DOI: 10.15514/ISPRAS-2023-35(2)-6

Типизированные неизвестные значения: шаг к решению проблемы представления отсутствующей информации в реляционных базах данных

С.Д. Кузнецов, ORCID: 0000-0002-8257-028X <kuzloc@ispras.ru>

Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
Московский государственный университет им. М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1
Московский физико-технический институт,
141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
НИУ «Высшая школа экономики»,
101978, Россия, Москва, ул. Мясницкая, д. 20

Аннотация. Состояние дел области управления отсутствующей информацией в реляционных базах данных оставляет желать лучшего. В стандарте SQL для представления отсутствующих данных используется универсальное null-значение, а управление основано на трехзначной логике, в которой null-значение отождествляется с третьим логическим значением. Это решение концептуально противоречиво и часто приводит к интуитивно непонятному поведению СУБД. Альтернативный подход с использованием типизированных специальных значений перекладывает всю обработку отсутствующих данных на пользователей. В этой статье мы анализируем многолетнюю историю исследований и разработок, которая привела к такой ситуации. Мы приходим к выводу, что в стандарте SQL и не могло появиться другое решение из-за выбора более 50 лет тому назад механизма универсального null-значения, а альтернативный механизм не может обеспечить системную поддержку специальных значений из-за использования двухзначной логики. Мы предлагаем комбинированный подход с использованием типизированных специальных значений на основе трехзначной логики. Этот подход позволяет использовать семантику типов данных при обработке запросов с условиями, включающими неизвестные данные. Кроме того, наш подход позволяет определить полноценную трехзначную логику, в которой специальное значение булевского типа является третьим логическим значением.

Ключевые слова: реляционные базы данных; отсутствующая информация; null-значение; трехзначная логика; типизированные специальные значения

Для цитирования: Кузнецов С.Д. Типизированные неизвестные значения: шаг к решению проблемы представления отсутствующей информации в реляционных базах данных. Труды ИСП РАН, том 35, вып. 2, 2023 г., стр. 73-100. DOI: 10.15514/ISPRAS–2023–35(2)–6

Typed unknown values: a step towards solving the problem of representing missing information in relational databases

S.D. Kuznetsov, ORCID: 0000-0002-8257-028X <kuzloc@ispras.ru>

Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia
Moscow Institute of Physics and Technology (State University),
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
National Research University Higher School of Economics
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia

Abstract. The state of affairs in the area of missing information management in relational databases leaves much to be desired. The SQL standard uses the universal null value to represent missing data, and the control is based on three-valued logic, in which the null value is identified with a third boolean value. This solution is conceptually inconsistent and often results in DBMS behavior that is not intuitive. An alternative approach using typed special values leaves all handling of missing data to users. In this article, we analyze the long history of research and development that led to this situation. We come to the conclusion that no other solution could have appeared in the SQL standard due to the choice of the mechanism of the universal null value more than 50 years ago, and the alternative mechanism cannot provide system support for special values due to the use of two-valued logic. We propose a combined approach using typed special values based on three-valued logic. This approach allows you to use the semantics of data types when processing queries with conditions that include unknown data. In addition, our approach allows us to define a full-fledged three-valued logic in which a special value of a Boolean type is the third boolean value.

Keywords: relational databases; missing information; null value; three-valued logic; typed special values

For citation: Kuznetsov S.D. Typed unknown values: a step towards solving the problem of representing missing information in relational databases. Trudy ISP RAN/Proc. ISP RAS, vol. 35, issue 2, 2023. pp. 73-100 (in Russian). DOI: 10.15514/ISPRAS-2023-35(2)-6

1. Введение

Проблема отсутствующей информации появилась одновременно с первыми попытками реализовать реляционную модель Эдгара Кодда. В первых своих статьях [1, 2], описывающих реляционную модель данных, Кодд вовсе не говорил о существовании этой проблемы. Похоже, что первый раз о потребности сохранять в реляционных баз данных отношения, в которых не во всех атрибутах кортежей содержатся реальные данные (потому что они неизвестны) Кодд написал в [3]. По-видимому, не просто так заметка [3] появилась в том же выпуске Бюллетеня ACM-SIGMOD за 1975-й год, что и отчет о панельной дискуссии, которая была проведена в том же году на Национальной компьютерной конференции [4]. Похоже, что на Кодда сильно подействовали соображения участников проектов по реализации реляционных систем баз данных, на что, в частности указывают слова благодарности в [3] в адрес участников проекта System R. Координированность работ Кодда и проекта System R подтверждается и тем фактом, что уже в 1976-м году была опубликована первая развернутая статья про язык баз данных SEQUEL 2 [5], в которой, по сути, повторялись мысли Кодда, зафиксированные в [3].

Еще через три года Кодд опубликовал свою знаменитую статью [6], в которой, в частности, определил реляционную алгебру в расчете на тот случай, когда в отношениях базы данных могут содержаться неопределенные значения. Можно считать, что с этого времени в языке SQL были закреплены механизм неопределенных значений и трехзначная логика, хотя, по-видимому, многим авторитетным специалистам этот подход не нравился, потому что в стандарте SQL он был в некотором виде зафиксирован только в 1999 г. [7].

Похоже, что и сам Кодд был не слишком удовлетворен средствами поддержки работы с отсутствующей информацией, которые вошли в SQL при его благословлении. В конце 1980-х он опубликовал книгу [8], представляющую новый вариант реляционной модели данных, в котором были исправлены некоторые недостатки первый версии модели, попавшие в SQL. Исправления коснулись и неопределенных значений: были введены два вида маркеров, один из которых использовался для представления неизвестных значений, а другой – для обозначения того, что для конкретного кортежа значение некоторого атрибута определено быть не может. Соответственно, для корректного использования этого подхода потребовалась уже четырехзначная логика. В сообществе SQL этот подход востребован не был.

Многим людям не нравился (и не нравится) механизм неопределенных значений и трехзначной логики, предложенный Коддом во второй половине 1970-х и поддерживаемый в SQL. Но наиболее активным и последовательным противником этого механизма всегда являлся ученик Кодда, последователь исходных идей реляционной модели данных, всемирно известный писатель и лектор Кристофер Дейт. Дейт критикует SQL вообще и средства поддержки неопределенных значений, включая трехзначную логику в частности, во множестве статей и большинстве книг. В этой статье мы не будем обсуждать эту критику. Заметим лишь, что она всегда очень подробна и убедительна.

Однако здесь для нас более важно то, что с середины 1980-х Дейт и его многолетний соратник и соавтор Хью Дарвен пытались (и до недавнего времени продолжали пытаться) найти решение проблемы отсутствующей информации в базах данных, не используя понятие нетипизированного неопределенного значения и не прибегая к трехзначной логике. Первый подход [9, 10], которого они в течение долгого времени придерживались, основывается на понятии специальных значений, расширяющих любой тип данных и приводящих к получения булевского значения *true* только при сравнении на равенство специального значения с нем же самим. Однако, как указывали критики этого подхода, он не работает в важных на практике случаях, когда на множестве значений базового типа, расширяемого специальным значениям, имеется отношение порядка. И хотя Дейт и Дарвен продолжали придерживаться подхода со специальными значениями до середины первого десятилетия 2000-х, позже они отказались от специальных значений и занялись поиском других решений.

Мы кратко обсудим эти решения в основной части статьи, но сейчас отметим, что несмотря на отсутствие в них видимых ошибок, ни одно из них не годится для практического использования. Ни одно из них даже не включено в последнее издание книги Дейта и Дарвена, в которой они излагают свое видение современной реляционной модели данных [11].

В результате многолетних размышлений у нас сложилось мнение, что

a) неопределенные значения и трехзначная логика, принятые в стандарте SQL:1999 [7], никуда не годятся из-за того, что NULL не типизирован, а без этого трехзначная логика становится, мягко говоря, неполноценной;
b) специальные значения Дейта и Дарвена [9, 10] (почти) никуда не годятся, поскольку их невозможно использовать без поддержки трехзначной логики;
c) значит, стоит попробовать скрестить специальные значения с трехзначной логикой, что
d) не решает проблему в целом, но выглядит приличнее, чем подход стандарта SQL и подход специальных значений.

Во втором разделе оставшейся части статьи сравнительно подробно описываются наиболее важные вехи истории проблемы отсутствующей информации. В третьем разделе мотивируется и обсуждается предлагаемое решение. Наконец, четвертый раздел содержит заключение с обсуждением ограничений и недостатков описанного подхода.

2. Исторический контекст

За десятилетия, прошедшие после введения Эдгаром Коддом реляционной модели данных [1, 2], была опубликована масса статей, так или иначе касающихся проблемы отсутствующих данных в реляционных базах данных. В этой статье мы не будем пытаться привести какой-либо обзор результатов во всех направлениях исследований, касающихся этой проблемы. Нас здесь интересует один аспект: представление в атрибутах кортежей отношений (или столбцах строк таблиц) отсутствующей информации и механизм манипулирования такими отношениями (или таблицами). Однако и этому аспекту было посвящено много публикаций чисто теоретического характера, которые (по нашему мнению) не оказали заметного влияния не технологию реляционных систем управления данными. Соответствующие результаты также не будут рассматриваться в этой статье.

В первом подразделе этого раздела будут обсуждаться основные вехи, которые привели к механизму работы с отсутствующими данными, зафиксированному в стандарте SQL:1999 [7]. Здесь, конечно, на первом месте стоят работы Кодда. Второй подраздел посвящен попыткам обойтись в реляционной модели данных без null-значения и трехзначной логики и опирается, главным образом, на работы Дейта и Дарвена. Основная цель раздела, кроме представления обзора работ, состоит в том, чтобы показать недостатки имеющихся подходов и мотивировать попытку автора данной статьи улучшить текущее состояние дел.

2.1 Путь к SQL:1999

2.1.1 Реляционная модель данных: версия 1

Как отмечалось в начале статьи, этот путь начался с публикации Коддом в 1975 г. небольшой заметки [3]. Основные результаты этой заметки (относящиеся к рассматриваемой здесь проблеме) состоят в следующем.

1) Предлагается решение только одной части проблемы – работа с отсутствующими данными без уточнений сути других частей. Для этого вводится понятие *null-значения* (*null value*)[1], обозначаемого в [3] символом @.
2) Арифметические операции сложения, вычитания, умножения и деления возвращают null-значение, если хотя бы операнд является null-значением.
3) Поскольку результатом сравнения $x = y$, в котором неизвестны какой-либо один или оба аргумента, не является ни *true*, ни *false*, вводится неизвестное логическое значение, что делает используемую логику трехзначной. Для обозначения неизвестного логического значения используется тот же символ @, потому что логические значения могут сохраняться в базе данных, и желательно обрабатывать все неизвестные или неопределенные значения единообразно[2]. Предлагаемая трехзначная логика определяется следующими таблицами истинности.

AND	*false*	@	*true*
false	*false*	*false*	*false*
@	*false*	@	@
true	*false*	@	*true*

OR	*false*	@	*true*
false	*false*	@	*true*
@	@	@	*true*
true	*true*	*true*	*true*

NOT(*false*) = *true*; NOT(@) = @; NOT(*true*) = *false*.

[1] Следуя традиции, мы используем в своей статье этот термин, хотя, вообще говоря, он некорректен. Null-значение – это не значение, потому что любое значение должно принадлежать хотя бы одному типу данных, а null-значение нетипизировано. Мы предлагаем считать этот термин просто сокращенной формой от *обозначения отсутствия значения*.

[2] Здесь Кодд говорит об этом мимоходом, но на самом деле предложенный подход к пониманию сути третьего логического значения имел далеко идущие последствия, о которых мы поговорим позже в этом разделе.

