Особенности табличных выражений SQL и их соответствие с концепциями реляционной модели данных

И.В. Блудов. ivan.bludov@gmail.com

Аннотация. В данной статье изложены материалы сравнения и критики SQL по поводу соответствия операторов SQL реляционной модели данных. Рассмотрены особенности табличных выражений в стандарте SQL и показаны случаи, в которых эти выражения противоречат реляционной модели. Приведены варианты использования таких табличных выражений в SQL, которые наиболее точно отражают концепции реляционной модели.

Ключевые слова: SQL, реляционная модель данных, SQL запросы, табличные выражения

1. Введение

Общеизвестно, что SQL является стандартным языком реляционных баз данных, но этот факт не делает его реляционным. В действительности, он отдалился от реляционной модели достаточно давно. Следует понимать различия SQL и реляционной теории, чтобы избежать возможных проблем, связанных с этими различиями.

Данная статья посвящена исследованию различий SQL и реляционной модели данных. Более конкретно, в ней рассматривается один из важных разделов SQL – табличные выражения. В статье проводится анализ табличных выражений стандарта SQL, их особенностей и сравнение с операторами реляционной модели данных. На основе этого анализа выясняется, как такие выражения следует использовать, чтобы они соответствовали реляционной модели. С целью написания качественного SQL следует применять соответствующие практики, что позволит ощутить преимущества реляционной модели.

2. Определения кортежа и отношения

Пусть *T1,T2,..,Tn* (n≥0) - определённые типы, не обязательно различные. Пусть *Ai* - отличные имена атрибутов, ассоциированные с каждым *Ti*. Каждая из *n* комбинаций «имя атрибута/тип» называется *атрибутом*. С каждым атрибутом ассоциировано *значение атрибута vi* типа *Ti*; каждая из *n* пар атрибут/значение называется компонентом. Множество из всех *n* таких компонент определяют *значение кортежа* (*кортеж*, для краткости) над атрибутами *A1,A2,..,An*. Множество из всех *n* атрибутов называется заголовком кортежа.

Пусть {*H*} будет заголовком кортежей, и пусть *t1, t2, ..., tm* будут различные кортежи заголовка {*H*}. Сочетание {*H*} и множества кортежей {*t1*, *t2*, ..., *tm*} называется *значением отношения* (*отношением*, для краткости), над атрибутами *A1*, *A2*, ..., *An*, где *A1*, *A2*, ..., *An* все атрибуты {*H*}. *Заголовком отношения* будет {*H*}, *телом отношения* будет множество кортежей {*t1*, *t2*, ..., *tm*}.

Выводы из определения:

- В реляционной модели каждый атрибут каждого отношения должен иметь имя (то есть *анонимные атрибуты запрещены*), и такие имена должны быть уникальны в данном отношении (то есть *запрещены дублирующиеся имена атрибутов*). В SQL данные правила уместны не везде: они верны для значений переменных таблиц, определённых с помощью CREATE TABLE и CREATE VIEW, но не для результатов запросов над такими таблицами.
- Отношения *никогда не содержат кортежей-дубликатов*. Это свойство следует из того, что тело отношения определено как множество кортежей, а множество в математике не содержит дублирующихся элементов. Поведение SQL в данном смысле не верно: поскольку таблицы в SQL могут содержать дублирующиеся строки, отсюда они не являются отношениями в общем смысле.
- *Кортежи в отношении не упорядочены сверху вниз.* Это свойство следует опять из того, что тело отношения определено как множество, а в математическом множестве элементы не упорядочены. Конечно, из того факта, что кортежи в отношениях не упорядочены, не следует то, что запросы не могут включать спецификации ORDER BY. Однако это означает, что результат такого запроса не будет отношением. Отсортированные результаты удобны для отображения, однако, такой запрос не является реляционный оператором.
- *Атрибуты в отношении также не упорядочены слева направо*, потому что заголовок отношения - также математическое множество. Поведение SQL в данном контексте неверно: в SQL таблицах колонки упорядочены слева направо.

