Решение проблемы NULL в запросах к реляционной базе данных, используя операторы реляционной алгебры A.

И.В. Блудов. ivan.bludov@gmail.com

Аннотация. В операторах ограничения предлагается логические выражения интерпретировать как реляционные. Точнее, считается, что операция реляционного ограничения (*R WHERE b*) над отношением *R* по некоторому логическому выражению *b* может быть представлена как соединение (*R<AND>B*) заданного отношения R с реляционным выражением *B*, полученным из исходного логического выражения *b* заменой логических операторов AND, OR и NOT на соответствующие реляционные операторы <AND>, <OR> и <NOT>. Тогда для некоторого кортежа *T* определим значение атрибута A как отношение с одним кортежем и одним значением интересующего нас атрибута – RELATION<A>{{a}}. Значение атрибута, указанное как NULL, в качестве значение «неизвестно», определим как отношение с заголовком из интересующего нас атрибута и телом, содержащим всевозможные значения типа атрибута *A* – RELATION<A>{…}. Сравнение значений атрибутов на равенство будет выглядеть как соединение таких значений атрибутов, представленных отношениями. Кортеж *T*, который может быть определен как декартовое произведение всех своих атрибутов, будет теперь представлять отношение R_T. Истинность такого кортежа *T*, представленного отношением R_T, по заданному логическому выражению *b*, означает истинность квантора всеобщности над значениями R_T по выражению *b*, что в свою очередь означает равенство соединения (R_T <AND> *B*) и R_T – (R_T <AND> *B*) = R_T.

Ключевые слова: реляционная модель данных; NULL; отсутствующая информация; реляционная алгебра A.

1. Введение

В основу данной статьи легла извечная проблема несостоятельности трехзначной логики, используемой в SQL для поддержки механизма работы с отсутствующей информацией. Кристофер Дейт в работе [1]

показывает невозможность использования n-значных логиках. В частности, он критикует предложенные Тэдом Коддом таблицы истинности элементарных логических операторов трехзначной логики, используемых в SQL, а также другие попытки Кодда введения таблиц истинности элементарных логических операторов для четырехзначной логики. Основная причина такой несостоятельности n-значных логиках состоит в том, что множества тождеств и противоречий (выражения, вычисляемых как *истинна* или *ложь* при любых значениях операнд), используемых в разных логиках, различны. В частности, оптимизаторы SQL-запросов трансформируют выражения, используя некорректное, с точки зрения двухзначной логики (реального мира), множество тождеств и противоречий. В результате чего такие трансформированные реляционные выражения возвращают некорректные, с точки зрения реального мира, результаты.

В качестве одного примера несостоятельности трехзначной логики в SQL Дейт приводит в [2],[3] следующий запрос. Для базы данных

Поставщики S

SNO*	S_CITY
S1	‘London’

Детали P

PNO*	P_CITY
P1	NULL

«получить пары SNO-PNO, для которых города поставщика и детали различны или городом детали не является Париж (либо оба эти условия)». Предлогается следующая «очевидная формулировка этого запроса на языке SQL»: SELECT S.SNO, P.PNO FROM S, P WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris'. В результате даже возник спор между Клодом Рубинсоном [4], с одной стороны, и Дейтом[5] и Джоном Грантом[6], с другой.

Чтобы подчеркнуть «вечность» описанной проблеме, можно повторить мысль Дейта о том, что время от времени люди из сообщества баз данных для решения описанной проблемы пытаются придумать какую-нибудь k-значную логику и определить для нее таблицы истинности основных операторов, попадая при этом в распространённую ловушку. Учитывая это, то есть оставаясь в рамках двухзначной логики, при исследовании проблемы отсутствующей информации я начинаю с понятия истинности логического выражение и того, чем, по сути, является отношение – множеством истинных утверждений [2], [3]. Высказывается идея о том, логические выражения можно рассматривать как подмножество другого класса выражений – реляционных выражений. В то время, как логические выражения оперируют одиночными утверждениями, реляционные выражения оперируют множествами таких утверждений. Используются операторы реляционной алгебры A [7], которые, по сути, являются логическими операторами над множеством истинных утверждений, то есть отношениями. В большей части статьи разбирается пример, который показывает, как можно в операторе реляционного ограничения (*R WHERE b*) логическое выражение *b* представить реляционным выражением, заменяя логические операторы OR, AND, NOT на соответствующие реляционные операторы алгебры A <OR>, <AND>, <NOT>. Далее известное значение атрибута A определяется, как отношение с одним кортежем и интересующим значением (RELATION<A>{{a}}); NULL в качестве значение неизвестно определяется, как отношение, содержащие все возможные значения атрибута *A* (RELATION<A>{…}); кортеж – как декартовое произведение всех своих атрибутов. Это позволяет получить устойчивое решение проблемы отсутствующей информации.