4) Для определения значения логического выражения предлагается следующий *принцип замены неопределенных значений*. Значением логического выражения является @ тогда и только тогда, когда одновременно выполняются два следующие условия:
   a. можно заменить каждое вхождение @ некоторым допустимым реальным значением (возможно, разными значениями для разных вхождений @) таким образом, что значением полученного выражения является *true*;
   b. можно заменить каждое вхождение @ некоторым допустимым реальным значением (возможно, разными значениями для разных вхождений @) таким образом, что значением полученного выражения является *false*.
5) В соответствии с принципом замены неопределенных значений значение @ вырабатывается любым сравнением $x\ \theta\ y$, где $\theta - <, \leq, >, \geq$, если один из аргументов или оба являются @.
6) Для каждого натурального числа *n* допустимым является кортеж, состоящий из *n* неопределенных значений. Однако никакое *n*-арное отношение не может содержать более одного такого кортежа. Обычным правилом является то, что никакие отношения не могут содержать кортежи-дубликаты (в некоторых реляционных системах это правило строго не соблюдается, и для этих систем описываемая схема нуждается в модификации). В соответствии с правилом недопущения дубликатов, неопределенное значение в одном кортеже считается таким же, как неопределенное значение в другом кортеже[3]. Это отождествление одного неопределенного значения с другим может казаться противоречащим нашему назначению истинностного значения для сравнения @ по равенству. Однако выявление кортежей для удаления дубликатов не следует путать со сравнением по равенству при вычислении условий поиска.
7) Кроме того, в статье приводится набросок того, как должны выглядеть реляционная алгебра и реляционное исчисление при наличии неопределенных значений.

Уже на следующий год участниками проекта System R была опубликована статья [5], в которой, в частности, кратко описывается внедрение идей Кодда [3] с очень незначительными (но важными) изменениями в прототип языка SQL – SEQUEL 2. Основные черты языка (и его реализации), связанные с поддержкой неопределенных значений, можно охарактеризовать следующим образом.

1) Допускается существование в базе данных неизвестных, или null-значений. На null-значение можно сослаться с помощью ключевого слова NULL.
2) При вычислении всех агрегатных функций (в System R они назывались встроенными, *built-in*), кроме COUNT), null-значения игнорируются.
3) При определении того, удовлетворяет ли данный кортеж условию раздела WHERE запроса, предикатам, в которых участвуют атрибуты, содержащие в соответствующем кортеже null-значение, присваивается неизвестное истинностное значение (обозначается знаком вопроса ?)[4].
4) Для вычисления всего условия раздела WHERE используется интерпретация логических операций AND, OR и NOT в трехзначной логике на основе тех же таблице истинности, что в [3].
5) Кортеж, для которого вычисляется условие раздела WHERE запроса, удовлетворяет этому условию в том и только в том случае, когда оно вычисляется в *true*.
6) Условия, задаваемые в ограничениях целостности (*assertion* в терминологии System R), также вычисляются в трехзначной логике и считаются удовлетворенными тогда и только тогда, когда результатом этого вычисления не является *false*.

[3] В одном и том же атрибуте.

[4] В этом состоит существенное отличие от [3], где Кодд используется одно и то же обозначение как для неизвестного значения вообще (не принадлежащего никакому типу данных), так и для третьего логического значения, которое должно было бы принадлежать булевскому (трехзначному) типу.

Больше в статьях, посвященных System R, проблематика отсутствующих значений и трехзначной логики не затрагивалась. Как мы увидим дальше, в контексте SQL инициатива (достаточно вялая) перешла к стандартизаторам этого языка.

В 1979 г. Кодд опубликовал статью [6], которая, как подчеркивает ее название, была главным образом направлена на расширение реляционной модели данных с целью обеспечения более высокого семантического уровня данных, хранимых в реляционных базах данных. Однако в [6] Кодд затрагивает и проблему отсутствующей информации, но при этом лишь дословно повторяет то, о чем говорилось в [3].

2.1.2 Реляционная модель данных: версия 2

Следующий раз Эдгар Кодд обратился к проблеме отсутствующей информации в 1986 г. в [12]. Кратко говоря, в этой статье он впервые предлагает расширение реляционной модели данных, позволяющее работать в двумя видами отсутствующих данных – неизвестными, но возможными (*missing and applicable, A*) и недопустимыми (*missing and inapplicable, I*) данными. Вот суть новых предложений.

1) Отсутствующие A- и I-данные обозначаются A- и I-маркерами (*mark*) соответственно.
2) Семантика того факта, что некоторое значение в базе данных отсутствует, отлична от семантики значения, хранимого в базе данных. Первый факт применим к любому значению, независимо от его типа. Семантика реального значения сильно зависит от его домена (или типа данных), из которого атрибут отношения получает свои значения.
3) Если x обозначает некоторое реальное значение, хранимое в базе данных, A – A-маркер и I – I-маркер, то

$$\begin{array}{lll} x + x = 2x & x + A = A & A + x = A \\ A + A = A & A + I = I & I + A = I \\ I + I = I & x + I = I & I + x = I \end{array}$$

   Аналогичные таблицы справедливы и для арифметических операций вычитания, умножения и деления, а также для операции конкатенации символьных строк.[5]
4) Введение второго неизвестного значения наводит на мысль об уместности применения четырехзначной логики, однако Кодд считает это *несрочным делом, которое можно отложить, пока у практиков не появится соответствующая потребность*. Поэтому в [12] сохраняется трехзначная логика. Это обосновывается следующим образом. *Сравнение по равенству двух маркеров недопустимых значений имеет смысл обрабатывать так же, как и сравнение двух фактических значений. Однако любое вхождение I-маркера может быть обновлено специально уполномоченным пользователем, преобразуясь в A-маркер, а затем, возможно, в фактическое значение*.
   Поэтому можно считать, что в операции реляционного языка всякий раз, когда A- или I-маркер приравнивается в условии к фактическому значению или к A-маркеру, или к I-маркеру истинностным значением такого условия всегда считается MAYBE (так в [12] называется третье истинностное значение). Аналогично, полагается, что результатом любого сравнения, в котором A- или I-маркер сравнивается с фактическим значением или A-маркером, или к I-маркером, является MAYBE.
   Правила построения сложных условий на основе простых сравнений определяются такими же таблицами истинности булевских операций дизъюнкции, конъюнкции и отрицания, как и в [3, 6], но теперь операндами логических операций являются

[5] Удивительно, что такие таблицы появились у Кодда только спустя десять нет после введения им неопределенных значений.

высказывания P и Q, каждое из которых может иметь значение *true* (*t*), *maybe* (*m*) или *false* (*f*)[6].

P	not P
t	f
m	m
f	t

P or Q		Q		
		t	m	f
P	t	t	t	t
	m	t	m	m
	f	f	m	f

P & Q		Q		
		t	m	f
P	t	t	m	f
	m	m	m	f
	f	f	f	f

В 1987 г. Кодд опубликовал статью [13], в которой содержатся разъяснения идей, высказанных в [12], а также несколько изменений, которые сам Кодд считает незначительными, однако нам они представляются очень существенными.

1) Кодд все-таки вводит четырехзначную логику. Как и в [12], отсутствующие A- и I-данные обозначаются A- и I-маркерами соответственно. Пусть имеется скалярная функций с одним или несколькими аргументами. Считается, что I-маркер сильнее A-маркера. В общем случае, если самым сильным маркером аргументов вызова является I-маркер, то результат вызова скалярной функции – I-маркер. Если, с другой стороны, самым сильным маркером является A-маркер, то результат – A-маркер.
   Например, арифметические выражения вычисляются по следующим правилам (символ % обозначает любую арифметическую операцию, z – обычное значение, a – A-маркер, i -- I-маркер):
   $z \% a = a$ $z \% i = i$ $a \% z = a$ $i \% z = i$
   $a \% a = a$ $a \% i = i$ $i \% a = i$ $i \% i = i$
2) Булевский тип (домен) расширяется двумя дополнительными значениями a-MAYBE и i-MAYBE. Кодд обозначает их так же, как и A-маркер и I-маркер, -- просто символами a и i соответственно, но подчеркивает, что t (*true*), f (*false*), a и i – это реальные значения булевского типа, а не маркеры отсутствующих данных. Ни в [13], ни в [8] явно не говорится, как вычисляются сравнения скалярных значений, но если следовать общему правилу, сформулированному в предыдущем разделе, то должна быть справедлива сконструированная нами таблица (Кодд ничего подобного не приводит и вообще не говорит о сравнениях в четырехзначной логике). В этой таблице по необходимости мы используем разные обозначения для маркеров отсутствующих данных (a и i) и для соответствующих булевских значений (a_B и i_B). Символ Θ обозначает произвольную операцию сравнения скалярных значений ($=, \neq, <, \leq, >, \geq$), z – обычное значение.
   $z \Theta a = a_B$ $z \Theta i = i_B$ $a \Theta z = a_B$ $i \Theta z = i_B$
   $a \Theta a = a_B$ $a \Theta i = i_B$ $i \Theta a = i_B$ $i \Theta i = i_B$
3) В соответствии с тем же правилом строятся таблицы истинности в четырехзначной логике для операций NOT, OR и AND. Ниже приводятся таблицы из [13], в которых для обозначения истинностных значений используются символы t, f, a и i.

P	not P
t	f
a	a
i	i
f	t

P or Q		Q			
		t	a	i	f
P	t	t	t	t	t
	a	t	a	a	a
	i	t	a	i	f
	f	t	a	f	f

P & Q		Q			
		t	a	i	f
P	t	t	a	i	f
	a	a	a	i	f
	i	i	i	i	f
	f	f	f	f	f

4) В [13] Кодд считает, что дополнительная сложность четырехзначной логики в настоящее время не оправдана, тем более что проектировщикам и пользователям СУБД не очень просто жить и с трехзначной логикой. Поэтому пока уместно продолжать встраивать в СУБД трехзначную логику. Однако внешние спецификации СУБД должны позволять переход с трехзначной на четырехзначную логику без ущерба для пользователей ~ без переписывания приложений (или с минимальным переписыванием). Если в СУБД встраивается четырехзначная логика, то либо она должен согласовываться подходом, описанным выше, либо отклонения от этого подхода должны быть хорошо обоснованы.[7]

[6] То есть здесь устранена путаница между третьим логическим значением и обозначением отсутствующих данных.

В начале 1990-го г. Кодд опубликовал книгу [8], в которой систематически и подробно описал вторую версию реляционной модели данных (RM/V2), считая первой версией (RM/V1) ту, которая была введена в начале 1970-х [1, 2]. Расширения реляционной модели (RM/T), введенные в [6], в RM/V2 не вошли, но Кодд планировал добавить их в следующих версиях реляционной модели данных. К сожалению, следующие версии реляционной модели так и не появились, и похоже, что работы Кодда, приведшие к написанию [8], остались недооцененными сообществом баз данных.

Что касается средств управления отсутствующей информацией, описываемых в [8], они в точности совпадают с теми, которые были введены в [12, 13]. Единственным существенным дополнением является то, что в RM/V2 строго требуется поддержка четырехзначной логики. Больше Кодд к этой теме не обращался, да и вообще не публиковал работ про реляционную модель данных.

2.1.3 Первые стандарты языка SQL

Тем временем, в 1986 г. Американским национальным институтом стандартов (ANSI) был принят первый национальный стандарт языка SQL (SQL-86) [14], который через год был одобрен Международной организацией по стандартизации (ISO) в качестве международного стандарта [15]. По поводу отсутствующих данных в стандарте практически полностью повторяются соображения разработчиков System R, приведенные в [5] (в еще более расплывчатом виде). Основные соответствующие спецификации выглядят так.

1) Тип данных – это множество представимых значений. Логическим представлением значения является литерал. Физическое представление значения определяется в реализации. Значение является примитивным в том смысле, что в рамках данного стандарта оно не подразделяется на более мелкие части. Значение является либо null-значением, либо не-null-значением. Null-значение – это определяемое в реализации зависящее от типа специальное значение, отличное от всех не-null-значений этого типа. Не-null-значение может быть либо строкой символов, либо числом.
2) Если в арифметическом выражении встречается хотя бы одно null-значение, то значением всего выражения является null-значение. Кроме арифметических выражений над данными какого-либо числового типа, в SQL-86 поддерживаются выражения над строками символов, но в них не может быть операций, так что это вырожденные выражения, всего лишь специфицирующие символьные строки.
3) Если в предикате сравнения x comp-op y (comp-op = | <> | < | > | <= | >=, x – выражение, y – выражение или подзапрос, выдающий не более одного значения) x и y являются выражениями, значением хотя бы одного из которых является null-значение, или если y – подзапрос, выдающий пустое множество результатов, то значением сравнения x comp-op y является *unknown*.