- *Кортежи никогда не содержат NULL* - поскольку, по определению, для каждого атрибута кортежи содержат значения соответствующего типа.
- *Отношение не содержат NULL* – поскольку тело отношения является множеством кортежей, а кортежи не содержат NULL.

3. Замкнутость реляционной модели. Правила выведения типов отношений и таблиц.

По определению результатом каждого реляционного оператора должно быть отношение. И напротив, любой оператор, результатом которого не является отношение по определению не может быть реляционным оператором. В случае SQL реляционные операторы не могут возвращать результаты, содержащие строки дубликаты, упорядоченные столбцы, NULL, неименованные столбцы, дублирующиеся имена столбцов. Замкнутость реляционной модели крайне важна, оно позволяет нам писать в реляционной модели вложенные выражения. Не используйте любые операторы, нарушающие замыкания, если Вы хотите использовать результат в дальнейших реляционных выражениях.

Реляционная модель использует специальные правила выведения типов отношений, которые предполагают, что, если нам известен заголовок входных отношений в операции, мы можем вывести заголовок выходного отношения этой операции. Преимущество такого метода в избегании сложностей, зависящих от порядкового номера атрибутов.

SQL же поддерживает выведение типов несколько не корректно. Во-первых, в SQL вообще нет понятия реляционных типов, взамен он оперирует типами строк. Во-вторых, он допускает в результирующих таблицах неименованные столбцы. В-третьих, допускает в результирующих таблицах дублирование имен столбцов. Для использования SQL реляционным образом Вам следует применять некоторые правила, как это делают реляционные операторы. *Это предполагает надлежащее именование атрибутов отношений:*

- Для всех атрибутов, представляющих «одну и ту же информацию», по возможности назначать одинаковые имена. И напротив, если два атрибута представляют разную информацию, правильно будет определить для них различные имена. Единственный случай, когда следовать данным рекомендациям невозможно, - если два атрибута одной таблицы предоставляют одинаковую информацию.
- Для каждой базовой таблицы определите представления во избежание возможного переименования атрибутов в будущем.
- Используйте такие представления вместо лежащих в их основе базовых таблиц.

- Используйте спецификацию AS для определения подходящих имен для любых столбцов, которые не имеют имени, или имя их не уникально.

Мы не можем игнорировать факт, что столбцы в SQL остаются упорядоченными даже если нам этого не нужно. Следующий код предполагает упорядоченность атрибутов:

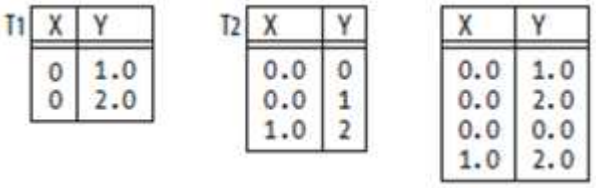
- SELECT *
- JOIN, UNION, INTERSECT и EXCEPT—в особенности если не определена спецификация CORRESPONDING в последних трех случаях.
- В перечислении имен атрибутов, следующем за определением переменной области значений
- В перечислении имен атрибутов, определённых в CREATE VIEW
- INSERT, если не определён список имен атрибутов.
- ALL и ANY сравнения, если размерность операндов больше одного
- В выражениях VALUES

4. Сравнение типов в SQL

Рассмотрим общеизвестный факт, что в реляционной модели два значения могут быть сравнены, только если они одного и того же типа. Основная идея здесь в том, что СУБД должна запрещать выполнять любые реляционные операторы (join, union, что угодно), которые подразумевают явное/неявное сравнение значений различных типов. Было бы полезно, если при попытке пользователя сравнить значения различных типов, СУБД запретит данное сравнение, сообщив об ошибке, и попросит пользователя исправить ошибку перед выполнением. Ни один из известных сегодня SQL продуктов не ведет себя подобным образом. В сегодняшних продуктах в зависимости от того, как Вы настроите базу данных, выполнение запроса прервется или же вернет не верный ответ (точнее не совсем не верный ответ, а правильный ответ на неправильный вопрос).