3. Представление логического выражения в операции ограничения как реляционного выражения

3.1 Рассмотрим операцию ограничения над отношением REL WHERE $P_{(REL)}$, где $P_{(REL)}$ – некоторое логическое выражение над атрибутами отношения REL. Будем рассматривать только некоторое подмножество из всех возможных логических выражений: пусть $P_{(REL)}$ определяется следующим образом:

<логическое_выражение>::= NOT <логическое_выражение>

<логическое_выражение> AND <логическое_выражение> |

<логическое_выражение> OR <логическое_выражение> |

<логическое_выражение> = <логическое_выражение> |

<логическое_выражение> <> <логическое_выражение> |

<сравнение_атрибута_и_константы> | <сравнение_двух_атрибутов> |

<логическая_константа>

Отметим, что можно определить операторы сравнения логических выражений:

A = B как *(A and B) or (not (A) and not (B))*;

A <> B как *(not(A) and B) or (A and not (B))*.

Тогда определение $P_{(REL)}$ можно сведено до следующего:

<логическое_выражение>::= NOT <логическое_выражение> |

<логическое_выражение> AND <логическое_выражение> |

(1) <логическое_выражение> OR <логическое_выражение> |

<сравнение_атрибута_и_константы>|<сравнение_двух_атрибутов>|

<логическая_константа>

3.2 Запишем некоторые общеизвестные правила преобразований в терминах новой реляционной алгебры A:

- *X WHERE (a OR b) =*

  (X WHERE *a*) UNION (X WHERE *b*) =

  (X WHERE a) <OR> (X WHERE b)
- *X WHERE (a AND b) =*

  (X WHERE *a*) INTERSECT (X WHERE *b*) =

  (X WHERE a) <AND> (X WHERE b)

Известно, что любое выражение вида (X WHERE *b*) может быть представлено, как соединение отношения *X* с некоторым другим отношением *B* таким, что заголовки (типы) отношений *X* и *B* одинаковые, а кортежи отношения *B* представляет все возможные истинные значения логического выражения *b*.

X WHERE b <=> X <AND> B.

- Закон Де Моргана:
  - для логических выражений:

    NOT (*a* AND *b*) = NOT (*a*) OR NOT (*b*);

    NOT (*a* OR *b*) = NOT (*a*) AND NOT (*b*);
  - для реляционных выражений:

    <NOT> (A <AND> B) = (<NOT> A) <OR> (<NOT>B);

    <NOT> (A <OR> B) = (<NOT> A) <AND> (<NOT>B);

- Свойство дистрибутивности:
  - для логических выражений:

    (*a* AND *b*) OR *c* = (*a* OR *c*) AND (*b* OR *c*);

    (*a* OR *b*) AND *c* = (*a* AND *c*) OR (*b* AND *c*);
  - для реляционных выражений:

    (A <AND> B) <OR> C = (A <OR> C) <AND> (B <OR> C);

    (A <OR> B) <AND> C = (A <AND> C) <OR> (B <AND> C);
- *X WHERE (NOT b)* = X MINUS (X WHERE *b*) = X <AND> <NOT>(X WHERE *b*) = X <AND> <NOT> (X <AND> B) = X <AND> ((<NOT>X) <OR> (<NOT>B)) = (X <AND> <NOT>X) <OR> (X <AND> <NOT>B) = *X <AND> <NOT> B*, поскольку (X <AND> <NOT> X) – пустое отношение.

3.3 Используя перечисленные выше правила (главным образом, правила для *X WHERE (a OR b); X WHERE (a AND b); X WHERE (NOT b)*) для преобразования исходного выражения *REL WHERE P(*$_{REL}$*)*, мы получим реляционное выражение с соединениями, пересечениями и отрицаниями простых ограничений вида *REL WHERE P(Atr_i)*, *REL WHERE P(Atr_i, Atr_j)*, *REL WHERE P()*, то есть с простыми логическими выражениями вида:

<логическое_выражение>::= <сравнение_атрибута_и_константы> |

<сравнение_двух_атрибутов> | <логическая_константа>

Приведем пример:

X WHERE (NOT((A=B)=(B=b)) AND

(((C=c) AND FALSE) OR TRUE)) **(2)**

=

X <AND>

<NOT> (

(X WHERE (A=B) <AND> X WHERE (B=b)) <OR> **(3)**

(X <AND> <NOT> (X WHERE (A=B)) <AND> <NOT> (X WHERE (B=b)))