[7] На самом деле, подход Кодда с четырехзначной логикой ненамного лучше того, который в настоящее время принят в SQL (об этом см. ниже в данном подразделе). Булевский тип выглядит неполноценным, потому что либо нужно прибегнуть в тем же ухищрениям, что и в SQL (запретить хранить в базах данных булевские значения, запретить прямо использовать в логических выражениях литералы булевского типа и т.д.), либо мы столкнемся, например, с потребностью представлять в базе данных отсутствующие значения булевского типа и, соответственно, каким-то образом вычислять логические выражения, в которых присутствуют маркеры отсутствия данных.

4) Для проверки того, что в указанном столбце текущей строки находится null-значение, в стандарте определен предикат IS NULL, выдающий значения *true* или *false.*
5) Условие выборки (*search condition*) – это, по сути, логическое выражение, строящееся на основе предикатов с использованием логических операций NOT, AND и OR с естественными приоритетами операций и возможностью расстановки скобок. Результатом условия выборки для текущей строки может быть *true*, *false* и *unknown* на основе интерпретации логического выражения в трехзначной логике с теми же таблицами истинности, что и в [5], с заменой обозначения неизвестного истинностного значения ? на *unknown*.
6) Результатом выполнения раздела WHERE является таблица, включающая те строки таблицы, полученной после обработки раздела FROM, для которых значением условия выборки является *true*.

Вдогонку к SQL-86, в 1989 г. был принят следующий международный стандарт SQL-89 [16]. Как следует из его полного названия, основные расширения языка касались средств поддержки целостности баз данных. В стандарте SQL-86 обеспечивались всего два вида определений ограничений целостности: NOT NULL на уровне определения столбца таблицы (запрет наличия в этом столбце любой строки соответствующей таблицы null-значения) и UNIQUE на уровне определения столбца или вне таких определений с возможностью определить возможный ключ, состоящий из нескольких столбцов (ни в одном столбце любого возможного ключа не допускается наличие null-значения). Эти возможности сохранились в SQL-89, но появились возможности определения в операции CREATE TABLE ограничений внешнего ключа, проверочных ограничений и, кроме того, определения первичного ключа таблицы.

1) Ссылочное ограничение внешнего ключа можно определить на уровне определения столбца (REFERENCES), если внешним ключом является соответствующий столбец, или в виде отдельного табличного ограничения (FOREGN KEY), допускающего внешние ключи из нескольких явно указываемых столбцов. В обоих случаях указывается имя таблицы, на строки которых будут указывать значения внешнего ключа, а также, возможно, список столбцов этой таблицы, составляющих какой-либо ее возможный ключ. В определении внешнего ключа допускается отсутствие явного указания этого списка, и тогда по умолчанию считается, что внешнему ключу соответствует первичный ключ соответствующей таблицы, который в этом случае должен быть в ней явно объявлен.
   Это единственный повод для введения SQL-89 возможности указания в определении возможного ключа наряду со спецификаций UNIQUE возможности указания PRIMARY KEY (не более чем для одного возможного ключа). Во всем остальном первичный ключ обладает ровно теми же свойствами, что и любой возможный ключ.
2) Концептуально любое ссылочное ограничение таблицы (как и любое проверочное ограничение, см. ниже) проверяется при выполнении любой операции обновления базы данных и считается удовлетворенным, если каждое значение внешнего ключа (возможно, составного), содержащееся в строках ссылающейся таблицы, ни один компонент которого не является null-значением, совпадает со значением соответствующего возможного ключа в какой-либо строке таблицы, на которую указывает ссылка, либо хотя бы один компонент значения внешнего ключа является null-значением (это соответствует варианту MATCH SIMPLE определения внешнего ключа в современных вариантах стандарта SQL).
3) Наконец, проверочное ограничение CHECK, которое также можно определить как на уровне определения столбца, так и в виде отдельного табличного ограничения, содержит условие выборки, которое должно вычисляться для каждой отдельной строки соответствующей таблицы и считается удовлетворенным, если ни для какой строки не принимает истинностного значения *false*.

В 1992 г. был принят следующий стандарт SQL-92. По сравнению с двумя первыми стандартами SQL-92 был очень значительно расширен, о чем можно судить даже просто по его объему в более чем 600 стр. против 120 стр. в SQL-89. К сожалению, средства поддержки отсутствующих данных остались на очень посредственном уровне. Перечислим части стандарта, затрагивающие интересующую нас тему.

1) Определение: null-значение (null) – это специальное значение, или маркер, используемый для обозначения отсутствия какого-либо типизированного значения.
2) В разделе «Типы данных»: тип данных – это множество представимых значений. Логическим представлением значения является литерал. … Значение является null-значением или не-null-значением.
   Null-значение – это зависящее от реализации специальное значение, отличное от любого не-null-значения типа данных, отсутствие значений которого оно обозначает. В действительности имеется только одно null-значение, и оно является элементом каждого типа данных SQL. Для null-значения отсутствует литерал, хотя в некоторых местах используется ключевое слово NULL для обозначения потребности в null-значении.[8]
3) В SQL-92 допускается использование значительно большего числа типов данных, чем в SQL-89: CHARACTER, CHARACTER VARYING, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, INTERVAL. Соответственно, увеличилось и число возможных операций, видов выражений и предикатов. Но здесь для нас наибольший интерес представляют предикаты сравнения и IS NULL. Теперь сравниваться могут не только типизированные значения, но также и так называемые строчные значения (*row value*), которые представляют собой упорядоченные последовательности значений допустимых в SQL-92 типов данных, задаваемые с помощью конструктора строчного значения (*row value constructor*)[9].
   Поскольку на множествах значений всех типов данных SQL-92 имеется отношение порядка, и элементы строчного значения упорядочены, определение результатов почти всех операций сравнения не вызывает затруднений: два строчных значения, которые должны быть одной и той же степени, сравниваются в лексикографическом порядке, и значение *unknown* вырабатывается в том и только в том случае, когда при очередном сравнении элементов строчных значений один или оба элемента являются null-значениями.
   Не очень красивая ситуация имеет место в случае предиката IS NULL. Для строчного значения R условие R IS NULL принимает значение true тогда и только тогда, когда все элементы R являются null-значениями. Условие же R IS NOT NULL принимает значение true тогда и только тогда, когда ни один элемент R не являются null-значением. Поэтому

[8] Очень некорректно написано. Если тип данных – это множество представимых значений, для представления которых используются литеры, то единственным возможным толкованием оборота «элемент типа данных» является «элемент множества значений типа данных». Но это значение, для которого должен иметься литерал. А у null-значения литерала нет, значит оно не может являться элементом множества значений типа данных. Кроме того, никакой «элемент» не может быть элементом множества значений более одного типа данных.

[9] Вообще-то, введение операции сравнения над срочными значениями без введения типа строчных значений само по себе некорректно. Одним из последствий отсутствия типа строчных значений является невозможность использования null-значения для обозначения неизвестного строчного значения, которое в этом случае представляется последовательностью null-значений той же степени, что и само строчное значение. В соответствии с этим, если строчное значение конструируется с использованием подзапроса, возвращающего не более одной строки, то случай пустого результата интерпретируется как наличие строки соответствующей степени, состоящей только из null-значений. Очевидно, что подзапрос мог бы явно выдать одну строку null-значений, и эти два случая неразличимы.

вопреки интуиции для R степени больше 1 NOT (R IS NULL) <> R IS NOT NULL. Вот все возможные комбинации.

	R IS NULL	R IS NOT NULL	NOT R IS NULL	NOT R IS NOT NULL
degree 1: null	true	false	false	true
degree 1: not null	false	true	true	false
degree >1: all null	true	false	false	true
degree >1: some null	false	false	true	true
degree >1: non null	false	true	true	false

4) Условия поиска в SQL-92 строятся так же, как в SQL-89, но в основе построения логических выражений можно использовать не предикаты, но и явные проверки булевских значений (*boolean test*) вида B IS [NOT] {TRUE | FALSE | UNKNOWN}, где B – это предикат или условие поиска в круглых скобках. Эти операции всегда принимают значения *true* или *false* в зависимости от значения B. Это позволяет конструировать условия поиска, вычисляемые в двухзначной логике. Кроме того, введение обозначений логических значений можно считать шагом на пути к принятию полноценного булевского типа данных со своими литералами, хотя в SQL-92 такой тип так и не был введен.

2.1.4 SQL:1999 и SQL:200n

Ко времени написания этой статьи в 21-м веке были приняты пять вариантов стандарта SQL [18-23]. Объем стандарта все время увеличивается: если основная часть (*Foundation*) стандарта SQL:1999 [19] занимала 1150 стр., то в стандарте SQL:2016 [23] аналогичная часть насчитывает уже более 1700 стр. Однако все, что касается работы с неизвестными значениями и трехзначной логикой, появилось в SQL:1999 и переносилось в следующие версии стандарта практически без изменений. Суть соответствующих спецификаций состоит в следующем.

1) Начиная с SQL:1999, во всех версиях стандарта SQL содержится одно и то же определение null-значения, которое содержится в первой части стандарта (*Framework*) [18] и мало отличатся от определения, приведенного в стандарте SQL-92; null-значение: специальное значение, используемое для обозначения отсутствия какого бы то ни было типизированного значения. Больше в стандарте SQL ничего не говорится о природе null-значений (не считая спецификации булевского типа, см. ниже), но из этого определения следует, что, во-первых, null-значение может помечать отсутствие значения любого типа данных и, во-вторых, само null-значение не является значением никакого типа данных.
2) Появился конструктор типов строчных значений. Как и следовало ожидать, строчный тип определяется как последовательность пар (<имя поля> <тип данных>). Возможность определения строчного типа влечет возможность определения в таблице базы данных столбца такого типа и, соответственно, возможность использования null-значения в случае отсутствия соответствующего строчного значения в этом столбце в некоторых строках таблицы (если это не запрещено ограничением NOT NULL данного столбца). Поэтому в спецификации предиката сравнения в SQL:1999 прежде всего говорится, что результатом сравнения двух строчных значений является *unknown*, если хотя бы один из операндов является null-значение.

   Однако, как в SQL-92, во всех следующих версиях стандарта полагается, что если строчное значение конструируется с использованием подзапроса, и этот подзапрос выдает пустой результат, то образуется строчное значение, все элементы которого являются null-значениями. Заметим, что теперь (см. сноску 9 в п. 2.1.3) это (пустое) значение является типизированным, его строчный тип выводится из спецификации

подзапроса. Легко проверить, что предикат сравнения выдаст один и тот же результат *unknown* в обоих случаях. Более того, результатом применения IS NULL будет в обоих случаях *true*. Вообще, возможность наличия типизированного составного значения, все компоненты которого не типизированы, кажется очень подозрительной идеей.

3) В SQL:1999 появился очень странный булевский тип. Определение заслуживает дословного перевода.

   Тип данных boolean состоит из двух различимых истинностных значений true *и* false*. Если это не запрещено ограничением NOT NULL, в типе данных boolean также поддерживается истинностное значение* unknown *как null-значение. В этой спецификации не делается различия между null-значением типа данных boolean и истинностным значением* unknown*, которое является результатом применения предиката, условия поиска или вычисления выражения с булевским значением; они могут использоваться равноправно, означая в точности одно и то же.*

   Все значения типа данных boolean и истинностные значения SQL являются взаимно совместимыми и присваиваемыми. Значение true *больше, чем* false*; любое сравнение, в котором участвуют null-значение или истинностное значение unknown, приводит к результату unknown. Значения* true *и* false *могут быть размещены в любом месте, имеющем тип данных boolean; возможность размещения* unknown*, или null-значения зависит от того, допускается ли наличие null-значения в целевом объекте.*

Почему нам не нравится булевский тип современных стандартов SQL? Во-первых, абсолютно непонятно, как можно отождествлять в булевском типе реальное значение *unknown* и специальное null-значение (псевдозначение). Null-значение обозначает отсутствие типизированного значения и не может поэтому являться типизированным значением.