В SQL же реализована строгая типизация, но в несколько слабой форме. Так сравнение строк и чисел не допустимо, однако сравнение целых и действительных чисел допустимо, поскольку перед выполнением сравнения целое число будет приведено к вещественному типу. Такое приведение типов широко распространено, однако, в контексте SQL его применение к реляционным операциям, таким, как объединение, пересечение и разность, приводит к весьма странным результатам, которые впоследствии содержат строки, не принадлежащие ни одному из операндов. Рассмотрим объединение таблиц T1 {X INTEGER, Y NUMERIC(5,1)} и T2 {X NUMERIC(5,1), Y INTEGER}.

```
SELECT X , Y FROM T1
UNION
SELECT X , Y FROM T2
```



T1	X	Y
	0	1.0
	0	2.0

T2	X	Y
	0.0	0
	0.0	1
	1.0	2

X	Y
0.0	1.0
0.0	2.0
0.0	0.0
1.0	2.0

Таким образом, результат содержит строки не принадлежащие ни таблице T1, ни таблице T2 – весьма странное объединение.

*Рекомендация***:** избегайте приведения типов, где это возможно. Так, в контексте SQL, следует быть уверенным, что колонки с одинаковым именем имеют одинаковый тип. Если избежать этого не удается - используйте явное приведение типов, используя CAST или его эквиваленты.

5. Табличные выражения. Особенности стандарта SQL в сравнении с реляционной моделью

В данной главе мы рассмотрим крайне важное *табличное выражение*, которое возникает в многочисленных контекстах языка SQL. В большинстве случаях, в действительности, его можно рассматривать как вершину синтаксического дерева. Следует отметить, что стандарт ссылается на эту конструкцию как на *выражения запроса* (*query expression*) и приписывает другое (более ограниченное) понятие для табличного выражения (“table expression”). Мы предпочитаем термин *табличное выражение*, поскольку он более точно отражает, что выражение возвращает таблицу (точнее *неименованную* таблицу). Напротив, стандарт предполагает “table expression” просто как частный случай, что мы называем select выражением без SELECT раздела как такового.

5.1 Соединения / JOIN EXPRESSIONS

В общем случае табличные выражение разделены на join и non-join табличные выражения

query-expression-body

::= joined-table | nonjoin-query-expression

Рассмотрим только первый из двух случаев. В общем случае *Join выражение* представляет явное соединение двух таблиц, представленных как *ссылка на*

таблицу (*table reference)*. Более точно, Join выражение это либо *cross join*, либо явно/неявно подразумевает наличие типа соединения (join type).

joined-table::=

table-reference CROSS JOIN table-primary

| table-reference [join-type] JOIN table-primary

[ON search-condition | USING (column-name-list)]

| table-reference NATURAL [join-type] JOIN table-primary

| table-reference UNION JOIN table-primary

Cross Join

“Cross join” - это другое название того, что более точно называть расширенное декартовое произведение (сокращенно просто декартово произведение в контексте SQL). Пусть A и B будут таблицы, представленные как результат вычислений table reference. Тогда результатом выражение “A CROSS JOIN B" будет таблица, содержащая все возможные строки R, которые являются конкатенацией строк из A и B. Из этого следует, что соединение A CROSS JOIN B семантически эквивалентно следующему select-выражению:

```
( SELECT A.*, B.*
FROM A, B )
```

Из определения видно, что общие атрибуты таблиц A и B встречаются в результате дважды.

Другие Join выражения

Синтаксис для других join выражений следующий:

table-reference [join-type] JOIN table-primary

[ON search-condition | USING (column-name-list)]

| table-reference NATURAL [join-type] JOIN table-primary

| table-reference UNION JOIN table-primary

Тип соединения (Join type) в свою очередь может быть одним из следующих:

join-type ::=

INNER | { LEFT | RIGHT | FULL } [OUTER]

Следует отметить что:

1. NATURAL и UNION не могут быть одновременно указаны.

2. Если указано NATURAL или UNION – тогда ни ON условие, ни USING условие не могут быть указаны.
3. Если не указано ни NATURAL, ни UNION – тогда либо ON условие, либо USING условие должны быть указаны.
4. Если тип соединение опущен – тогда INNER тип предполагается по умолчанию
5. Слово OUTER в случаях LEFT, RIGHT, и FULL является необязательным и не оказывает никакого эффекта на общий смысл выражений.

