NULL в СУБД Firebird

Подводные камни и поведение NULL в СУБД Firebird

Paul Vinkenoog

19 октября 2006 – Версия документа 0.2-ru

Перевод документа на русский язык: Сергей Ковалёв

Содержание

Введение

Время от времени в службу поддержки Firebird приходят вопросы, касающиеся «странных вещей», происходящих с NULL в СУБД Firebird. Концепция кажется сложной для понимания - возможно, частично это вызвано названием, которое наводит на мысль, что ничего «плохого» не произойдет, если вы прибавите это к числу или добавите его в конец строки. На самом деле, выполнение этих операций приведет к тому, что все выражение окажется равным NULL.

Эта статья исследует поведение NULL в СУБД Firebird, обозначая общие ошибки, и показывает вам, каким образом обезопасить выражения, которые содержат NULL или могут принимать значения NULL.

Если вам нужна просто короткая справка, чтобы освежить свою память, перейдите к заключению (которое, действительно, является очень кратким).

> **Замечание**
>
> Некоторые предложения и примеры в этом руководстве были позаимствованы из *Firebird Quick Start Guide*, которое впервые было опубликовано компанией IBPhoenix, теперь являющейся частью Firebird Project.

Что такое NULL?

В SQL NULL не является значением - это *состояние*, указывающее, что значение элемента неизвестно или не существует. Это не ноль, не пустота, не «пустая строка», и оно не ведет себя как какое-то из этих значений. Некоторые вопросы SQL являются более запутанными, чем NULL, и его работа станет не сложна для понимания, как только вы запомните следующее простое определение: NULL - значит *неизвестно*.

Позвольте мне повторить, что:

NULL означает НЕИЗВЕСТНО

Держите эту строку в уме во время чтения оставшейся части статьи, и большинство, на первый взгляд, нелогичных результатов, где вы получаете NULL, на практике объяснят сами себя.

NULL в выражениях

Как многие из нас узнали, к своему огорчению, NULL - заразен: используйте его в числовых, строковых выражениях или в выражениях, содержащих дату/время, и в результате вы всегда получите NULL; используйте его в логических (булевых) выражениях, и результат будет зави-

сеть от типа опрерации и других вовлеченных значений.

Пожалуйста обратите внимание, что в СУБД Firebird версий до 2.0 чаще всего недопустимо прямое использование константы NULL в операциях и сравнениях. Если вы видите NULL в приведенных ниже выражениях, понимайте это как «значение поля, переменной или другого выражения, результат вычисления которого есть NULL».

Выражения, возвращающие NULL

Выражения в этом списке *всегда* возвратят NULL:

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

Если вам трудно понять, почему, вспомните, что NULL - значит «неизвестно». Также давайте взглянем на следующую таблицу, где приведено объяснение каждого случая. В таблице мы не пишем NULL в выражении (как уже упоминалось, часто это недопустимо); вместо этого мы используем две сущности - A и B - которые обе являются NULL. A и B могут быть значениями полей, переменных или целых выражений со своими собственными правилами - но если только они являются NULL, они ведут себя одинаково в приведенных выражениях.

Таблица 1. Операции с NULL-сущностями A и B

Если A и B являются NULL, то:	Получаем:	Потому что:
`1 + 2 + 3 + A`	NULL	Если A незвестно, то 6 + A также неизвестно.
`'Home ' || 'sweet ' || A`	NULL	Если A неизвестно, 'Home sweet ' || A - неизвестно.
`MyField = A`	NULL	Если A неизвестно, вы не можете сказать, имеет ли MyField то же самое значение...
`MyField <> A`	NULL	...но вы так же не можете сказать, имеет ли MyField *отличающееся* значение!
`A = B`	NULL	Если A и B незивестны, невозможно знать равны ли они.
`not (A)`	NULL	Если A неизвестно, инвертированное значение также неизвестно.

