si NULL est présent dans une expression, le plus souvent l'expression entière devient NULL.
- Dans les fonctions d'agrégat seuls les champs non-NULL sont pris en compte. Exception: `COUNT(*)`.
- Quelques fois les UDF convertissent NULL <-> non-NULL d'une manière qui semble aléatoire.
- Si l'expression de test d'une instruction `if` est NULL, le bloc `then` est ignoré et le bloc `else` exécuté.
- Pour déterminer si A est NULL, utilisez « `A is (not) null` ».
- Les fonctions COALESCE et \*NVL peuvent convertir NULL en une valeur.
- L'assignation de NULL est comme assigner une valeur : avec « `A = NULL` » ou une liste d'insertion.

Souvenez-vous, c'est comme cela que NULL fonctionne *dans Firebird SQL*. Il peut y avoir des (subtiles) différences avec d'autres SGBDR.

## Historique du document

L'historique exact est enregistré dans le module manual de notre arbre CVS; voir [http://sourceforge.net/cvs/?group\\_id=9028](http://sourceforge.net/cvs/?group_id=9028)

### Historique des versions

|          |               |    |                                                                                                                                                                                     |
|----------|---------------|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0.1      | 8 Avril 2005  | PV | Première édition.                                                                                                                                                                   |
| 0.2      | 15 Avril 2005 | PV | Ajout de l'information que Fb 2.0 légalise les comparaisons « A = NULL ». Texte changé dans « Tester si quelque chose est NULL ». Légère modification de « Travailler avec NULLs ». |
| 0.2-fr   | 02 mai 2005   | PM | Traduit en français par Philippe Makowski.                                                                                                                                          |
| 0.2.1-fr | 04 mai 2005   | PM | Corrections de fautes d'orthographe et typographiques.                                                                                                                              |

## Licence

Le contenu de cette documentation est soumis à la « Licence » Public Documentation License Version 1.0 ; vous pouvez utiliser cette Documentation seulement si vous respectez les termes de cette Licence. Des copies de cette Licence sont disponibles à <http://www.firebirdsql.org/pdfmanual/pdl.pdf> (PDF) et <http://www.firebirdsql.org/manual/pdl.html> (HTML).

Le titre d'origine est : *Firebird Null Guide*.

Le rédacteur initial de la première version est : Paul Vinkenoog.

Copyright (C) 2005. Tous droits réservés. Contact: paulvink at users dot sourceforge dot net.

Traduction française par Philippe Makowski - voir [historique du document](#) - Copyright (C) 2005. Tous droits réservés. Contact: makowski at firebird-fr dot eu dot org.