)

<AND> ((X WHERE (C=c) <AND> X WHERE FALSE) <OR> X WHERE TRUE)

3.4 Далее простые ограничения вида REL WHERE P(Atr_i) будем рассматривать как соединение REL<AND>PAtr_i, где кортежи отношения PAtr_i представляет все возможные истинные значения логического выражения P(Atr_i):

- Сравнение атрибута и константы: (X WHERE A<cmp_op> const) = (X <AND> CONST)
- Сравнение двух атрибутов: (X WHERE A<cmp_op>B) = (X <AND> AB)
- Логическая константа:

  (X WHERE TRUE) = (X <AND> TABLE_DEE) и

  (X WHERE FALSE) = (X <AND> TABLE_DUM)

Тогда реляционное выражение (3) преобразуется к следующему виду:

X <AND>

<NOT>(

(X <AND> AB <AND> X <AND> B) <OR> **(4)**

(X <AND> <NOT> (X <AND> AB) <AND> <NOT> (X <AND> B))

)

<AND> ((X <AND> C <AND> X <AND> TABLE_DUM)

<OR> X <AND> TABLE_DEE)

3.5 Теперь, когда у исходное отношение *REL WHERE P(*$_{REL}$*)* представлено в виде (4), «вытащим» (вынесем за скобки) конъюнкт REL. То есть представим выражение (4) в виде соединения REL с выражением, не содержащим отношение REL. Отметим, что это возможно из-за дистрибутивности операторов <AND> и <OR>, а также из-за того, что реляционный оператор <NOT> (используемый в выражение *X <AND> (<NOT> EXP)*) хоть и возвращает дизъюнкцию конъюнктов, содержащих как X, так и <NOT> X, конъюнкты с <NOT> X сократятся, поскольку им всегда будет предшествовать соединение с X.

Тогда реляционное выражение (4) преобразуется к следующему виду:
X <AND>

<NOT>(AB <AND> B <OR> <NOT> AB <AND> <NOT> B) **(5)**

<AND> (C <AND> TABLE_DUM <OR> TABLE_DEE))

Если в исходном ограничении (2) представить равенство

(A=B) = (B=b) как (((A=B) AND (B=b)) OR (NOT (A=B) AND

NOT (B=b))),

то получим выражение

X WHERE (NOT (((A=B) AND (B=b)) OR (NOT (A=B) AND NOT (B=b)))

AND (((C=c) AND FALSE) OR TRUE)) (6),

которое очень схоже с выражением (5):

X WHERE NOT (((A=B) AND (B=b)) OR (NOT (A=B) AND NOT (B=b))) AND ((C=c) AND FALSE OR TRUE)	X <AND> <NOT>(AB <AND> B <OR> <NOT> AB <AND> <NOT> B) <AND> (C <AND> TABLE_DUM <OR> TABLE_DEE))

На схожести выражений (5) и (6), полученных в результате приведенных выше преобразований, основана одна из главных идей данной статьи о ***возможности интерпретации логического выражения как реляционного выражения***. То есть оператор реляционного ограничения *(X WHERE b)* с некоторым логическим выражением *b* вида (1) может быть представлен как соединение *X* и некоторого реляционного выражения R_b – *X <AND>* R_b, где реляционное выражение R_b получено из логического выражения *b* заменой логических операторов NOT, AND, OR на соответствующие реляционные операторы <NOT>, <AND>, <OR>; сравнения атрибутов и значений/атрибутов на отношения, представляющих данные сравнения на множестве всевозможных значений; логических констант TRUE и FALSE на отношения-константы TABLE_DEE и TABLE_DUM соответственно. Кроме того, поскольку в

реляционной алгебры A любое ограничение *(X WHERE b)* может быть представлено как соединение *X* с некоторым отношением *B*, кортежи которого представляет все возможные истинные значения логического выражения *b*, по построению *B* является результатом вычисления R_b. Таким образом, R_b является функций ограничения над всеми возможными значениями *X*, а *X <AND>* R_b является вызовом этой функции для заданного значения отношения *X*.