Рассмотрим следующие варианты соединения:

1. table-reference JOIN table-primary ON search-condition
2. table-reference JOIN table-primary USING (column-name-list)
3. table-reference NATURAL JOIN table-primary

Пусть A и B будут таблицы, представленные как результат вычисления table reference.

1. "A JOIN B ON cond," (где cond – условное выражение) по определению семантически эквивалентно следующему select выражению:

   (SELECT A .*, B.*

   FROM A, B

   WHERE cond)

   Из определения видно, что общие атрибуты таблиц A и B встречаются в результате дважды.
2. Пусть список атрибутов в условии USING будет определён как *C1, C2 ,..., Cn*, и каждый атрибут C l, C2, ..., Cn должен одновременно быть атрибутом A и B. Тогда соединение семантически эквивалентно *Случаю 1*, в котором ON условие представлено как - ON A.Cl = B.Cl AND A.C2 = B.C2 AND ... AND A.Cn = B.Cn. За исключением того, что общие атрибуты *C1, C2 ,..., Cn* встречаются в результате только один раз, а не дважды. A атрибуты в результирующей таблицы упорядочены следующим образом: сперва идут общие атрибуты (упорядоченные слева на право, как это указано в условии USING); затем остальные атрибуты A

(в порядке каком они указаны в A); затем остальные атрибуты B(в порядке каком они указаны в B).

3. Выражение семантически эквивалентно *Случаю 2,* в котором список атрибутов включает *все* общие атрибуты A и B. Отметим, что если таких общих атрибутов нет, в таком случае выражение A NATURAL JOIN B сводиться к A CROSS JOIN B.

Рассмотрим следующие примеры:

```
S JOIN SP ON S.SNO = SP.SNO

S JOIN SP USING ( SNO )

S NATURAL JOIN SP
```

Эти три выражения эквивалентны: хотя первое возвращает таблицу, содержащую два одинаковых столбца SNO, второе и третье содержит только один такой столбец.

Отметим, что, хотя результатом вычислений этих выражений является таблица, содержащая множество строк, эти выражения не могут быть использованы как самостоятельный оператор. Необходимо определить *cursor* над этими выражениям и получать строки одну за другой с помощью такого курсора.

***Рекомендации**:*

1. Использование NATURAL JOIN для обозначения соединения более предпочтительно, чем остальные методы - такая формулировка будет наиболее краткой. Однако следует убедиться, что одноимённые столбцы имеют одинаковый тип.
2. Старайтесь избегать использования JOIN ON, поскольку он гарантировано возвращает результат с дублируемыми именами столбцов. Однако, если вам действительно необходимо выполнить некоторый вид экви-соединения, самостоятельно выполните надлежащее именование столбцов над полученным результатом.
3. В использования JOIN USING убедитесь, что столбцы с одинаковым именем имеют одинаковый тип.
4. В случае использования CROSS JOIN убедитесь, что отсутствуют общие имена столбцов.

Замечание по внешнему соединению

Внешние соединения LEFT, RIGHT, FULL and UNION были специально разработаны, чтобы вставлять NULL в таблицы результат, поэтому их

следует рассматривать в контексте влияния NULL на SQL модель данных. Они будут опущены в текущем контексте. Говоря реляционным языком, это некоторый вид принуждения – оно позволяет выполнить объединение таблиц, даже если типы операндов не удовлетворяют требованиям объединения. Внешнее соединение заполняет недостающие столбцы обоих операндов NULL-ами перед выполнением объединения, и таким образом операнды станут удовлетворять требованиям объединения. Но никто не мешает заполнять недостающие столбцы подходящими значениями взамен NULL.