NULL в логических выражениях

Мы уже рассмотрели, что `not (NULL)` дает в результате `NULL`. Для операторов `and` (логическое И) и `or` (логическое ИЛИ) взаимодействие несколько сложнее:

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

В СУБД Firebird не существует логического (булева) типа данных, тем не менее существуют константы `true` (истина) и `false` (ложь). В левой колонке таблицы с объяснениями, которую вы видите ниже, `(true)` и `(false)` представляют собой вложенные выражения, возвращающие `true`/`false`.

Таблица 2. Логические (булевые) операции с `NULL`-сущностью A

Если A является `NULL`, то:	Получаем:	Потому что:
`A or (false)`	`NULL`	«`A or false`» всегда имеет то же значение, что и A, которое неизвестно.
`A or (true)`	`true`	«`A or true`» всегда `true` - содержимое A не важно.
`A or A`	`NULL`	«`A or A`» всегда равно A, которое является `NULL`.
`A and (false)`	`false`	«`A and false`» всегда `false` - содержимое A не важно.
`A and (true)`	`NULL`	«`A and true`» всегда имеет то же значение, что и A, которое неизвестно.
`A and A`	`NULL`	«`A and A`» всегда равно A, которое является `NULL`.

Все эти результаты находятся в соответствии с булевой логикой. Факт состоит в том, что в порядке вычисления «`X or true`» и «`X and false`» вам просто нет *необходимости* знать значение X, что так же базируется на известной особенности, которую мы знаем в различных языках программирования под названием «сокращенное (ускоренное) вычисление булевых выражений».

Больше логики (или нет)

Полученные выше результаты сокращенного вычисления могут привести вас к следующим идеям:

- 0 умноженное на x равно 0 для любого x. Поэтому, даже если значение x неизвестно, `0 * x` равно 0. (Обратите внимание: это верно только если x имеет числовой тип данных, а не NaN или бесконечность.)

- Пустая строка располагается лексикографически перед любой другой строкой. Поэтому, `S >= ''` равно true не зависимо от значения S.

- Каждое значение равно самому себе, независимо от того, известно оно или нет. Таким образом, хотя `A = B` обоснованно вернет NULL, если A и B являются различными NULL-сущностями, `A = A` должно всегда возвращать `true`, даже если A является NULL.

Как это реализовано в СУБД Firebird? Что ж, мне очень жаль, но я должен сообщить вам, что, несмотря на такую неотразимую логику и аналогию с результатами булевых операций, описанных выше, следующие выражения всегда дают в итоге NULL:

- `0 * NULL`
- `NULL >= ''`
- `'' <= NULL`
- `A = A` (если A является полем или переменной со значением NULL)

Это сделано, чтобы быть последовательными.

NULL в агрегатных функциях

В агрегатных функциях, таких как COUNT, SUM, AVG, MAX и MIN, NULL обрабатывается отличным образом: для вычисления результата принимаются в рассмотрение только значения полей, не содержащие NULL. То есть, если у вас есть такая таблица:

MyTable

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

...выражение **`select sum(Amount) from MyTable`** вернет 54, что получается из 37 + 5 + 12. Если бы все пять значений полей были просуммированы, в результате должен был бы получиться NULL. Для AVG суммируются значения полей, не содержащие NULL, а сумма делится на количество таких значений.

Есть одно исключение из этого правила: `COUNT(*)` вернет количество всех записей, даже тех записей, значения всех полей которых являются NULL. Но COUNT(*ИмяПоля*) ведет себя как и остальные агрегатные функции, то есть считает записи, в которых значение указанного поля не является NULL.

Еще одно свойство достойно упоминания. `COUNT(*)` и COUNT(*ИмяПоля*) никогда не вернут

NULL: если нет записей в наборе данных, обе функции вернут 0. Также, COUNT(*ИмяПоля*) вернет 0, если все значения поля *ИмяПоля* в наборе данных являются NULL. Другие агрегатные функции вернут NULL в таком случае. Имейте в виду, что даже SUM вернет NULL, если использован пустой набор данных, что противоречит общей логике.