3.6 Вернемся к выражению (3) и пункту, для которого выполним подстановку следующего вида:

- Сравнение атрибута и константы:

(X WHERE A<cmp_op>const) = (X <AND> X{A} <AND> CONST)

- Сравнение двух атрибутов:

(X WHERE A<cmp_op>B) = (X <AND> X{A,B} <AND> AB)

- Логическая константа:

(X WHERE TRUE) = (X <AND> TABLE_DEE) и

(X WHERE FALSE) = (X <AND> TABLE_DUM)

Получим:

X <AND>

<NOT>(

(X WHERE (A=B) <AND> X WHERE (B=b)) <OR>

(X <AND> <NOT> (X WHERE (A=B)) <AND> <NOT> (X WHERE (B=b)))

)

<AND> ((X WHERE (C=c) <AND> X WHERE FALSE) <OR> X WHERE TRUE)

=

X <AND>

<NOT>(

(X <AND> X{A,B} <AND> AB <AND> X <AND> X{B}<AND> B) <OR>

(X <AND> <NOT> (X <AND> X{A,B} <AND> AB) <AND> <NOT> (X <AND> X{B} <AND> B))

)

<AND> ((X <AND> C <AND> X <AND> TABLE_DUM) <OR> X <AND> TABLE_DEE)

Над полученным выражением вновь «вытащим» конъюнкт *X*:

X <AND>

<NOT>(

(X <AND> X{A,B} <AND> AB <AND> X <AND> X{B}<AND> B) <OR>

(X <AND> <NOT> (X <AND> X{A,B} <AND> AB) <AND> <NOT> (X <AND> X{B} <AND> B))

)

<AND> ((X <AND> C <AND> X <AND> TABLE_DUM) <OR> X <AND> TABLE_DEE)

=

X <AND>

<NOT>(

((X{A,B} <AND> AB)<AND>(X{B}<AND> B)) <OR>

(<NOT> (X{A,B} <AND> AB) <AND> <NOT> (X{B} <AND> B)) **(7)**

)

<AND> (C <AND> TABLE_DUM <OR> TABLE_DEE)

Таким образом, выражение (7) в сравнении с выражение (5), отчасти уже содержит вызов функции ограничения. То есть если

<NOT> (AB <AND> B <OR> <NOT> AB <AND> <NOT> B)

<AND> (C <AND> TABLE_DUM <OR> TABLE_DEE)) -

функция ограничения в общем виде, тогда

<NOT>(

((X{A,B} <AND> AB)<AND>(X{B}<AND> B)) <OR>

(<NOT> (X{A,B} <AND> AB) <AND> <NOT> (X{B} <AND> B))

)

<AND> (C <AND> TABLE_DUM <OR> TABLE_DEE)

уже отчасти содержит вызов общей функции ограничения с интересующими нас значениями, и может быть рассмотрен как частный случай функции ограничения.

3.7 Вернемся к рассмотрению операцию ограничения *REL WHERE P(*$_{REL}$*)*. Отношение REL может быть представлено как объединение множества всех своих кортежей, точнее как объединение отношений, содержащих только один кортеж из REL, для каждого кортежа из REL.

(UNION(R_{Tuple1},R_{Tuple2},…,R_{Tuple_N}))WHERE P($_{REL}$)	(<OR>(R_{Tuple1},R_{Tuple2},…,R_{Tuple_N})) <AND> P_{REL}

Используя свойство дистрибутивности операций ограничения и объединения, получим

UNION(	<OR>(
R_{Tuple1} WHERE P($_{REL}$),	R_{Tuple1} <AND> P_{REL},
R_{Tuple2} WHERE P($_{REL}$),	R_{Tuple2} <AND> P_{REL},
…,	…,
R_{Tuple_N} WHERE P($_{REL}$))	R_{Tuple_N} <AND> P_{REL})

Очевидно, что ограничение *R_{Tuple_K} WHERE P(*$_{REL}$*)* (*R_{Tuple_K} <AND> P_{REL}*) возвращаете либо пустое отношение, если условие ложно; *либо входное отношение, если условие истинно*. (*Примечание*: в дальнейшем будем называть отношения вида *R_{Tuple_K} – отношениями-кортежами*)

3.8 Теперь применим ограничения из пункта **3.6**, где сравнения атрибут представлены как

- Сравнение атрибута и константы:

(X WHERE A<cmp_op>const) = (X <AND> X{A} <AND> CONST)

- Сравнение двух атрибут:

(X WHERE A<cmp_op>B) = (X <AND> X{A,B} <AND> AB)

индивидуально к отношениям-кортежам X_{Tuple_K} (R_{Tuple_K} из пункта **3.7**).

Таким образом, проекции вида X_{Tuple_K}{A} дадут отношение с одним кортежем, содержащим одно значение для атрибута A –

RELATION<A>{{A a}} (*примечание*: в дальнейшем будем называть такие отношения – *отношениями-значениями*). Проекции X_{Tuple_K}{A, B}, поскольку X_{Tuple_K} содержит только один кортеж, можно рассматривать как соединение таких отношений-значений X_{Tuple_K}{A} <AND> X_{Tuple_K}{B}.