Во-вторых, следствием этого неправомерного отождествления является отсутствие реальной упорядоченности значений булевского типа. В трехзначной логике третье логическое значение имеет естественную семантику *maybe,* в соответствии с которой *true* > *unknown* > *false*. Казалось бы, что либо на множестве значений типа данных есть отношение порядка, либо его нет. Странно, когда два из трех значения булевского типа упорядочены, а у третьего своего места нет.

Наконец, начиная с первых попыток Кодда расширить реляционную модель средствами представления отсутствующей информации, считалось, что если в любом выражении встречается null-значение, то результатом вычисления этого выражения является null-значение. Как мы покажем в следующем разделе статьи, часто это соглашение является мягко говоря сомнительным, но стандарт SQL его придерживается для всех разновидностей выражений, кроме булевских. Почему арифметическое выражение $0 \times NULL = NULL$, а логическое выражение $false$ AND NULL $= false$?

2.2 Представление отсутствующей информации без null-значений

Этот подраздел практически целиком опирается на работы Дейта и Дарвена. Для начала заметим, что, начиная с 1984 г. (до выхода первого стандарта SQL) [24], Дейт систематически, почти во всех своих публикациях критиковал различные аспекты языка SQL. По нашему мнению, эта критика очень полезна для многочисленных разработчиков приложений баз данных, использующих SQL на практике. Она же в большой степени мотивировала и стимулировала разработку Дейтом и Дарвеном их собственной современной версии реляционной модели данных [11]. Однако в этой статье нас интересуют идеи Дейта и Дарвена по поводу средства работы с отсутствующей информацией.

2.2.1 Значения по умолчанию

Впервые к теме неопределенных значений Дейт обратился в четвертом издании своей знаменитой книги «Введение в системы баз данных» [25, 5.5 “Null Values”]. Через три года

этот материал в немного переработанной форме был опубликован в виде отдельной статьи [26]. Наиболее важными моментами являются следующие.

1) В [25] говорится, что проблема null-значения недостаточно хорошо изучена, из-за чего включение поддержки null-значений в работающую систему следует считать преждевременным. В [26] Дейт уже строго заявляет, что использование null-значения для представления отсутствующей информации не является удовлетворительным решением этой проблемы. Концепция null-значения создает гораздо больше проблем, чем решает.
2) Тем не менее, далее приводится схема работы с null-значениями полностью в духе Кодда [3] с той же путаницей между null-значением и третьим значением булевского типа. После этого следует критика с демонстрацией различных аномалий, к которым приводит наличие null-значения в языке SQL (Дейт ссылается на вариант SQL, использовавшийся в IBM и послуживший основой первых стандартов SQL). И, наконец, предлагается альтернативный подход к решению проблемы представления в базах данных отсутствующей информации на основе значений атрибутов отношений, назначаемых этим атрибутам по умолчанию.
3) В определении каждого атрибута любого именованного отношения присутствует раздел DEFAULT, в котором определяется значение по умолчанию этого атрибута, либо задается спецификация NODEFAILT, означающая отсутствие у атрибута значения по умолчанию. Для всех атрибутов, входящих в состав первичного ключа, обязательно отсутствие значений по умолчанию. Для всех прочих атрибутов при отсутствии раздела DEFAULT предполагается наличие предопределенного (зависящего от типа атрибута) значения по умолчанию (например, пробелов для символьных строк или нулей для числовых типов).
4) При вставке в отношение новой строки пользователь должен обеспечить явные значения всех атрибутов, у которых нет значений по умолчанию. Для других атрибутов при отсутствии явных значений система использует значения по умолчанию.
5) Имеется встроенная функция DEFAULT (R.A), возвращающая значение по умолчанию атрибута A базового отношения R. Если у атрибута A нет значения по умолчанию, фиксируется ошибка. С использованием этой функции можно корректно писать запросы, включающие вычисления агрегатных функций.

Как видно, в [25, 26] приводится лишь набросок предлагаемого подхода, и этот набросок повторяется спустя еще несколько лет в [27]. Поэтому [27] не способствует пониманию подхода Дейта, но в этой статье приводятся очень здравые соображения о невозможности определения трехзначного булевского типа при наличии универсального null-значения. Эти соображения созвучны тем, которые приводились в конце предыдущего подраздела по поводу ущербности булевского типа в современных стандартах SQL, и мы коротко их перескажем.

1) Если обозначить через UNK null-значение, обозначающее неизвестность данных, то результатом вычисления любого допустимого скалярного выражения x $alpha$ y, где $alpha$ – бинарная операция, а x и y – скалярные значения, является UNK, если хотя бы одним операндом является UNK.
2) Результатом любого допустимого выражения x $theta$ y, где $theta$ – операция скалярного сравнения, а x и y – скалярные значения, является третье логическое значение *unknown* (*unk*), если хотя бы одним операндом является UNK.
3) UNK и *unk* – это не одно и то же, т.е. истинностное значение *unknown* означает не то же самое, что «истинностное значение неизвестно». Если v – это переменная булевского типа, имеющая значение *unk*, то известно, что она имеет это значение. Это означает совсем не то же самое, что «значение v неизвестно».
4) Пусть для логических операций NOT, AND и OR действуют традиционные для трехзначной логики таблицы истинности. Предположим, что значение логической переменной неизвестно, т.е. обозначено через UNK. Можно согласиться, что результатом выражения *true* OR UNK может быть *true*, а результатом *false* AND UNK – *false*. Но результатом операции NOT UNK может быть только UNK. Значит ли это, что в действительности логика должна быть четырехзначной?

В законченном виде описание подхода к представлению отсутствующей информации с помощью значений атрибутов по умолчанию появилось спустя еще два года [9]. В описании выделяются структурные аспекты предлагаемой схемы, аспекты, связанные с поддержкой целостности баз данных, и манипуляционные аспекты. В отличие от предыдущих публикаций, в [9] используется терминология стандарта SQL (*таблицы* вместо *отношений*, *столбцы* вместо *атрибутов*, *строки* вместо *кортежей*), а вместо DEFAULT используется более уместное в этом контексте UNK (UNKNOWN).

1) *Структурные аспекты*. В определении любого столбца каждой таблицы либо должен присутствовать раздел UNK, в котором задается представление UNK для этого столбца, либо должна присутствовать спецификация UNKS NOT ALLOWED. UNK-значение для данного столбца (если оно существует) должно являться значением домена этого столбца. Для некоторых столбцов может оказаться, что любая допустимая конфигурация битов является возможным реальным значением этого столбца (не UNK). По мнению Дейта, на практике такая ситуация будет возникать достаточно редко, и поэтому конкретные средства для определения UNK не предлагаются. В таких случаях придется явно вводить отдельные, контролируемые пользователями столбцы-индикаторы.
   При вставке строки в базовую таблицу пользователь должен указать значение для любого столбца с UNKS NOT ALLOWED. Для остальных столбцов, если пользователь не указывает значение, система вставляет соответствующее UNK-значение.
   При добавлении столбца к базовой таблице определение этого столбца должно содержать раздел UNK, и во всех существующих строках таблицы в новый столбец заносится соответствующее UNK-значение.
2) *Аспекты, связанные с поддержкой целостности*. От правила *целостности сущностей* (значения первичных ключей базовых отношений не должны содержать null-значений) можно полностью отказаться, поскольку нет null-значения. Из этого следует также, что исчезает различие между первичным ключом отношения и любым его возможным ключом.
   Правило ссылочной целостности слегка упрощается: для каждого значения внешнего ключа должно существовать значение соответствующего целевого первичного ключа. Однако из этого упрощенного правила следует, что если значение внешнего ключа является UNK-значением, то в целевой таблице должна существовать строка, в которой значением первичного ключа также является UNK-значение.
3) *Манипуляционные аспекты*. Если R – это переменная диапазона, пробегающая по строкам некоторой базовой таблицы, а C – некоторый столбец этой таблицы, то встроенная функция UNK (R.C) возвращает UNK-значение этого столбца, если оно для него определено. Если столбец C определен с указанием UNKS NOT ALLOWED, фиксируется ошибка.
   Встроенная функция IS_UNK (R.C) эквивалентна истинностному выражению R.C = UNK (R.C), т.е. она возвращает *true* для тех и только тех строк таблицы, с которой связана переменная R, для которых значением столбца C является UNK-значение.
   Встроенная функция IF_UNK (R.C, exp) является сокращенной формой выражения IF IS_UNK (R.C) THEN exp ELSE R.C.

С одной стороны, описанный подход действительно позволяет сохранять в базе данных информацию о неизвестных данных и обеспечивает пользователям возможность писать корректные запросы за счет явного выявления наличия UNK-значений. С другой стороны, UNK-значения – это допустимые значения доменов соответствующих столбцов. Пользователи могут включать в запросы арифметические выражения и сравнения, содержащие UNK-значения разных столбцов. Результаты таких выражений, очевидно, бессмысленны, как и результаты соответствующих запросов. Другими словами, система

полностью перекладывает на пользователей ответственность за корректность работы с отсутствующей информацией.

2.2.2 Специальные значения

Дейт обычно ссылается на [10], как на основную работу, описывающую его усовершенствованный подход к представлению отсутствующих данных на основе двухзначной логики. Однако в [28], где Дейт и Дарвен в соответствующих местах также ссылаются на [10], в действительности содержатся некоторые важные исправления и уточнения. Поэтому при рассмотрении сути этого подходы мы будем одновременно использовать и [10], и [28][10].

1) При определении любого (скалярного) типа данных[11] можно дополнительно к множеству его основных значений специфицировать любое число специальных значений[12], поведение которых отличается от поведения основных значений. Для каждого специального значения типа T задаются его идентификатор (например, UNK (*unknown)* или NA (*non-applicable*)), а также литеральные представления тих значений (например, '?' для UNK или '!' для NA). [13]

   Система не знает смысла специальных значений, их интерпретация известна только пользователям. Однако система знает, что эти значения являются не обычными, а специальными значениями типа T, и для каждого специального значения предопределяет операции UNK_T () и NA_T (), возвращающие значения T ('?') и T ('!') соответственно.

   Для каждого специального значения типа T также предопределяются операции вида IS_UNK (scalar_exp), где scalar_exp – выраженне типа T. Операция возвращает истинностное значение *true*, если значением этого выражения является UNK_T (), и *false* в противном случае.

   Наконец, предлагаются две операции внешнего соединения отношений (LEFT JOIN и RIGHT JOIN), которые похожи на соответствующие операции NATURAL OUTER JOIN SQL, но вместо null-значений используются явно указываемые специальные значения.

[10] Хотя при этом возникают трудности, связанные с разным контекстом: в [10] используется контекст стандарта SQL, а в [28] – контекст определяемого варианта реляционной модели данных. В своем изложении мы будем стараться сгладить различия этих контекстов.

[11] В [28] под *скалярным типом данных* понимается инкапсулированный тип, *реальные представления* значений которого скрываются от пользователей за одним или нескольких *возможных представлений*, каждое из которых является некоторой структурой (записью) с типизированными полями (ранее определенных или предопределенных типов), взаимно однозначно отображаемой на реальное представление типа его реализатором, Операции над значениями скалярного типа определяются в терминах одного из его возможных представлений и автоматически отображаются в операции над его реальным представлением.

[12] Именно этим объясняет Дейт в [10] переход от термина *значение по умолчанию* к термину *специальное значение*: специальное значение не может служить в качестве значения по умолчанию, потому что в одном типе может быть много специальных значений. Однако, по нашему мнению, не менее, а скорее более важным отличием специальных значений от значений по умолчанию является то, что в [9] значением столбца по умолчанию служит выбираемое пользователем обычное значение домена (типа данных), а в [10, 28] специальные значения являются явно вводимыми дополнительными значениями, про которые системе известно, что каждое из них отличаются как от любого обычного значения, так и любого другого специального значения того же типа.

[13] Сравнительно подробно схема специальных значений обсуждается только для скалярных типов данных, хотя в [28] вводятся еще две разновидности типов – кортежные типы (похожие на строчные типы SQL) и типы отношений, основанные на кортежных типах. Типы данных этих разновидностей являются безымянными, так что напрямую подход, применяемый для именованных скалярных типов, для них не работает. В одной из сносок в [28] мимоходом говорится, что нужно было бы что-то придумать по этому поводу, но до этого дело не дошло.