Обработка NULL в UDF

UDF (*User Defined Functions* - функции, определяемые пользователем) - это функции, которые не являются внутренними функциями ядра, они определены в отдельных (внешних) модулях. СУБД Firebird сопровождается двумя библиотеками UDF: ib_udf (унаследованная от СУБД InterBase) и fbudf. Вы можете добавить другие библиотеки, например купив их, скачав их из Интерента, или написав самостоятельно. UDF не могут использоваться «автоматически», как встроенные функции, - сначала их нужно «декларировать» в базе данных. Это так же верно и для UDF, которые поставляются в стандартных библиотеках с СУБД Firebird.

Преобразование NULL <-> не-NULL, о котором вы не просите

Рассказ о том, как объявлять, использовать и писать UDF выходит за пределы темы, рассматриваемой этим руководством. Однако, мы должны предупредить вас, что UDF порой выполняют неожиданное преобразование NULL. Иногда это приводит к тому, что входной NULL конвертируется в обыкновенное значение, а в других случаях происходит изменение верного входного значения, например, '' (пустая строка), на NULL.

Основной причиной этой проблемы является «старый стиль» вызова UDF, при котором невозможно передать NULL в качестве входного аргумента функции. Когда вызывается такая функция, как LTRIM (отсечение слева) с аргументом NULL, аргумент передается в функцию в виде пустой строки. Изнутри функции *нет способа* определить, является ли аргумент действительно пустой строкой или значением NULL. Что же в таком случае делать автору функции? У него есть выбор: либо брать аргумент по полученному значению, либо считать, что первоначально это был NULL и поступать с ним соответственно.

В зависимости от типа результата, вернуть NULL может быть возможным, даже если получить NULL невозможно. Таким образом, могут произойти следующие неожиданные вещи:

- Вы вызваете функцию с аргументом NULL. Он передается как значение, например, 0 или ''. Внутри функции этот аргумент не изменяется назад на NULL, и возвращается не-NULL результат.
- Вы вызываете функцию с верным аргументом, таким как 0 или ''. Он передается как есть (очевидно). Но код функции предполагает, что это значение на самом деле представляет собой NULL, и, ввиду отсутствия других ориентиров, трактует и возвращает его как NULL.

Оба преобразования являются не желательными, но второе, вероятно, более не желательно, чем первое (лучше предположить значение для NULL, чем утерять верное значение). Если вер-

нуться к нашему примеру с LTRIM: до версии Firebird 1.0.3 (включительно) эта функция возвращала NULL, если вы передавали ей пустую строку; а начиная с версии 1.5 она больше не возвращает NULL. В этих последних версиях строка NULL «обрезается» до пустой строки. Это не правильно, но здесь выбрано меньшее из двух зол: в более ранней версии верные пустые строки безжалостно превращались в NULL - строки с неизвестным значением.

Быть готовым к нежелательным преобразованиям

Нежелательные преобразования, описанные выше, обычно случаются только со старыми (давно существующими и не адаптировавшимися) библиотеками UDF, но их существует довольно много (чаще всего встречается в ib_udf). Также, ничего не предостерегает небрежного разработчика от подобных ошибок в функциях нового стиля. Итак, если вы используете UDF и не знаете, как функция ведет себя с аргументами, содержащими NULL, то:

1. Посмотрите на декларацию функции, чтобы увидеть, как передаются аргументы и возвращаются значения. Если там сказано «by descriptor», это должно быть безопасно (хотя осуществить дополнительную проверку никогда не помешает). Во всех остальных случаях обратитесь к остальным пунктам.

2. Если у вас есть исходный код и вы понимаете язык, на котором написана библиотека функций (например, C/C++), проверьте код функции.

3. Проверьте функцию на корректность ее работы с входным параметром NULL и с обычным входным параметром, который может быть интерпретирован функцией как NULL (0 для числовых аргументов и/или '' для строковых аргументов).