5.2 Ссылки на таблицу / TABLE REFERENCES

Ссылка на таблицу (*table reference*) может ссылаться на некоторую таблицу, не важно именованная это таблица (например, базовая таблица или представление) или неименованная. Ссылки на таблицу используется в SQL для двух целей: они определяют операнды в разделе FROM select выражений и операнды в join выражениях. Что касается синтаксиса, то ссылка на таблицу состоит либо из join выражения (рассмотреные выше), либо из имени таблицы (a), либо из табличного выражения в скобках (b). В *Случае (a)* ссылка на таблицу может выключать необязательную конструкцию AS, целью которой является объявление *переменной области значений (range variable)* для таблицы в запросе, а также указывать имена столбцов для этой таблицы. В *Случае (b)* ссылка на таблицу *должна* включать в себя такую AS конструкцию.

table-reference

::= joined-table

| table [[AS] range-variable [(column-commalist)]]

| (query-expression-body) [AS] range-variable [(column-commalist)]

Список column-commalist, если он определен, должен явно указывать имя для каждого атрибута в соответствующей таблице и не может определять одно имя дважды.

Стоит отметить, что стандарт SQL не использует термин “переменные области значений”. Вместо этого он использует термин “correlation name”, но не определяет тип объектов, к которым можно обращаться, используя данное имя. “Переменные области значений” (range variable) - это общепринятый термин. SQL всегда требует, чтобы select-выражение (также как и join-выражения) формулировались в терминах переменных область значений. Если такие переменные не объявлены явно, тогда SQL предполагает наличие

неявных переменных области значений с тем же именем, что и соответствующие таблицы. Нет ничего плохого в том, чтобы объявлять такие перемены области значений там, где они не требуются явным образом, особенно в сложных выражениях они помогут сделать выражение более понятным. Однако будьте внимательны, поскольку правила области видимости имен в SQL сложны в понимании.

Исследуя вопросы выведения типов в SQL, стоит рассмотреть пример:

```
SELECT P.* , S.SNO , S.SNAME , S.STATUS
FROM P , S
WHERE P.CITY = S.CITY
AND P.PNAME > S.SNAME
```

Выражение P.PNAME > S.SNAME весьма необычно, поскольку выражение предполагает применение к результату выражения FROM, и естественно, переменные отношения P и S не являются частью этого результата. В действительности, сложно объяснить как нечто P.PNAME может появиться в выражениях WHERE и SELECT, поскольку всё должно быть описано исключительно в терминах результата выражения FROM. Стандарт SQL разъясняет это, но способом гораздо более сложным, чем описан в реляционной модели.

Область видимости переменных области значений – то есть контекст, в котором на них можно ссылаться и имя этой переменой должно быть уникальным, определён следующим образом:

- Если ссылка на таблицу присутствует в FROM выражении, тогда областью видимости переменной является select-выражение его включающее (разделы SELECT, FROM, WHERE, GROUP BY, HAVING – которые образует данное select-выражение). Исключая любые вложенные select-выражения или join-выражения, в которых другие переменные области значений могут быть объявлены с тем же именем.
- Если ссылка на таблицу присутствует в join-выражении, тогда областью видимости является это join-выражение. Исключая любые вложенные select-выражения или join-выражения, в которых другие переменные области значений могут быть объявлены с тем же именем.

Описанные замечания по поводу области видимости переменных не описывают все возможные случаи. В случае join-выражений вопрос видимости переменных области значений весьма сложный. Например, если join-выражение, содержащее ссылку на таблицу, присутствует в свою очередь

в разделе FROM – тогда область видимости таблицы расширяется до select-выражения, включающей данный раздел FROM:

```
SELECT DISTINCT SP.PNO, S.CITY
FROM SP NATURAL JOIN S
```

Причина вероятно в том, что область видимости в данном случае должна быть такой же, как если бы join выражение заменили бы двумя операндами join, разделёнными запятой.

```
SELECT ...
FROM SP, S
```

Однако одно очень нелогичное следствие из этих правил видимости показано в следующем примере.

```
SELECT DISTINCT SP.*
FROM SP NATURAL JOIN S
```

Результат выражения будет включать столбцы PNO и QTY, но не столбец SNO – потому что в результат join-выражение не содержит столбец SP.SNO (в действительности, использование SP.SNO в SELECT выражении должно было быть синтаксической ошибкой). Вместо этого результат join выражения содержит столбец SNO без всякого уточнения, что является результатом своего рода объединения столбцов S.SNO и SP.SNO.