2) Для любых двух значений a и b (обычных и специальных) типа T определены операции сравнения по равенству и неравенству. Сравнение $a = b$ возвращает значение *true* в том и только в том случае, когда a и b являются одним и тем же значением. Сравнение $a \neq b$ возвращает значение *true* в том и только в том случае, когда a и b не являются одним и тем же значением. Однако при попытке определить на основе двухзначной логики, например, операции сравнения $a >$ UNK_T () и $a \leq$ UNK_T () как возвращающие $false$ для любого значения a, отличного от UNK_T (), возникает противоречие. Действительно, предположим, что $(a >$ UNK_T $()) = false$. Тогда $(a \leq$ UNK_T $()) =$ NOT $(a >$ UNK_T $()) =$ NOT $(false) = true$.

   В результате в [10] Дейт приходит к выводу, что при использовании типов со специальными значениями сравнение значений a и b на $>, <, \geq, \leq$ допускается в тех и только тех случаях, когда a и b являются обычными значениями. Если хотя бы один из операндов такой операции сравнения является специальным значением, возникает исключительная ситуация. Заметим, что в [28] операции сравнения на $>, <, \geq, \leq$ для типов со специальными значениями уже не упоминаются.

3) Вычислительные выражения со значениями типов со специальными значениями никогда не приводят к результатам, являющимся специальными значениями. В частности, отсутствует предписание, в соответствии с которым $a +$ UNK_T () = UNK_T () для любого значения a типа T. Это мотивируется следующими соображениями. Если $a +$ UNK_T () = UNK_T (), то из соображений согласованности должно быть и $a -$ UNK_T () = UNK_T (). Но тогда должно выполняться и UNK_T () − UNK_T () = UNK_T (), что, по мнению Дейта, может привести к аномалиям, похожим на аномалии null-значений в SQL.

Судя по всему, Дейт и Дарвен были не слишком удовлетворены свой схемой специальных значений (мы вернемся к ней в следующем разделе статьи), потому что в последнем издании их книги о современном представлении реляционной модели данных [11] вообще отсутствуют предложения по поводу средств работы с отсутствующей информацией.

2.2.3 Сохранение в базе данных отсутствующей информации без поддержки null- или специальных значений

Однако спустя четыре года после публикации [11] Дейт и Дарвен выпустили книгу [29], в которой отдельная часть из четырех глав посвящена различным подходам, позволяющим сохранять и использовать в базах данных отсутствующую информацию без использования каких-либо средств явного представления такой информации. Эта часть книги написана Хью Дарвеном, который во введении отмечает, что не стремится отстаивать достоинства ни одной из описываемых схем, а публикуются они в целях их обсуждения.

Первая из описываемых в [29] схем основывается на *декомпозиции отношений*. Идея принадлежит Дарвену и была изначально представлена в [30].

1) Предположим, что имеется переменная отношения $r\,\{key, a1, a2, \ldots, an, b1, b2, \ldots, bm\}$, в любом допустимом значении которой информация может отсутствовать только в атрибутах $b1, b2, \ldots, bn$. Тогда на первом шаге r вертикально декомпозируется на $m + 1$ переменных отношений $ra\,\{key, a1, a2, \ldots, an\}$, $rb1\,\{key, b1\}, \ldots, rbm\,\{key, bm\}$, в допустимых значениях которых информация может отсутствовать только в атрибутах bi $(i = 1, \ldots, m)$.
2) На втором шаге каждая переменная отношения $rbi\,\{key, bi\}$ горизонтально декомпозируется на переменные отношений $rbi\,\{key, bi\}, rbi1\,\{key\}, \ldots, rbik\,\{key\}$ такие, что в допустимых значениях rbi атрибут bi содержит обычные значения, а в допустимых значениях $rbi1$ содержатся кортежи, в которых значению ключа соответствует отсутствие информации в атрибуте bi вида 1 (например, значение bi неизвестно), …, в допустимых значениях $rbik$ содержатся кортежи, в которых значению

ключа соответствует отсутствие информации в атрибуте bi вида k (например, значение bi неприменимо).

3) Чтобы можно было «склеить» значения переменных отношений $ra, rbi, rbi1, \dots, rbik$ $(i = 1, \dots, m)$, нужно, чтобы поддерживалось специальное ограничение целостности: для каждого значения ключа ra должно найтись совпадающее с ним значение ключа в значении ровно одной переменной $rbi, rbi1, \dots, rbik$.
4) Запросы над такими декомпозированными базами данных с неявно содержащейся отсутствующей информации можно выполнять с использованием переименования атрибутов, добавления столбцов, заполнения их определяемыми пользователями обозначениями отсутствующих данных и выполнения операций естественного соединения.

На маленьком примере, приведенном Дарвеном, все это выглядит нестрашно, но для отношений с большим числом атрибутов, в которых могут отсутствовать данные, по нашему мнению, практически неприменимо.

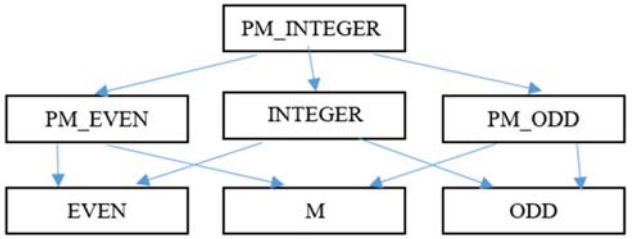
Второй подход основан на понятии *мультиотношения*. В некотором смысле этот подход похож на подход на основе декомпозиции, поскольку в хранимых кортежах отсутствуют атрибуты с отсутствующими данными, но позволяет обойтись без порождения многочисленных переменных отношений и без потребности в их соединении при выполнении запроса. Приводится только набросок этой схемы, но и он слишком объемен, чтобы можно было описать его подробно в этой статье. Ограничимся очень кратким изложением основных идей.

1) Как и у обычных отношений, у каждого мультиотношения *mr* есть заголовок *MRH* и тело *MRB*. Однако в тело *mr* могут входить все кортежи, соответствующие любому подмножеству *MRH* (включая пустое подмножество), т.е. кортежи *mr* могут иметь степени от 0 до n, где n – степень заголовка.
2) Участником мультиотношения *mr* называется отношение *p* с заголовком *PH*, являющимся некоторым подмножеством *MRH,* и телом *PB,* включающим все кортежи из *MRB*, которые соответствуют *PH.* Тем самым, у любого мультиотношения степени *n* имеется 2^n участников, некоторые из которых могут быть пустыми.
3) Для переменной мультиотношения *MR* можно определять ограничения целостности. Обязательным ограничением любой *MR* является ограничение ключа, в котором задается такое подмножество *MRH* $\{a1, a2, \dots, an\}$, что все атрибуты ai участвуют в каждом кортеже любого допустимого в *MR* значения *mr* и во всех кортежах составные значения $\{a1, a2, \dots, an\}$ различны.
4) Определяется набор алгебраических операций над мультиотношениями, часть которых является прямым переложением определений реляционных операций в терминах мультиотношений. Однако определение некоторых операций настолько же неочевидно, как и в схеме специальных значений (см. п. 2.2.2). К числу таких операций относится совершенно необходимая операция ограничения (фильтрации) мультиотношений. В условие ограничения *mr* могут входить имена атрибутов, не входящих в некоторые кортежи *mr*, и чтобы решить проблему вычисления условия для этих кортежей без потребности в трехзначной логике, предлагается в обязательном порядке использовать предикаты PRESENT и ABSENT, являющиеся прямыми аналогами IS [NOT] NULL в SQL.

Дарвен полагает, что на основе схемы мультиотношений можно было бы разработать новый язык баз данных, который позволял бы обращаться с таблицами SQL как с мультиотношениями, полагая что в строке таблицы, в которой некоторый столбец содержит null-значение, этот столбец можно считать отсутствующим. Однако, по нашему мнению, это не принесет особых преимуществ, поскольку существующий SQL и так позволяет избежать трехзначной логики, если в каждом условии использовать IS [NOT] NULL.

Следующий подход, обсуждаемый в [29], опирается на использование *механизма наследования*. Идея этого подхода принадлежит Эрвину Смауту [31]. В детали мы здесь вдаваться не будем, поскольку для этого потребовалось бы описать суть необычного и нетривиального (хотя очень логичного) механизма наследования типов данных, определенного в [11].

1) Суть идеи состоит в том, что в корневом супертипе любой иерархии наследования типов[14] можно определить, кроме множества обычных значений этого типа, значение *MISSING.* После этого можно определить любой подтип в двух вариантах – T и PM_T, в первом из которых содержатся только обычные значения, а во втором – еще и значение *MISSING.* Вот пример графа подтипизации, в котором определяются типы PM_INTEGER (все целые числа плюс *MISSING*), INTEGER (все целые числа), PM_EVEN (четные целые плюс *MISSING*), EVEN, PM_ODD (нечетные целые плюс *MISSING*), ODD, а также тип M, включающий единственное значение *MISSING*[15].



2) Значение *MISSING* не может участвовать в выражениях любого типа, которому принадлежит *MISSING*. Допускается сравнение по равенству и неравенству всех значений любого типа, включающего *MISSING*, с очевидной семантикой на основе интерпретации *MISSING* как неизвестного значения. Сравнения на $>, <, \geq, \leq$ с участием значения *MISSING* запрещаются (исключительная ситуация?).

Все это дает основания полагать, что схема на основе наследования не дает ничего такого, чего бы не обеспечивала схема специальных значений (в [29] к этому выводу приходит сам Дарвен).

Наконец, последний подход основан на использовании *атрибутов со значениями-отношениями.* Основная идея состоит в том, чтобы использовать пустое множество в качестве значения атрибута, реальное значение которого отсутствует. Более конкретно, предлагается следующее.

1) Если требуется определить в базе данных переменную отношения $R\{a1{:}T1, a2{:}T2, \dots, an{:}Tn\}$ такую, что в допустимых значениях r переменной R у некоторого атрибута ai в некоторых кортежах r могут отсутствовать значения, то в действительности определяется переменная отношения $ZR\{a1{:}T1, a2{:}T2, \dots, zai{:}RELATION\{ai{:}T\}, \dots, an{:}Tn\}$. Другими словами, *RH* отличается от *ZRH* тем, что типом (переименованного) i-го атрибута становится тип унарного отношения с атрибутом ai типа *T*. При этом в любом кортеже любого допустимого значения zr переменной *ZR* значение-отношение атрибута zai не должно иметь мощность больше 1[16]. Такие отношения степени 1 и мощности не большей 1 Дарвен называет *ZOO*-отношениями (cardinality Zero or One, degree One).

[14] На самом деле, это не совсем иерархия, поскольку в [11] поддерживается множественное наследование: тем не менее, у любой такой «иерархии» имеется корневой супертип.

[15] Тип M появляется в графе наследования, поскольку в применяемой модели наследования каждое значение супертипа должно являться значением хотя бы одного его непосредственного подтипа.

[16] Это ограничение переменной отношения *ZR.*

2) Если в условии и/или результате запроса к zr участвовали значения атрибута ai, то (1) из zr удаляются все кортежи, с пустыми значениями атрибута zai[17]; (2) в добавляется атрибут $ai{:}\,T$, и в каждом кортеже получаемого отношения $zr1$ значением ai становится значение атрибута ai в единственном кортеже значения-отношения атрибута zai того же кортежа $zr1$; (3) из $zr1$ удаляется атрибут zai, и над получаемым отношением обычным образом выполняется фильтрация и формирование результирующего отношения.

Подводя итог этого пункта, явно заметим, что только первый из четырех рассмотренных подходов предлагает решение, реально выходящее за пределы схемы специальных значений, но приводящее в общем случае к нереалистично большому числу переменных отношений в базе данных, усложнению и потенциальному замедлению выполнения запросов. Каждый из трех последних подходов логичен и красив, но по сути это переложение идей специальных значений.