4. Если функция выполняет нежелательное преобразование NULL <-> не-NULL, вы должны учитывать это в вашем коде прежде, чем будете вызывать UDF (смотрите также Проверка на NULL где-то в этом руководстве).

Декларации функций для сопровождающих библиотек UDF могут быть найдены в подкаталоге Firebird bin/examples (v. 1.0) или bin/UDF (v. 1.5 и старше). Смотрите файлы с расширением .sql

Еще об UDF

Чтобы больше узнать об UDF, обратитесь к *InterBase 6.0 Developer's Guide* (скачать бесплатно на http://www.ibphoenix.com/downloads/60DevGuide.zip), *Using Firebird* и *Firebird Reference Guide* (оба на CD), или *Firebird Book*. CD и книга могут быть приобретены через http://www.ibphoenix.com/.

NULL в операторе if

Если выражение оператора `if` вычисляется как `NULL`, то предложение `then` пропускается и выполняется предложение `else` (если оно есть). Но берегитесь! В этом случае значение выражения может *вести себя*, подобно `false`, но это не значит, что *значение* равно `false`. Это все еще `NULL`, могут произойти непонятные вещи, если вы забудете об этом. Следующие примеры рассматривают несколько ужасающее поведение `NULL` в операторе `if`:

- ```
  if (a = b) then
    MyVariable = 'Equal';
  else
    MyVariable = 'Not equal';
  ```

  Если `a` и `b` оба `NULL`, `MyVariable` станет равно «`Not equal`» после выполнения этого кода. Причина состоит в том, что выражение «`a = b`» дает в итоге `NULL`, если хотя бы один из аргументов `NULL`. Если значение тестового выражения `NULL`, то блок `then` пропускается, а исполняется блок `else`.

- ```
  if (a <> b) then
    MyVariable = 'Not equal';
  else
    MyVariable = 'Equal';
  ```

  Здесь, `MyVariable` станет «`Equal`», если `a` является `NULL`, а `b` - нет, или наоборот. Объяснение аналогчично приведенному в предыдущем примере.

- ```
  if (not (a <> b)) then
    MyVariable = 'Equal';
  else
    MyVariable = 'Not equal';
  ```

  Это выглядит так, как будто здесь будет получен тот же самый результат, как и в предыдущем примере. Не так ли? После всего мы инвертируем тестовое выражение и меняем местами предложения `then` и `else`. И на самом деле, если ни одна из переменных не является `NULL`, оба фрагмента кода эквивалентны. Но как только `a` или `b` становятся `NULL`, то же получаем и для всего тестового выражения, и выполняется предложение `else`, а в результате получаем «`Not equal`».

  > **Замечание**
  >
  > Конечно мы в курсе, что третий пример полностью эквивалентен первому. Мы просто привели его, чтобы еще раз подчеркнуть, что `not(NULL)` является `NULL`. Таким образом, в случае, когда тестовое выражение вычисляется, как `NULL`, `not()` не инвертирует его.

Проверка на `NULL`

В свете того, какой беспорядок может навести `NULL`, вы часто захотите узнать, является ли что-либо `NULL` *прежде*, чем вы будете использовать это в выражении. Для некоторых очевидная
  ```

проверка должна выглядеть так:

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

И в самом деле, есть системы управления базами данных, которые поддерживают такой синтаксис проверки на NULL. Но стандарт SQL не допускает этого, не допускает этого и СУБД Firebird. В версиях до 2.0 такой синтаксис был даже недопустим. С версии 2.0 он разрешается, но сравнение всегда вернет NULL, независимо от состояния и значения A. Поэтому такой тест на NULL является бесполезным - нам нужен явный результат true или false.

Корректный способ проверки на NULL такой:

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

Эта проверка всегда вернет true или false - и никакой неразберихи. Примеры:

- ```
  if (MyField is null) then...
  ```
- ```
  select * from Pupils where PhoneNumber is not null
  ```
- ```
  select * from Pupils where not (PhoneNumber is null)
  /* делает то же самое, что и предыдущий пример */
  ```
- ```
  update Numbers set Total = A + B + C where A + B + C is not null
  ```

Можно сказать, что в то время как равенство «**=**» (когда используется как оператор равенства) может сравнивать значения, «**is**» проверяет состояние.

Установка значения поля или переменной в NULL

Поля и переменные могут быть установлены в NULL с помощью того же синтаксиса, что и для обычных значений:

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

- «Минуточку... но вы сказали, что MyField = NULL было недопустимо!»

Это верно... для *оператора сравнения* «=» (по крайней мере для СУБД Firebird до версии 2.0). Но здесь мы говорим о знаке «=», как об *операторе присваивания*. К сожалению, в SQL оба эти оператора имеют один и тот же символ. В случае присваивания, которое выполняется с помощью «=» или внутри списка вставки, вы можете трактовать NULL, как любое другое значение, -
  ```

специальный синтаксис не требуется.

Взаимодействие с NULL

Этот раздел содержит некоторые практические советы и примеры, которые могут быть использованы вами в вашей повседневной работе с NULL.

Проверка на NULL - если это имеет значение

Достаточно часто вам нет необходимости принимать специальные меры для полей или переменных, которые могут быть NULL. Например, если вы делаете так:

```
select * from Customers where Town = 'Ralston'
```

очевидно, что вы не хотите видеть клиентов, город которых не указан. Аналогично:

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

не будут включены люди с неизвестным возрастом, что так же верно и безопасно. Но:

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

выглядит менее обоснованным: если вы не знаете возраст человека, вы не можете явно лишить его права голоса. Хуже того:

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

не принесет ожидаемый результат, как и в предыдущем случае. Если кто-то с возрастом NULL имеет реальный возраст до 18, вы предоставите несовершеннолетнему право голоса!

Правильным подходом является явная проверка на NULL:

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

> **Замечание**
>
> else всегда ссылается на последний if в этом же блоке. Но чаще всего лучше предотвращать путаницу, помещая ключевые слова begin...end вокруг группы строк. Я не сделал этого здесь, чтобы сократить количество строк. И поэтому я был вынужден добавить это примечание. ;-)

Определение, одинаковы ли значения полей

Иногда вы хотите определить, являются ли значения двух полей или двух переменных одинаковыми, и вы будете считать их равными, если они оба NULL. Правильной проверкой для этого является следующая:

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

или, если вы хотите убрать возможное недопонимание:

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

Предупреждение. Если только одна из величин A и B является NULL, тестовое выражение станет NULL, а не false! Это нормально для оператора if, и мы даже можем добавить предложение else, которое будет выполнено, если A и B не равны (включая случай, когда одна из величин NULL, а другая - нет):

```
if (A = B or A is null and B is null)
  then ...код для выполнения, если A равно B...
  else ...код для выполнения, если A и B различаются...
```

Однако откажитесь от идеи инвертирования выражения и использования его как проверки на неэквивалентность (как я это однажды сделал):

```
/* Не делайте так! */
if (not(A = B or A is null and B is null))
  then ...код для выполнения, если A отличается от B...
```

Приведенный выше код работает корректно, если оба A и B являются NULL или оба не являются NULL. Но в нем не выполняется предложение then, если только одна из частей (A или B) является NULL.

Если вы хотите выполнять что-либо, когда A и B отличаются, вы должны либо использовать корректное выражение, приведенное выше, и поместить пустой оператор в предложение then, или использовать это более длинное выражение для проверки:

```
/* Это корректный тест на неэквивалентность: */
if (A <> B
    or A is null and B is not null
    or A is not null and B is null) then...
```

Определение изменения значения поля

В триггерах часто бывает полезным знать, что значение определенного поля изменилось (включая переход от NULL к не-NULL значению и наоборот) или осталось тем же самым. Это не что иное, как особый случай использования проверки на (не)эквивалентность значений двух