5.3 Объединение, разность и пересечение

SQL операторы UNION, EXCEPT и INTERSECT основаны на хорошо известных операторах теории множеств: объединение, разность и пересечение. Две таблицы, которые являются операндами объединения, разности и пересечения, должны иметь одинаковую степень (то есть одинаковое количество столбцов), соответствующие столбцы должны иметь *совместимые типы данных*.

UNION и EXCEPT присутствуют в “nonjoin query expressions”, а INTERSECT присутствует в “nonjoin query terms”.

```
nonjoin-query-expression
        ::=     nonjoin-query-term
```



```
                | query-expression-body { UNION | EXCEPT } [ ALL |
                DISTINCT ]
                        [ CORRESPONDING [ BY ( column-commalist ) ]
                ] query-term
nonjoin-query-term
        ::=     nonjoin-query-primary
                | query-term INTERSECT [ ALL | DISTINCT]
                        [ CORRESPONDING [ BY ( column-commalist ) ]
                ] query-primary
```

Пересечение / INTERSECT

Пусть таблицы A и B будут результатом вычислений table-term и table-primary. Возможны следующие варианты пересечений:

```
1. A INTERSECT [ ALL | DISTINCT] CORRESPONDING BY (
column-commalist ) B
2. A INTERSECT [ ALL | DISTINCT]  CORRESPONDING B
3. A INTERSECT [ ALL | DISTINCT]  B
```

Если не указано спецификатор ALL или DISTINCT, тогда DISTINCT предполагается по умолчанию.

Рассмотрим три случая, которые не содержат опцию ALL.

1. Пусть *Cl, C2 ... Cn* список атрибутов в BY выражении. Каждый атрибут из *Cl, C2 ... Cn* одновременно определяет атрибуты *A* и *B*. Данный случай по определению семантический эквивалентен следующему выражению:

```
( ( SELECT Cl, C2, ..., Cn FROM A )
INTERSECT
( SELECT Cl, C2, ..., Cn FROM B ) )
```

Другими словами, пусть *AC* будет таблица, полученая из *A* удалением всех столбцов, которые не отмечены в BY выражении; и таблица *BC* получена из *B* аналогичным образом. Тогда результатом пересечения будет таблица из *n* столбцов, строка *R* присутствует в результате тогда и только тогда, когда она присутствует и в *AC,* и в *BC*. Результат выражения не содержит строки дубликаты.

2. Пересечение по определению семантически эквивалентно пересечению в *Случае 1,* в котором список атрибутов включает в себя все атрибуты, общие для *A* и *B*.
3. В данном случае пересечение выполняется не согласно сравнению значений атрибутов с одинаковым именем, как в *Случае 1 и 2*, а сравниваются в *соответствии порядковой позицией данных атрибутов*. То есть, *i-ый* атрибут из *A* сравнивается с *i-ым* атрибут из *B* (для всех *i* от 1 до *n,* где *n* – степень *n;* напоминаем, что *A* и *B* должны иметь одинаковую степень). Результат также будет степени *n*, и некая строка присутствует в результате тогда и только тогда, когда она одновременно присутствует и в *A,* и в *B*. Опять же результат выражения не будет содержать дубликаты.

Если определена опция ALL в указанных трех случаях, то результат будет идентичным, за исключением того, что результат будет содержать строки дубликаты. Точнее, предположим, что строка *R* содержится *m* раз в первом операнде и *n* раз во втором операнде ($m > 0$, $n > 0$) – тогда строка *R* будет присутствовать в результате *p* раз, где *p* меньшее из *m* и *n*.