3. Типизированные неизвестные значения и трехзначная логика

Как показывает разд. 2, в настоящее время отсутствует удовлетворительное решение проблемы поддержки в базах данных отсутствующей информации. Несмотря на многолетние попытки решить эту проблему подход, применяемый в SQL-ориентированных СУБД на основе стандарта SQL, и схемы, предлагавшиеся Дейтом и Дарвеном, обладают, на наш взгляд, существенными недостатками, В этом разделе мы предпринимаем еще одну попытку предложить подход к решению проблемы отсутствующей информации. Но сначала обсудим причины неудач предыдущих подходов, что требуется для мотивации наших собственных идей.

3.1 Почему имеющиеся подходы неудачны?

Подходы, идущие от Кодда и приведшие к схеме, которая используется в современном SQL, и предлагавшиеся Дейтом и Дарвеном, неудачны по-разному, и причины этих неудач нуждаются в раздельном обсуждении.

3.1.1 Тупик на пути Кодда

Корень зла кроется, конечно, в понятии универсального null-значения, которое не является значением ни одного типа данных, но может использоваться вместо любого значения любого типа данных в любом месте, где могло бы появиться это значение. Конечно, при таком определении null-значения естественно полагать, что результатом вычисления любого выражения, в котором вместо реального значения встречается NULL, является NULL.

Однако, не говоря уже о более серьезных последствиях этого подхода, которые мы обсудим немного позже, сам факт разрешения использовать в выражениях чего-бы то ни было, кроме значений типа данных этого выражения, является парадоксальным и неприемлемым. Конечно, эту ситуацию никак не исправляет отказ Кодда в его поздних работах от термина *null-значение* с его заменой на термин *маркер*.

Но даже если закрыть глаза на теоретическую недопустимость использования null-значения в типизированных выражениях, правила вычисления выражений, содержащих универсальное null-значение, нарушают принцип замены неопределенных значений самого Кодда [3]. Этот принцип почему-то формулируется только для операций сравнения (операция сравнения выдает третье логическое значение *unknown* в том и только в том случае, когда при замене всех вхождений null-значения одним набором реальных значений оно выдает *true*, а при их замене другим набором реальных значений – *false*).

[17] Конечно же, на основе предиката IS_EMPTY.

Очевидно, что аналогичный принцип должен использоваться при вычислении всех видов операций: операция выдает null-значение в том и только в том случае, когда при замене всех вхождений null-значения разными наборами реальных значений она может выдавать разные значения. Тривиальным примером является арифметическое выражение $0 \times \text{NULL}$, значением которого, конечно, должен быть 0, а не NULL. Но для этого требуется, чтобы в контексте вычисления арифметического выражения null-значение интерпретировалось не как универсальный маркер отсутствующего значения, а как значение соответствующего числового типа (хотя и неизвестное).

Зато принцип замены неопределенных значений прекрасно работает для булевских операций: операция *true* OR NULL, очевидно, должна выдавать *true*, *false* OR NULL = NULL, NOT NULL = NULL и т.д. Другими словами, при следовании этому принципу для булевского типа данных универсальное null-значение ведет себя точно так же, как третье логическое значение *unknown*. При этом к самому принципу замены претензий нет, он полностью логичен.

Таким образом, мы видим, что появление в современных стандартах SQL неполноценного трехзначного булевского типа данных (см. конец п. 2.1.4) вполне закономерно, и единственной причиной этой неполноценности является поддержка единственного и универсального null-значения. Разработчики стандарта SQL:1999 [19] и более поздних стандартов справедливо хотели оснастить стандарт современной системой типов данных. Обойтись без булевского типа, значения которого можно сохранять в базах данных, было невозможно. Но тогда из-за универсальности null-значения приходится допустить наличие в базе данных неизвестных булевских значений, обозначаемых null-значением, что, в свою очередь, приводит к недопустимому отождествлению значения булевского типа *unknown* и null-значения, которое является лишь обозначением отсутствия реального значения. По-другому просто не получается.

Еще одним следствием ущербности булевского типа SQL является потеря упорядоченности булевских значений в трехзначной логике. Было бы логично полагать, что $false < unknown < true$. Если сопоставить логическим значениям $false, unknown, true$ числа $0, 0.5, 1$ в том же порядке, то логические операции выражаются через численные операции: $\text{NOT}\, x = 1 - x;\ x \text{ AND } y = min\,(x, y), x \text{ OR } y = max\,(x, y)$. Из-за отождествления *unknown* и null-значения неизвестное логическое значение теряет свою позицию.

Можно сказать, что булевский тип в стандарте SQL – это не только не настоящий трехзначный булевский тип, но и не тип данных вообще. С нашей точки зрения, это полный тупик в многолетней истории попыток решить проблему отсутствующей информации на основе универсального null-значения.

В заключение этого пункта сделаем несколько замечаний по поводу последней схемы Кодда с двумя маркерами (A-маркер a для неизвестных значений и I-маркер i для неприменимых значений) и четырехзначной логикой (см. п. 2.1.2).

Во-первых, совершенно непонятны правила вычисления арифметических выражений

$$\begin{array}{cccc} \text{z}\,\%\,\text{a} = \text{a} & \text{z}\,\%\,\text{i} = \text{i} & \text{a}\,\%\,\text{z} = \text{a} & \text{i}\,\%\,\text{z} = \text{i} \\ \text{a}\,\%\,\text{a} = \text{a} & \text{a}\,\%\,\text{i} = \text{i} & \text{i}\,\%\,\text{a} = \text{i} & \text{i}\,\%\,\text{i} = \text{i} \end{array},$$

в которых маркер неизвестных значений ведет себя совершенно так же, как и маркер неприменимых значений. Если, например, можно согласиться с тем, что $\text{z} + \text{a} = \text{a}$, то никак нельзя принять, что $\text{z} + \text{i} = \text{i}$ (из последнего следует, например, что нельзя посчитать месячный доход служащего, которому не полагается премия).

Во-вторых, аналогичные сомнения применимы и к правилам определения результатов операций сравнения[18]

$$\text{z}\,\Theta\,\text{a} = \text{a}_\text{B} \qquad \text{z}\,\Theta\,\text{i} = \text{i}_\text{B} \qquad \text{a}\,\Theta\,\text{z} = \text{a}_\text{B} \qquad \text{i}\,\Theta\,\text{z} = \text{i}_\text{B}$$

[18] Для справедливости еще раз повторим, что Кодд в явном виде эти правила не определял. Они построены автором данной статьи «по образу и подобию» других конструкций Кодда.

$a\,\Theta\,a = a_B$ $a\,\Theta\,i = i_B$ $i\,\Theta\,a = i_B$ $i\,\Theta\,i = i_B$,

в соответствии с которым четвертое логическое значение i_B формируется точно так же, как и третье логическое значение a_B (*unknown*). Но если мы будем продолжать понимать *unknown* в смысле *maybe*, то i_B должно означать *not-maybe.* Можно согласиться с тем, что $z > a = a_B$, но никак нельзя принять, что $z > i = i_B$ (наверное, это следует понимать так: не может существовать служащий с размером зарплаты, большим размера премии, если этот служащий вообще не получает премию). Да и то, что, $i\,\Theta\,i = i_B$, вызывает большие сомнения.

Далее, еще в 1990-м г. Джин Гессерт [32] обратил внимание на то, что определение Коддом операции отрицания, в соответствии с которым $\mathrm{NOT}\,(a_B) = a_B$ и $\mathrm{NOT}\,(i_B) = i_B$, приводит к тому, что в его четырехзначной логике перестают действовать законы де Моргана. В частности, $i_B = \mathrm{NOT}\,(a_B\ \mathrm{AND}\ i_B) = \mathrm{NOT}\,(a_B)\ \mathrm{OR}\ \mathrm{NOT}\,(i_B) = a_B\ \mathrm{OR}\ i_B = a_B$.

В [33] Дейт анализирует всю историю четырезхначного подхода Кодда и отмечает, что в ней имелось три итерации. После публикации в [8, 13] первого варианта этой схемы, в которой $false\ \mathrm{OR}\ i_B = i_B\ \mathrm{OR}\ false = false$, во втором тираже [13] таблица истинности для OR была изменена так, что $false\ \mathrm{OR}\ i_B = i_B\ \mathrm{OR}\ false = i_B$ (как замечает, Дейт, это по сути ничего не изменило). Наконец, по словам Дейта [33], впоследствии Кодд еще раз пересмотрел свои таблицы истинности, определив $\mathrm{NOT}\,(a_B) = i_B$ и $\mathrm{NOT}\,(i_B) = a_B$ (соответствующая публикация, скорее всего, сделана не была). После этого законы де Моргана заработали.

Кроме того, если обозначить логические значения $false, i_B, a_B, true$ числами $0, \frac{1}{3}, \frac{2}{3}, 1$ в том же порядке, то логические операции, как и в трехзначной логике, выражаются через численные операции: $\mathrm{NOT}\ x = 1 - x;\ x\ \mathrm{AND}\ y = min\,(x, y), x\ \mathrm{OR}\ y = max\,(x, y)$.

Однако, даже если смириться со всеми неясностями и странностями подхода с двумя маркерами и четырехзначной логикой, возникает та же проблема, что и при наличии одного null-значения (только еще более неприятная), вызываемая универсальностью и общеупотребимостью A-маркера и I-маркера. Если разрешить сохранять логические значения в базах данных (т.е. разрешить включение в отношения атрибута типа BOOLEAN), то естественно допустить, что в некоторых кортежах значения этого атрибута может быть неизвестным, а в некоторых – неприменимым.

Это приводит к необходимости определения результатов операций NOT (a), NOT (b), $true\ \mathrm{AND}\ a, true\ \mathrm{AND}\ b$ и т.д. Напрягая воображение, можно согласиться с тем, что двуместные логические операции с операндами a и b ведут себя так же, как соответствующие операции с операндами a_B и i_B, т.е., например, $true\ \mathrm{AND}\ a = a$, $true\ \mathrm{AND}\ b = b$, но никакое воображение не может допустить, что $\mathrm{NOT}\,(a) = b$. Другими словами, чтобы можно было отождествить в контексте булевского типа логические значения a_B и i_B с маркерами a и b соответственно, нужно вернуться к исходному определению Коддом операции отрицания ($\mathrm{NOT}\,(a_B) = a_B$ и $\mathrm{NOT}\,(i_B) = i_B$), которое приводят к тому, что законы де Моргана не работают.

Вывод состоит в том, что схема Кодда с двумя маркерами и четырехзначной логикой не позволяет определить даже ущербный булевский тип в духе стандарта SQL.

3.1.2 Рыцари двухзначной логики

Начиная с 1984-го года, в бесчисленных публикациях Дейт сначала в одиночку, а потом вместе с Дарвеном критиковал Кодда и язык SQL за использование null-значения и трехзначной логики. Практически всегда эта критика была обоснованной с приведением примеров запросов, которые выдают противоречащие интуиции результаты. Одновременно Дейт и Дарвен пытались придумать решение проблемы отсутствующей информации с использованием традиционной двухзначной логики. Однако, по нашему мнению, им так и не удалось изобрести схему, которая могла бы конкурировать с плохо обоснованной, логически

противоречивой, но пригодной для практического использования схемой работы с отсутствующей информацией стандарта языка SQL.

В действительности, наличие трехзначной логики не слишком беспокоит пользователей SQL-ориентированных СУБД, поскольку возможность включать в выражения null-значение и интерпретация результата *unknown*, возникающего при вычислении условных выражение разделов WHERE и HAVING запросов, как запрещающего, т.е. эквивалентного *false* по своему действию, позволяет в большинстве случаев формулировать интуитивно понятные запросы, выдающие результаты, также соответствующие интуиции. В сомнительных случаях при построении условий выборки можно использовать предикат IS [NOT] NULL, позволяющий избежать null-значения в выражениях и, тем самым, остаться в рамках двухзначной логики.

Кратко подытожим плюсы и минусы подходов Дейта и Дарвена, обсуждавшихся в подразделе 2.2. Очевидным минусом всех подходов (с нашей точки зрения) является перекладывание на пользователей учета наличия отсутствующих данных в отношениях, к которым адресуются запросы. Поскольку пользователи, формулирующие запросы, не обязаны знать, содержат ли все атрибуты во всех кортежах отношений только реальные данные или же в некоторых кортежах значения некоторых атрибутов отсутствуют, все запросы должны содержать условия типа IS [NOT] UNK и явно описываемые операции, которые нужно выполнить, если такое условие принимает значение *true*.