полей. Просто вместо A и B используйте New.ИмяПоля и Old.ИмяПоля:

```
if (New.Job = Old.Job or New.Job is null and Old.Job is null)
  then ...поле Job осталось тем же...
  else ...поле Job изменилось...
```

Замещение NULL значением

Функция COALESCE

В Firebird 1.5 есть функция, которая может конвертировать NULL во что-то еще. Это позволит вам выполнять преобразование «на лету» и использовать результат в последующей обработке без необходимости использования конструкции «if (MyExpression is null) then». Функция называется COALESCE, и вы можете вызвать ее так:

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

COALESCE возвращает первое не-NULL выражение из списка аргументов. Если все выражения являются NULL, она вернет NULL.

Вот как вы можете сконструировать полное имя человека с помощью COALESCE из первого, среднего и последнего имени, предполагая, что среднее имя может иметь значение NULL:

```
select FirstName
       || coalesce(' ' || MiddleName, '')
       || ' ' || Lastname
from Persons
```

Или для создания наиболее информативного имени, насколько это возможно, из таблицы, которая содержит псевдонимы, и в предположении, что и псевдоним и первое имя могут быть NULL:

```
select coalesce (Nickname, FirstName, 'Mr./Mrs.')
       || ' ' || Lastname
from OtherPersons
```

COALESCE поможет вам только в тех ситуациях, когда NULL можно трактовать одинаково, как некоторое допустимое значение для типа данных. Если для NULL необходима специальная обработка, как в примере с «правом голоса», показанном ранее, вы можете использовать только выражение «if (MyExpression is null) then».

Firebird 1.0: функции *NVL

В СУБД Firebird 1.0 не существует функции COALESCE. Однако, вы можете использовать четыре UDF, которые предоставляют бОльшую часть функциональности функции COALESCE. Эти UDF расположены в библиотеке fbudf и называются

- iNVL для целочисленных аргументво
- i64NVL для агрументов типа bigint
- dNVL для агрументов типа double precision
- sNVL для строк

Функции *NVL получают два аргумента. По аналогии с COALESCE, они возвращают первый аргумент, если он не является NULL; в противном случае они возвращают второй аргумент. Пожалуйста, обратите внимание, что это библиотека для Firebird 1.0 fbudf - и поэтому функции *NVL доступны только для Windows.

Заключение

NULL в двух словах:

- NULL - значит *неизвестно*.
- Если NULL фигурирует в выражении, чаще всего все выражение даст в результате NULL.
- В агрегатных функциях только не-NULL значения полей используются при вычислении. Исключение - функция COUNT(*).
- UDF иногда конвертируют NULL <-> не-NULL как бы случайным образом.
- Если выражение оператора if вычисляется как NULL, то блок then пропускается, а блок else выполняется.
- Чтобы проверить, что A является NULL, используйте «A is (not) null».
- Функции COALESCE (1.5) и *NVL (1.0) могут конвертировать NULL в значение.
- Присваивание NULL выполняется так же, как и присваивание значений - с помощью «A = NULL» или в списке вставки.

Помните, так работает NULL в *СУБД Firebird*. Могут быть (трудно уловимые) отличия в сравнении с другими СУБД.

История документа

Точный файл истории записан в модуле `manual` в дереве CVS; смотрите http://sourceforge.net/cvs/?group_id=9028

История переиздания

0.1	8 апр 2005	PV	Первая редакция.
0.2	15 апр 2005	PV	Упоминается, что Fb 2.0 допускает сравнение «`A = NULL`». Изменен текст в «Проверка на `NULL`». Немного изменен раздел «Взаимодействие с `NULL`».
0.2-ru	25 сен 2006	SK	Документ переведен на русский язык.
0.2-ru	19 окт 2006	PM	Корректировка перевода на русский язык.

Условия лицензии

Оригинальное название документа: *Firebird Null Guide*.

Автор исходного документа: Paul Vinkenoog.

Перевод на русский язык: Сергей Ковалёв.

Корректор перевода: Павел Меньщиков.