Объединение / UNION

Для операции объединения множество возможных случаев аналогично операции пересечения:

1. A UNION [ALL | DISTINCT] CORRESPONDING BY (column-commalist) B
2. A UNION [ALL | DISTINCT] CORRESPONDING B
3. A UNION [ALL | DISTINCT] B

Эти случаи определены аналогично случаям пересечения, учитывая, что для операции UNION некая строка содержится в результирующей таблице тогда и только тогда, когда она содержится как минимум в одной из таблиц операнда. По умолчанию, результат не содержит строки дубликата. В случае, если указана опция ALL, и предполагая, что строка *R* содержится *m* раз в первом операнде и *n* раз во втором операнде ($m > 0$, $n > 0$) – тогда строка *R* будет присутствовать в результате *p* раз, где $p = m + n$.

Разность / EXCEPT

Множество возможных случаев аналогично случаям пересечения:

1. A EXCEPT [ALL | DISTINCT] CORRESPONDING BY (column-commalist) B
2. A EXCEPT [ALL | DISTINCT] CORRESPONDING B

3. A EXCEPT [ALL | DISTINCT] B

И снова эти случаи определены аналогично случаям пересечения, учитывая, что для операции EXCEPT некая строка содержится в результирующей таблице тогда и только тогда, когда она содержится в первой таблице операнде и не содержится во второй таблице операнде. По умолчанию результат не содержит строки дубликаты. В случае если указана опция ALL, и предполагая, что строка *R* содержится *m* раз в первом операнде и *n* раз во втором операнде ($m > 0$, $n > 0$) – тогда строка *R* будет присутствовать в результате *p* раз, где *p* больше из $m - n$ и *0*.

Рекомендации:

- Убедитесь, что соответствующие столбцы имеют одинаковые имена и типы.
- Всегда определяйте COREESPONDING, где это возможно. Поскольку не все SQL продукты могут поддерживать ее, в таком случае, вам придется самостоятельно позаботиться об упорядочивание столбцов.
- Не используйте опцию BY в спецификации CORRESPONDING
- Никогда не определяйте опцию ALL, и более того, желательно использовать опцию DISTINCT явно. Зачастую пользователи указывают ALL в случаях, если знают об отсутствии дубликатов во входных таблицах, и указывают системе не тратить время, удаляя их.

Замечание по поводу синтаксиса

Дополнительно следует отметить некоторое несоответствие синтаксиса NATURAL JOIN и UNION (INTERSECT и EXCEPT). Предположим, что A и B именованные таблицы, типы которых удовлетворяют требованиям UNION, тогда следующие выражения имеют верный или не верный синтаксис:

A NATURAL JOIN B	-- верно
A UNION B	-- не верно
TABLE A UNION TABLE B	-- верно
TABLE A NATURAL JOIN TABLE B	-- не верно
SELECT * FROM A UNION SELECT * FROM B	-- верно
SELECT * FROM A NATURAL JOIN SELECT * FROM B	-- не верно
(TABLE A) UNION (TABLE B)	-- верно
(TABLE A) NATURAL JOIN (TABLE B)	-- не верно
(TABLE A) AS AA NATURAL JOIN (TABLE B) AS BB	-- верно

5.4 Простая таблица / Simple table

Как было описано ранее, простой запрос (query primary) используется для обозначения второго операнда операции пересечения.

query-primary

::= joined-table | nonjoin-query-primary

nonjoin-query-primary

::= (nonjoin-query-expression) | simple-table

simple-table

::= query-specification

| TABLE table

| table-value-constructor

Ранее мы уже рассмотрели соединения (joined table) и пересечения, объединения и разность (nonjoin query expressions). Сейчас мы остановимся на рассмотрении выражения “TABLE table” и конструкторы таблиц (table-value-constructor). Select выражения (query-specification) будут рассмотрены позже.

- Выражение “TABLE table” (где table – именованная таблица: базовая таблица или представление) по определению семантически эквивалентно выражению:
  (SELECT * FROM table)
- Конструктор таблиц представляет собой список конструкторов строк:
  VALUES row-value-expression-list,

где каждый конструктор строки определяет в точности одну строку, а конструктор таблицы определяет таблицу, которая является своего рода “UNION ALL” таких строк.