Теперь сравнения атрибутов будут выглядеть следующим образом.

- Сравнение атрибута и константы:

X_{Tuple_K} WHERE A<cmp_op>const = X_{Tuple_K} <AND> X_{Tuple_K}{A} <AND> CONST

- Сравнение двух атрибут:

X_{Tuple_K} WHERE A<cmp_op>B = X_{Tuple_K} <AND> X_{Tuple_K}{A} <AND> AB <AND> X_{Tuple_K}{B}

Если вернуться от абстрактных отношений A и AB, содержащих всевозможные значения, используемых в определения сравнений атрибутов, тогда сравнение несут смысл эквисоединения над входными отношениями-значениями (опишем с привлечением терминов SQL):

- Сравнение атрибута и константы:

X_{Tuple_K} WHERE A<cmp_op>const =

X_{Tuple_K} <AND> (X_{Tuple_K}{A} JOIN RELATION<CONST>{{const}} ON (A<cmp_op>CONST)){A}

- Сравнение двух атрибут:

X_{Tuple_K} WHERE A<cmp_op>B =

X_{Tuple_K} <AND> (X_{Tuple_K}{A} JOIN X_{Tuple_K}{B} ON (A<cmp_op>B))

Конечно, иногда может быть удобно для ограничения *X_{Tuple_K} WHERE A<cmp_op>B* использовать отрицание от обратного оператора сравнения *X_{Tuple_K} WHERE NOT(A<inverse_cmp_op>B)*. Который можно упростить:

X_{Tuple_K} WHERE A<cmp_op>B =

X_{Tuple_K} WHERE NOT (A<inverse_cmp_op>B) =

X_{Tuple_K} <AND> <NOT> (X_{Tuple_K} {A} <AND> <INVERSE_AB> <AND> X_{Tuple_K} {B})

3.9 Все описанные выше утверждения верны для реляционной модели без NULL. Теперь же введем NULL и покажем операции сравнения над ними.

Основная идея состоит в том, чтобы рассматривать NULL для некоторого атрибута $A_{UNKNOWN}$, в случае значение не известно (значение не определено, значение не достоверно, значение не предоставлено), как отношение с заголовком, содержащим только интересующий нас атрибут $A_{UNKNOWN}$, и телом, содержащим всевозможными значениями для заданного типа атрибута $A_{UNKNOWN}$. Более точно, нас в большей мере интересует проекция X_{Tuple_K} { $A_{UNKNOWN}$ } из пункта **3.8**, то есть результатом такой проекции будет отношение с заголовком из атрибута $A_{UNKNOWN}$, и телом со всевозможными значениями для типа атрибута $A_{UNKNOWN}$.

Что касается NULL для атрибута A_{NA}, в качестве значение недопустимо (значение не существует, значением является пустое множество), изначально планировалось определить его как пустое отношение с заголовком из атрибута A_{NA}, однако, как оказалось такое поведение не всегда верно, и даже может быть упрощено. Рассмотрим его чуть позже.

3.10 С введением NULL, в случае значение не известно, отношение X_{Tuple_K}, изначально содержавший только один кортеж, теперь же содержит множество кортежей, которое получилось как декартовое произведение своих атрибутов. Ограничение *X_{Tuple_K} WHERE $P_{(REL)}$*, как говорилось в пункте **3.7** «*возвращаете входное отношение, если условие истинно*», принимает смысл истинности квантора всеобщности для отношения *X_{Tuple_K}* над выражением *$P_{(REL)}$*.То есть если результат *X_{Tuple_K} WHERE $P_{(REL)}$* равен исходному отношению *X_{Tuple_K}*, это означает истинность квантора всеобщности, и потому кортеж *Tuple_K* из отношения REL (см. пункт VII), добавляется в результат ограничение *REL WHERE $P_{(REL)}$*.

4. Примеры

Рассмотрим приведенный во введении пример Дейта, показывающий несостоятельность трехзначной логики в SQL.

Поставщики и детали

Поставщики S

SNO*	S_CITY
S1	‘London’

Детали P

PNO*	P_CITY
P1	NULL

Запрос

SELECT S.SNO, P.PNO FROM S, P

WHERE S.CITY <> P.CITY OR P.CITY <> ’Paris’

Представим соединение кортежей S и P как отношение TUPLE_S_JOIN_P, и далее будем использовать операторы реляционной алгебры, а не SQL.