Не будем здесь повторять, чем нас не устраивает подход значений по умолчанию (см. конец п. 2.1.1). Но вот у следующего подхода, основанного на расширении типов данных специальными значениями, имеется очень серьезный потенциал, не используемый его создателями. Главными чертами этого подхода является то, что (а) специальные значения не универсальны и (б) для каждого типа данных можно определить свои специальные значения. В принципе можно было позволить системе или пользователям определять поведение специальных значений в операциях соответствующего типа данных.

Но даже если ограничиться только одним специальным значением UNK_T () (см. п. 2.2.2), в двухзначной логике, которой принципиально придерживаются Дейт и Дарвен, определить эти операции оказывается невозможно. В соответствии с семантикой UNK_T () (*maybe*), $a - \mathrm{UNK\_T}\,() = \mathrm{UNK\_}\,() - a = \mathrm{UNK\_T}\,() - \mathrm{UNK\_T}\,() = \mathrm{UNK\_T}\,()$, $a\ \ a\,\theta\,\mathrm{UNK\_T}\,() = \mathrm{UNK\_T}\,()\ \theta\ a = \mathrm{UNK\_T}\,()\ \theta\ \mathrm{UNK\_T}\,() = \mathrm{UNK\_BOOLEAN}\,()$[19] для каждого неспециального значения a типа T. Как видно, здесь UNK_BOOLEAN () – это специальное значение, добавленное к двухзначному булевскому типу. Очевидно, что попытки определить поведение этого специального значения в операциях NOT, AND и OR приводят нас к традиционной трехзначной логике, в которой UNK_BOOLEAN () играет роль третьего логического значения.

Но поскольку трехзначная логика запрещена, Дейт полностью запрещает и использование специальных значений в выражениях и операциях сравнения для всех видов сравнения, кроме = и ≠. Но даже при таких ограничениях кажется сомнительным, что $\mathrm{UNK\_T}\,() = \mathrm{UNK\_}\,()$, а $a \neq \mathrm{UNK\_T}\,(), \mathrm{UNK\_}\,() \neq a$ для каждого неспециального значения a типа T. В результате единственным способом формулировать запросы к базе данных, атрибуты отношений которой могут содержать специальные значения типа T, является явное использование предиката IS_UNK.

Как мы отмечали в конце п. 2.2.3, из четырех подходов, предложенных Дарвеном после схемы специальных значений, только в первом (подход с декомпозицией отношений) специальные значения явно или неявно не используются. Вообще, этот подход кажется нам перспективным, если не замахиваться на поддержку произвольного числа причин отсутствия

[19] Мы умышленно используем обозначения Дейта, чтобы показать ход рассуждений, который можно было применить на его месте.

данных в атрибутах отношений. В общем виде, представленном в [29], он не кажется практически применимым.

3.2 Если бы губы Никанора Ивановича да приставить к носу Ивана Кузьмича

Собственно, это мы и пытаемся сделать в этом подразделе, а именно, мы скрещиваем специальные значения Дейта с трехзначной логикой Кодда и, отчасти, с подходом Дарвена с декомпозицией отношений. Конечно, для этого нужно многое пересмотреть в обоих подходах, но и получаемые преимущества выходят за пределы простого объединения положительных черт каждого из этих подходов.

В предлагаемой схеме мы ограничиваемся только одной разновидностью специальных значений с семантикой *maybe*. Этому есть несколько объяснений:

- чаще всего точные данные невозможно занести в базу данных именно потому, что они неизвестны;
- семантика неизвестных данных кажется нам значительно понятнее, чем семантика неприменимых данных (*non-available*, или *not-maybe*);
- мы не хотим выходить за пределы трехзначной логики;
- в дополнение к основной схеме мы предлагаем использовать идеи Дарвена, что в случае потребности справиться с неприменимыми данными без потребности в четырехзначной логике;
- наконец, прочие разновидности причин отсутствия данных кажутся нам надуманными и/или редко встречающимися.

При этом мы хотим добиться следующих целей:

- сделать специальные значения полноценными и полноправными значениями соответствующих типов данных с возможностью применения к ним всех операций с поддержкой интуитивной семантики, вообще говоря, индивидуальной для каждого типа данных;
- сделать трехзначный булевский тип полноценным и полноправным типом данных за счет интерпретации специального значения, расширяющего двухзначный булевский тип, как третьего логического значения трехзначного булевского типа.

Обсудим, каким образом можно достичь этих целей. Поскольку, в конечном счете, мы стремимся «облагородить» механизмы null-значений и трехзначной логики стандарта SQL, будем использовать терминологию SQL[20].

1) Множество значений {t} каждого допустимого в стандарте SQL типа данных T расширяется специальным значением t_ukn, отличным от любого t. Для предопределенных и конструируемых типов такое расширение с определением семантики операций, включающих операнды-специальные значения, можно было бы предопределить. Для типов данных, определяемых пользователями, это должны делать разработчики соответствующих новых типов данных.
2) Операции над значениями расширенных типов должны быть уточнены с учетом семантики основных значений и возможности использования специального значения в качестве операнда. Например, для всех числовых типов данных, очевидно, $0 \times$ t_ukn $=$ t_ukn $\times 0 = 0$, а для типов множеств $\{\emptyset\}$ INTERSECT t_ukn = t_ukn INTERSECT $\{\emptyset\} = \{\emptyset\}$.

   Для типов символьных и битовых строк, а также для типов массивов результатом конкатенации обычного значения со специальным значением в действительности должен являться первый операнд, за которой следует специальное значение. Для такой «строки» можно было бы выполнять операции взятия или поиска позиции подстроки, не обязательно получая в результате t_ukn соответствующего типа.
3) Двухзначный булевский тип также расширяется специальным значением, которое в силу его повышенной важности мы будем называть особым образом – *unknown*. По определению специальное значение *unknown* является результатом выполнения любой операции сравнения, если эта операция не вырабатывает обычное булевское значение *true* или *false*. Для множества значений расширенного булевского типа вводится отношение порядка $false < unknown < true$.

   Естественным образом с учетом семантики двухзначного булевского типа и специального значения *unknown* доопределяется набор операций расширенного булевского типа:

$$\text{NOT } unknown = uknown,$$

$$(uknown \text{ AND } uknown) = (true \text{ AND } uknown) = (uknown \text{ AND } true) = unknown,$$

$$(false \text{ AND } uknown) = (uknown \text{ AND } false) = false,$$

$$(true \text{ OR } uknown) = (uknown \text{ OR } true) = true,$$

$$(uknown \text{ OR } uknown) = (false \text{ OR } uknown) = (uknown \text{ OR } false) = unknown.$$

   Другими словами, $\{false, unknown, true\}$ – это множество значений полноценного трехзначного булевского типа и при этом $unknown$ остается специальным значением этого типа с семантикой *maybe*.
4) Хотя, вообще говоря, результатом любой операции сравнения $t\_ukn\ \theta\ t\_ukn$, $t\ \theta\ t\_ukn, t\_ukn\ \theta\ t$, где θ – $=, \neq, >, \geq, <, \leq$, а t – обычное значение некоторого типа T, является $unknown$, в операциях сравнения должны учитываться табличные ограничения тех таблиц, из столбцов строк которых выбираются операнды. Например, если для столбца целого типа установлено ограничение $value > 0$, и в некоторых строках соответствующей таблицы значением этого столбца является t_ukn, то для этих строк результатом сравнения $t\_ukn > 0$, должно являться *true*. Аналогично, пусть в таблице имеются два столбца $C1$ и $C2$ одного и того же типа, и уставлено табличное ограничение $CHECK\ C1 > C2$, Тогда, если в некоторых строках этой таблицы значением столбца $C2$ является t_ukn, то для этих строк результатом сравнения $C1 > t\_ukn$, должно являться *true.*
5) Условие выборки запроса к базе данных, вычисляемое для очередной строки (группы строк) таблицы, является разрешающим в том и только в том случае, когда результатом вычисления является *true*. Чтобы можно было формулировать запросы, направленные на выборку строк, которые содержат специальные значения, для каждого типа данных T определяется операция IS t_ukn (T_exp), где T_exp – выражение типа T, результатом которой является *true*, если результатом T_exp является t_ukn, и *false* в противном случае.
6) В SQL имеются две полезные конструкции *SCALAR SUBQUERY* и *ROW SUBQUERY* с неочевидной семантикой.

   SCALAR SUBQUERY – это вложенный подзапрос, результатом которого должна являться таблица SS степени 1, содержащая ровно одну строку. По правилам SQL результат скалярного подзапроса неявно преобразуется в значение единственного столбца единственной строки таблицы SS. При этом, если мощность SS оказывается больше 1, возникает исключительная ситуация, а если мощность SS равна 0, то выдается NULL.

[20] Это не означает, что описываемые предложения можно напрямую использовать для переделки стандарта SQL. Стандарт SQL при всех его недостатках – это гигантская, тщательно продуманная и сбалансированная спецификация, в которой изменения в одной части неминуемо влекут изменения в других частях (скорее всего, во многих других местах). И, конечно, это в полной мере относится к старейшему унаследованному механизму null-значений. Поэтому мы даже не будем пытаться формулировать все технические проблемы, с которыми пришлось бы столкнуться при замене этого механизма на нечто, основанное на наших идеях.

Точно так же ведет себя операция *ELEMENT*, извлекающая ровно одно значение из заданного мультимножества, мощность которого должна быть равна 1.

Можно понять причины, по которым в стандарте SQL были приняты такие правила, поскольку скалярные подзапросы могут участвовать в выражениях в разных местах SQL-запроса. Даже неочевидное назначение NULL в качестве значения подзапроса с пустым результатом можно оправдать, поскольку в SQL NULL – это универсальное обозначение отсутствия данных. Но в предлагаемой нами схеме совершенно недопустимо считать, что результатом пустого скалярного подзапроса является специальное значение типа столбца таблицы SS.

Похожим образом ведет себя *ROW SUBQUERY* – вложенный подзапрос, результатом которого должна являться таблица RS, содержащая ровно одну строку. По правилам SQL результат строчного подзапроса неявно преобразуется в строчное значение единственной строки таблицы RS. Тип этого значения и типы его элементов определяются списком выборки строчного подзапроса. При этом, если мощность RS оказывается больше 1, возникает исключительная ситуация, а если мощность RS равна 0, то выдается строка той же степени, что и степень RS, каждым элементом которой является NULL. По-видимому, так же ведет себя операция *ELEMENT*, извлекающая ровно одно строчное значение из заданного мультимножества мощности 1 с элементами строчного типа.

Опять же, эти правила SQL можно понять, поскольку строчные подзапросы могут участвовать в операциях сравнения. Даже неявное отождествление NULL и строки, составленной из NULL, можно (хотя и с трудом) объяснить универсальностью null-значения в SQL. Но в предлагаемой нами схеме совершенно недопустимо считать, что результатом пустого строчного подзапроса является строка, составленная из специальных значений типов элементов строчного типа. Во-первых, такая строка могла бы появиться и в том случае, когда мощность RS равна 1. Во-вторых, строка, составленная из специальных значений типов элементов строчного типа, является обычным значением этого строчного типа, у которого имеется собственное специальное значение.

Мы видим два возможных выхода из этого положения. Самым простым решением является генерация исключительной ситуации в случаях, года мощность SS, RS или мультимножества, к которому применяется операция *ELEMENT*, оказывается не равной 1. Другим вариантом является введение предиката IS [NOT] EMPTY, применяемого к выражению exp типа мультимножества (которое, возможно, включает подзапросы) и выдающего *true (false)* в том и только в том случае, когда результат вычисления exp пуст (непуст). Конечно, такую операцию можно было бы использовать при построении выражений с переключателем, а также можно было бы ввести сокращенную форму, аналогичную *COALESCE* в SQL:

$$COALESCE\_ON\_EMPTY\ (exp, v)$$
$$\equiv CASE\ WHEN\ exp\ IS\ NOT\ EMPTY\ THEN\ ELEMENT(exp)\ ELSE\ v\ END.$$

7) Что касается данных, отсутствующих по причине их неприменимости, мы предлагаем использовать для работы с ними упрощенную схему декомпозиции отношений Дарвена. Упрощения происходят из-за того, что мы имеем дело только с одной причиной отсутствия данных, и трудно представить, что в одной таблице во многих столбцах могли бы отсутствовать данные по причине их неприменимости в некоторых строках (скорее всего, это свидетельствовало бы о наличии очень плохой схемы базы данных[21]).

   В этом случае мы прибегаем к схеме декомпозиции, а не вводим еще одну разновидность специальных значений по нескольким причинам. Во-первых, схема декомпозиции

[21] Для большей аккуратности стоит уточнить, что нам неизвестны ситуации, в которых нельзя было бы избежать на стадии проектирования базы данных появления таблиц с большим числом столбцов с отсутствующими данными по причине их неприменимости.

позволяет не сохранять явно информацию об отсутствии значения. Во-вторых, семантика вычислительных операций и в особенности операций сравнения по меньшей мере неочевидна. Наконец, у четвертого логического значения, которое неизбежно появилось бы при введении второй разновидности специальных значений, отсутствует интуитивно понятный смысл, не говоря уже о наличии отрицательного опыта Кодда с использованием четырехзначной логики (см. п. 3.1.1).

Предлагаемая упрощенная схема декомпозиции выглядит следующим образом. Пусть имеется исходная таблица $r\{key, a1, \dots, an, b1, \dots bk\}$, где key – первичный или возможный ключ, в столбцах $a1, \dots, an$ не могут отсутствовать данные по причине их неприменимости, в столбцах $b1, \dots bk$ данные могут отсутствовать. Тогда, если $k = 1$, то r декомпозируется на две таблицы $rab\{ key, a1, \dots, an, b1\}$ и $ra\{ key, a1, \dots, an\}$, в которых rab содержит все допустимые строки с присутствующим значением столбца $b1$, а ra – те строки, в которых значение столбца $b1$ отсутствует.

Если $k = 2$, то r декомпозируется на четыре таблицы $rab1b2\{ key, a1, \dots, an, b1, b2\}$, $rab1\{ key, a1, \dots, an, b1\}$, $rab2\{ key, a1, \dots, an, b2\}$ и $ra\{ key, a1, \dots, an\}$, где $rab1b2$ содержит все допустимые строки с присутствующими значениями столбцов $b1$ и $b2$, $rab1$ – все строки, в которых значения $b1$ присутствуют, а значения $b2$ отсутствуют, $rab2$ – все строки, в которых значения $b2$ присутствуют, а значения $b1$ отсутствуют и ra – все строки, в которых отсутствует значения и столбца $b1$, и столбца $b2$. Для произвольного k появятся $C_k^0+C_k^1+\dots+C_k^{k-1}+C_k^k = 2^k$ таблиц, что, конечно, недопустимо для больших k.

Если пользователям нужно формулировать запросы к таблице, которая включает все значения ключа исходной таблицы r, можно определить представление (*VIEW*), в котором (для $k = 2$, например) объединяются (*UNION*) таблицы $rab1b2, rab1$ с добавленным столбцом $b2$, заполненным во всех строках каким-либо реальным значением, которое уместно в данном приложении, $rab2$ с добавлением и заполнением столбца $b1$ и ra с добавленными столбцами $b1$ и $b2$ заполненными во всех строках теми же значениями, что в $rab2$ и $rab1$ соответственно.

4. Заключение

Современное состояния средств поддержки отсутствующей информации в реляционных базах данных оставляет желать лучшего. Следуя предписаниям основателя реляционной модели данных Эдгара Кодд, разработчики стандарта языка SQL затвердили в нем механизм универсального null-значения, приводящий к многочисленным противоречащим интуиции ситуациям при формулировке и выполнении запросов, и вынуждающий использовать странный трехзначный булевский тип данных, в котором нетипизированное null-значение отождествляется с третьим значением булевского типа.

В течение многих лет альтернативные подходы к управлению отсутствующей информации разрабатывались и предлагались Кристофером Дейтом и Хью Дарвеном. В этих подходах для представления отсутствующих данных использовались сначала выделенные пользователями значения типов данных, а затем расширяющие типы данных специальные значения на основе традиционной двухзначной логики. Фактически, при следовании этим подходам вся нагрузка по управлению отсутствующими данными перекладывается на пользователей.

Тщательно проанализировав всю историю формирования и развития этих подходов, мы пришли к выводу, что ситуация, сложившаяся в стандарте SQL, является прямым следствием универсальности null-значения. Ограниченность же подходов Дейта и Дарвена неизбежна при использовании двухзначной логики. При этом и подход, идущий от Кодда, и схема специальных значений Дейта и Дарвена обладают и очевидными достоинствами: первый подход облегчает формулировку запросов, второй позволяет избавиться от универсального null-значения.

В предложенном нами подходе механизм специальных значений используется в комбинации с трехзначной логикой. На основе специальных значений мы обеспечиваем управление отсутствующими данными только одной категории – данные неизвестны. За счет введения специального значения в булевский тип, мы получаем полноценную трехзначную логику, в которой специальное значение играет роль третьего логического значения. Типизация специальных значений (отказ от универсального null-значения) позволяет учитывать при вычислении выражений особенности семантики типов данных и избегать ошибок, которые могут возникать в SQL-запросах. Для поддержки отсутствующих данных категории «данные неприменимы» мы предлагаем использовать упрощенный вариант схемы Дарвена на основе декомпозиции таблиц.

По нашему мнению, применение типизированных специальных значений вместо универсального null-значения позволило бы улучшить стандарт SQL. Однако мы отдаем себе отчет в том, что это очень большая работа, требующая пересмотра многих спецификаций. Имеется и техническая проблема, связанная с тем, что большинство типов данных в SQL являются безымянными, так что не очень понятно, как обозначать литералы специальных значений. Возможно, эту проблему можно было бы решить путем переосмысления понятия домена SQL таким образом, чтобы с помощью этого механизма можно было определять именованные типы данных со специальным значением.

Список литературы / References

[1] Codd E.F. Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks. IBM Research Report RJ599 (# 12343), 1969. Reprinted at ACM SIGMOD Record, 2009, Vol. 38, No. 1, 2009, pp. 17-36. Имеется перевод на русский язык: Э.Ф. Кодд. Выводимость, избыточность и согласованность отношений, хранимых в крупных банках данных. URL: http://citforum.ru/database/classics/first_rel_paper/.

[2] Codd E.F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, Volume 13, Number 6, 1970, pp. 377-387. Имеется перевод на русский язык: Е.Ф. Кодд. Реляционная модель данных для больших совместно используемых банков данных. URL: http://citforum.ru/database/classics/codd/.

[3] Codd E.F. Understanding Relations (Installment #7). Bulletin of ACM-SIGMOD: The Special Interest Group on Management of Data, vol. 7, no. 3-4, 1975, pp. 23-28.

[4] Codd E.F. Implementation of Relational Data Base Management Systems (NCC 1975 Panel). Bulletin of ACM-SIGMOD: The Special Interest Group on Management of Data, vol. 7, no. 3-4, 1975, pp. 3-22.

[5] Chamberlin D.D., Astrahan M.M. et al. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. IBM Journal of Research and Developoment, V. 20, No. 6, 1976, pp. 560-575. Имеется перевод на русский язык: Д.Д. Чамберлин, М.М. Астрахан, К.П. Эсваран, П.П. Грифитс, Р.А. Лори, Д.В. Мел, П. Райшер, Б.В. Вейд. SEQUEL 2: унифицированный подход к определению, манипулированию и контролю данных. URL: http://citforum.ru/database/classics/sequel_2/.

[6] Codd E.F. Extending the Database Relational Model to Capture More Meaning. ACM Transactions on Database Systems, vol. 4, issue 4, 1979, pp. 397-434. Имеется перевод на русский язык: Э.Ф. Кодд. Расширение реляционной модели для лучшего отражения семантики. URL: http://citforum.ru/database/classics/codd_2/.

[7] ISO/IEC 9075-2:1999. Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).

[8] Codd E.F. The Relational Model for Database Management: Version 2. Addison-Wesley, 1990, 538 p.

[9] Date C.J. The Default Values Approach to Missing Information. In C.J. Date (with Hugh Darven). Relational Database: Selected Writings 1989-1991. Addison-Wesley, 1992, pp. 343-354.

[10] Date C.J. Faults and Defaults (in five parts). In C. J. Date (with Hugh Darven and David McGoveran). Relational Database: Selected Writings 1994-1997. Addison-Wesley, 1998, 608 p.

[11] Date C.J. Hugh Darwen. Databases, Types and the Relational Model: The Third Manifesto. 3rd Edition. Addison-Wesley, 2006, 556 p.

[12] Codd E.F. Missing information (applicable and inapplicable) in relational databases. ACM SIGMOD Record, vol. 15, issue 4, 1986, pp. 53-78.

[13] Codd E.F. More commentary on missing information in relational databases (applicable and inapplicable information). ACM SIGMOD Record, vol. 16, issue 1, 1987, pp. 42-50.

[14] ANSI X3.135-1986. Information Technology – Database Languages – SQL.

[15] ISO 9075:1987. Information processing systems — Database language — SQL.

[16] ISO/IEC 9075:1989. Information processing systems — Database Language SQL with integrity enhancement.

[17] ISO/IEC 9075:1992. Information technology — Database languages — SQL.

[18] ISO/IEC 9075-1:1999. Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).

[19] ISO/IEC 9075-2:1999. Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).

[20] ISO/IEC 9075-2:2003. Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).

[21] ISO/IEC 9075-2:2008. Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)

[22] ISO/IEC 9075-2:2011. Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)

[23] ISO/IEC 9075-2:2016. Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)

[24] Date C.J. A critique of the SQL database language. ACM SIGMOD Record, vol. 14, issue 30, 1984, pp 8-54.

[25] Date C.J. Introduction to Database Systems, volume 2. Addison-Wesley, 1983, 383 p.

[26] Date C.J. Null Values in Database Management. In C.J. Date. Relational Database: Selected Writings. Addison-Wesley, 1986, pp. 313-334.

[27] Date C.J. NOT is Not “Not”! (Notes on Three-Valued Logic and Related Matters). In C.J. Date with a Special Contribution by Andrew Warden. Relational Database Writings 1985-1989. Addison-Wesley, 1990, pp. 217-248.

[28] Date C.J., Darwen H. Foundation for Future Database Systems: The Third Manifesto. 2nd Edition. Addison-Wesley, 2000, 608 p.

[29] Date C.J., Darwen H. Database Explorations: Essays on The Third Manifesto and Related Topics. Trafford Publishing, 2010, 548 p.

[30] Darwen H. How to Handle Missing Information without Using NULL. Presentation Slides. Available at: https://www.dcs.warwick.ac.uk/~hugh/TTM/Missing-info-without-nulls.pdf, accessed 11.05.2023.

[31] Darwen H., Smout E. How to Handle Missing Information Using S-by-C. Available at: https://www.dcs.warwick.ac.uk/~hugh/TTM/HTHMIUS-by-C-review-draft.pdf, accessed 11.05.2023.

[32] Gessert G.H. Four Valued Logic for Relational Database Systems. ACM SIGMOD Record, vol. 19, issue 10, 1990, pp 29-35

[33] Date C.J. Why Three- and Four-Valued Logic Don’t Work. In C.J. Date. Date on Database. Writings 2000–2006, Apress, 2012, pp. 329-342.

Информация об авторе / Information about the author

Сергей Дмитриевич КУЗНЕЦОВ – доктор технических наук, профессор, главный научный сотрудник ИСП РАН, профессор кафедр системного программирования МГУ, МФТИ и ВШЭ. Научные интересы: управление данными, архитектуры систем управления данными, модели и языки данных, управление транзакциями, оптимизация запросов.

Sergey Dmitrievich KUZNETSOV – Doctor of Technical Sciences, Professor, Chief Researcher at ISP RAS, Professor at the Departments of System Programming of MSU, MIPT, and HSE. Research interests: data management, architectures of data management systems, data models and languages, transaction management, query optimization.