TUPLE_S_JOIN_P WHERE (S_CITY<>P_CITY) OR (P_CITY<>’Paris’)

=

TUPLE_S_JOIN_P <AND>

(<NOT> ((TUPLE_S_JOIN_P {S_CITY} RENAME S_CITY AS P_CITY)

<AND> TUPLE_S_JOIN_P {P_CITY})

<OR>

<NOT> (TUPLE_S_JOIN_P {P_CITY} <AND> RELATION< P_CITY>

{{‘Paris’}}))

(см. Преобразование 3)

Выполним подстановку значений P_CITY

4.1 Значение P_CITY определено

Отношение TUPLE_S_JOIN_P:

SNO	S_CITY	PNO	P_CITY
S1	‘London’	P1	‘Paris’

TUPLE_S_JOIN_P =

RELATION<SNO> {{S1}} <AND> RELATION<S_CITY> {{‘London’}} <AND>

REALTION {{P1}} <AND> RELATION<P_CITY>{{‘Paris’}}

TUPLE_S_JOIN_P WHERE (S_CITY<>P_CITY) OR (P_CITY<>’Paris’)

=

TUPLE_S_JOIN_P <AND>

(<NOT>(RELATION<P_CITY>{{‘London’}} <AND>

RELATION< P_CITY>{{‘Paris’}})

<OR>

<NOT>(RELATION<P_CITY>{{‘Paris’}} <AND>

RELATION<P_CITY>{{‘Paris’}}))

=

TUPLE_S_JOIN_P <AND>

(<NOT> RELATION<P_CITY>{} <OR>

<NOT>RELATION<P_CITY>{{‘Paris’}})

=

TUPLE_S_JOIN_P <AND>

(RELATION<P_CITY>{…} <OR> RELATION<P_CITY>{…/{‘Paris’}})

где RELATION< P_CITY>{…} означает отношение, содержащее всевозможные кортежи; а RELATION<P_CITY>{…/{‘Paris’}} означает отношение, содержащее всевозможные кортежи за исключением кортежа {‘Paris’}

=

TUPLE_S_JOIN_P <AND> RELATION<P_CITY>{…}

Очевидно, что рассмотренное ограничение вернет отношение TUPLE_S_JOIN_P, что будет означать истинность квантора всеобщности – то есть для всех кортежей отношения TUPLE_S_JOIN_P верно выражение (*S_CITY <> P_CITY OR P_CITY <> 'Paris'*)

4.2 Значение P_CITY неопределенно

Отношение TUPLE_S_JOIN_P:

SNO	S_CITY	PNO	P_CITY
S1	'London'	P1	UNKNOWN

TUPLE_S_JOIN_P =

RELATION<SNO> {{S1}} <AND> RELATION<S_CITY> {{'London'}} <AND>

REALTION {{P1}} <AND> RELATION<P_CITY>{…}

TUPLE_S_JOIN_P <AND>

(<NOT>(RELATION< P_CITY>{{'London' }} <AND>

RELATION<P_CITY>{…})

<OR>

<NOT>(RELATION<P_CITY>{…} <AND> RELATION<P_CITY>{{'Paris'}}))

=

TUPLE_S_JOIN_P <AND>

(<NOT> RELATION<P_CITY>{{'London'}} <OR>

<NOT>RELATION<P_CITY>{{'Paris'}})

=

TUPLE_S_JOIN_P <AND>

(RELATION<P_CITY>{…/'London'} <OR>

RELATION<P_CITY>{…/{'Paris'}})

=

RELATION<SNO, S_CITY, PNO, P_CITY>{{S1, 'London', P1, …}} <AND>

RELATION<P_CITY>{…}

Данное ограничение также вернет отношение TUPLE_S_JOIN_P, то есть истинность квантора всеобщности для значений RELATION<SNO, S_CITY, PNO, P_CITY>{{S1, 'London', P1, …}} над выражением (S_CITY <> P_CITY OR P_CITY <> 'Paris')

4.3 Значение P_CITY недопустимо

Отношение TUPLE_S_JOIN_P:

SNO	S_CITY	PNO	P_CITY
S1	'London'	P1	N/A

Как было упомянуто ранее в пункте **3.10**, изначально планировалось определить NULL, в качестве значение недопустимо, как пустое отношение, с заголовком соответствующего атрибута. Поскольку сравнение на равенство атрибутов представляется как соединение *отношений-значений* сравниваемых атрибутов, а соединение с пустым отношением всегда даст пустое отношение, что означало бы ложь. Однако сравнение значения двух атрибутов с недопустимым значением, тоже даст пустое отношение, хотя ожидалась истина. Кроме того, не очевидно как представлять TUPLE_S_JOIN_P. Поскольку TUPLE_S_JOIN_P = RELATION<SNO> {{S1}} <AND> RELATION<S_CITY> {{'London'}} <AND> REALTION {{P1}} <AND> RELATION<P_CITY>{}, естественно даст пустое отношение, а для пустого отношения квантор всеобщности всегда истинен, даже если условие ложно. Однако в действительности случай NULL, как значение недопустимо, более вырожденный, чем случай NULL, как значение неизвестно, поскольку возвращает истину, только если оба значения недопустимы. Потому результат его сравнение может быть просто представлен логическими константами, или же TABLE_DEE и TABLE_DUM, если мы работаем с отношениями. Остается открытым вопрос из пункта **3.10**, как определить добавлять ли кортеж *Tuple_K* из отношения *REL* в результат ограничение *REL WHERE P(REL)*, где равенство отношения X_{Tuple_K} и ограничения X_{Tuple_K} *WHERE P(REL)* означает истинность квантора всеобщности для отношения-кортежа X_{Tuple_K}. Истинность же квантора всеобщности в случае NULL, значение недопустимо, предлагается ввести как истинность равенство проекции *(*X_{Tuple_K} *ALL BUT (*A_{NA}*,…))* и ограничения *(*X_{Tuple_K} *ALL BUT (*A_{NA}*,…) WHERE P(REL))*.

Так результат (TUPLE_S_JOIN_P {S_CITY} RENAME S_CITY AS P_CITY) <AND> TUPLE_S_JOIN_P {P_CITY} определим как TABLE_DUM, результат (TUPLE_S_JOIN_P {P_CITY} <AND> RELATION< P_CITY> {{‘Paris’}}) также определим как TABLE_DUM. Тогда

(TUPLE_S_JOIN_P ALL BUT P_CITY) <AND>

(<NOT>TABLE_DEE <OR> <NOT>TABLE_DUM)

=

(TUPLE_S_JOIN_P ALL BUT P_CITY) <AND> TABLE_DEE

=

(TUPLE_S_JOIN_P ALL BUT P_CITY)

Очевидно что, квантор всеобщности истинен.

(см. Примеры)

5. Смежные вопросы

5.1 Сравнения атрибута на равенство больше/меньше

Рассмотрим вопросы сравнения атрибута и константы на равенство больше/меньше. Из пункта **3.8** операторы сравнения атрибута и константы определены следующим образом:

X_{Tuple_K} WHERE A<cmp_op>const = X_{Tuple_K} <AND> X_{Tuple_K}{A} <AND> CONST, или же

X_{Tuple_K} WHERE A<cmp_op>const =

X_{Tuple_K} <AND> (X_{Tuple_K}{A} JOIN RELATION<CONST>{{const}} ON (A<cmp_op>CONST)){A}

Пусть атрибут A имеет тип T, для которого определены операции сравнения на больше/меньше. По определению каждый тип определяет

множеством значений, принадлежащих данному типу. Операция ограничения в реляционной алгебре A предполагает существование отношения CONST, кортежами которого являются все значения T, удовлетворяющие условию *t<cmp>const*. Результатом соединение $X_{Tuple\_K}\{A\}$ *<AND> CONST*, поскольку $X_{Tuple\_K}\{A\}$ представляет отношение с всевозможными значениями над типом T, очевидно будет отношение *CONST*.

Запрос 5.1.1

Rel:

A REAL
UNKNOWN

Rel *WHERE* A>3 OR A<4 =

Rel <AND> ((Rel{A} <AND> $REAL_{A>3}$) <OR> (Rel{A} <AND> $REAL_{A<4}$))

где $\mathrm{REAL}_{A>3}$ означает отношение RELATION<A REAL>{A|A>3}=

Rel <AND> (RELATION<A>{…} <AND> $REAL_{A>3}$)<OR>(RELATION<A>{…}<AND>$REAL_{A<4}$)

=

Rel <AND> ($REAL_{A>3}$<OR> $REAL_{A<4}$)=

Rel <AND> REALITON<A > {…}

Квантор всеобщности истинен.

Запрос 5.1.2

TYPE REAL_1_TO_10 IS REAL CONSTRAINT R>=1 AND R<=10;

Rel:

A REAL_1_TO_10
UNKNOWN

Rel WHERE A>0 =

Rel <AND> Rel{A} <AND> $\text{REAL}_{A>0}$ =

Rel <AND> RELATION<A REAL_1_TO_10>{…} <AND> $\text{REAL}_{A>0}$ =

Rel <AND> RELATION<A REAL_1_TO_10>{…}

Квантор всеобщности истинен

5.2 Сравнение двух неопределённых значений

Рассмотрим сравнение двух неопределённых значений. Очевидно, что сравнения двух атрибутов, описанных в пункте 3.8 как:

X_{Tuple_K} WHERE A<cmp_op>B = X_{Tuple_K} <AND> X_{Tuple_K}{A} <AND> AB <AND> X_{Tuple_K}{B},

или же

X_{Tuple_K} WHERE A<cmp_op>B = X_{Tuple_K} <AND> (X_{Tuple_K}{A} JOIN X_{Tuple_K}{B} ON (A<cmp_op>B)),

В случае, когда атрибуты представлены неопределённым значением, проекции X_{Tuple_K}{A} и X_{Tuple_K}{B} представляют множество всевозможных значений, тогда возникает сложность в представлении отношения AB, не говоря о вычислительных сложностях. Поскольку нам к тому же следует проверять истинность квантора всеобщности: «каждый человек выше любого другого» – то предполагается, что такое сравнение неопределённых значений атрибутов будет ложью. То есть соединения: (*X_{Tuple_K}{A} <AND> AB <AND> X_{Tuple_K}{B}*) или же (*X_{Tuple_K}{A} JOIN X_{Tuple_K}{B} ON (A<cmp_op>B)*) – будут возвращать пустые отношения. Конечно, это не позволит показать истинность выражения: «Все либо выше других, либо ниже других, либо одинокого роста». Также следует помнить о типах атрибутов, поскольку неопределённые значения определяются как множество всевозможных значений над заданным типом: так очевидно, что какой-либо человек выше, скажем, к примеру, кого-либо муравья.

Также высказывается идея, что при добавлении неопределённого значения в базу данных, в системе автоматически вводится некий уникальный идентификатор для нового неопределённого значения. Тогда становится возможным сравнение неопределенного значения с самим собой, что дает истину. Более того, если такое соотношение

уникального идентификатора и неопределённого значения существует, тогда система базы данных сможет всегда отслеживать миграцию такого неопределенного значения, и выполнять корректное сравнивание их, даже если они находятся в разных отношениях.

6. Заключение

В данной статье я попытался доказать, что ограничении (*R WHERE B*) некоторого отношения R по логическому выражению b может быть записано как соединение R<AND>B, где B реляционное выражение, полеченное из логического выражения b заменой, логических операторов AND, OR, NOT на соответствующие операторы новой реляционной алгебры <AND>, <OR>, <NOT>, а сравнение атрибутов, представленных отношением, представляется эквисоединением. Тогда я определяю NULL, в качестве значение неизвестно, как отношение со всевозможными значениями над соответствующим типом атрибута. Кортежа T из R, содержащего неопределённое значение для некоторого атрибута, представляется как декартовое произведение своих атрибутов – R_T. Истинность такого кортежа T, представленного отношением R_T, по заданному логическому выражению b, означает истинность квантора всеобщности над значениями R_T по выражению b, что означает равенство соединения (R_T <AND> B) и R_T. Также я рассмотрел некоторые практические вопросы сравнения неопределённого значения атрибута, представленного отношением со всевозможными значениями.

7. Список литературы

[1] C. J. Date. “Why Three- and Four-Valued Logic Don’t Work” in “Date on Database. Writings 2000–2006”. Apress, 2006. Впервые опубликована на сайте http://www.dbdebunk.com (February 2006).

[2] C. J. Date. Database in Depth: Relational Theory for Practitioners. Sebastopol, Calif.: O’Reilly Media, Inc. (2005).

[3] C. J. Date. SQL and Relational Theory: How to Write Accurate SQL Code. O’Reilly Media, Inc. (2009).

[4] Claude Rubinson. Nulls, Three-Valued Logic, and Ambiguity in SQL : Critiquing Date’s Critique. SIGMOD Record, December 2007 (Vol. 36, No. 4). См. также перевод: http://citforum.ru/database/articles/nulls/

[5] C. J. Date. A Critique of Claude Rubinson’s Paper Nulls, Three - Valued Logic, and Ambiguity in SQL: Critiquing Date’s Critique. SIGMOD Record, Vol. 37, No. 3, September 2008. См. также перевод: http://citforum.ru/database/articles/date_vs_rubinson/

[6] John Grant. Null Values in SQL. SIGMOD Record, Vol. 37, No. 3, September 2008. См. также перевод: http://citforum.ru/database/articles/grant_vs_rubinson/
[7] C. J. Date, Hugh Darwen. “Foundation for Future Database Systems: The Third Manifesto”, Addison-Wesley Pub Co; 2nd edition (2000).
Имеется перевод: Дейт К., Дарвен Х. Основы будущих систем баз данных. Третий манифест. 2-е изд. (под ред. С. Д. Кузнецова). М.: Янус-К, 2004.