Конструктор строк в свою очередь может иметь следующую форму:

[ROW](row-value-constructor-element-conmalist) | (query-expression)

Другими словами, конструктор строк представляет собой или список компонентов строки в скобках, или табличное выражение в скобках:

1. Компонентом строки может быть скалярное выражение или одно из ключевых слов: DEFAULT или NULL. (DEFAULT и NULL допустимы, только если конструктор строки используется в INSERT выражении)

2. Табличное выражение в скобках должно вычисляться как таблица, содержащая в точности одну строку. В этом случае значением конструктора строки будет полученная строка. Стандарт SQL определяет такое выражение как строчное выражение (*row subquery).* Возникнет ошибка времени выполнения, если данное строчное выражение вернет таблицу, содержащую более одной строки. *Однако, если данное строчное выражение вернет таблицу, не содержащую ни одной строки, тогда значением конструктора строки будет строка полностью состоящая из NULL.*

***Рекомендации**:*

- Поскольку столбцы таблицы упорядочены слева направо, и в конструкторе не поддерживается определение значения атрибута по его имени, вам необходимо убедиться, что все строки имеет соответствующий тип – то есть на i-ой позиции строки должно находиться значение i-ого атрибута таблицы.
- Убедитесь, что не описали одну строку дважды.

Пусть T будет именованная таблица, имена и типы атрибутов совпадают с таблицей поставщиков S. Тогда следует отметить, что следующие два INSERT выражения отличаются по смыслу:

1. INSERT INTO T VALUES (SELECT * FROM S WHERE SNO = 'S6')
2. INSERT INTO T (SELECT * FROM S WHERE SNO = 'S6')

Предполагая, что выражение SNO = 'S6' ложно, тогда первое выражение вставляет в таблицу T строку, полностью состоящую из NULL; в то время как второе выражение не вставляет ни одной строки.

5.5 SELECT выражение

В качестве select-выражения может быть рассмотрено табличное выражение, которое не содержит никаких JOIN, UNION, EXCEPT или INTERSECT. Хотя такие операторы могут использоваться в качестве вложенных.

query-specification

::= SELECT [ALL | DISTINCT] select-list

FROM table-reference-commalist

[WHERE search-condition]

[GROUP BY [ALL | DISTINCT] column-commalist]

[HAVING search-condition]

Раздел SELECT

SELECT [ALL | DISTINCT] select-list

1. Если не указан спецификатор ALL или DISTINCT, ALL предполагается по умолчанию.
2. Мы предполагаем, что результатом вычисления разделов FROM, WHERE, GROUP BY и HAVING (не важно, какие из них присутствуют в запросе, а какие нет) будет таблица *T1*.(В действительности *T1*- неименованная таблица).
3. Пусть *T2* будет таблица, полученная из *T1* вычислением списка select-list.
4. Если указано ключевое слово DISTINCT, тогда *T3* будет таблица, полученная из *T2* удалением строк дубликатов. Если DISTINCT не указано, тогда таблица *T3* будет равна *T2*.
5. *T3* будет результат вычисления данного select-выражения.

Рассмотрим список элементов select-list. Возможны два случая, причем второй случай - всего лишь сокращённая форма первого случая.

1. Элемент select-list имеет вид:

   scalar-expression [[AS] column]

Скалярное выражение обычно включает один или более атрибутов таблицы *T1*. Для каждой строки таблицы *T1* вычисляется данное скалярное выражение. Список результатов вычисления таких скалярных выражений по строке таблицы *T1* является строка таблицы *T2*. Раздел AS, если он присутствует, определяет имя столбца в результирующий таблице E2 для заданного скалярного выражения. Само ключевое слово AS является необязательным.

2. Элемент select-list имеет вид:

   [range-variable .] *

- Если имя переменной области значений опущено (то есть, элемент select-list представлен ‘*’), тогда этот select-item должен быть единственным в разделе SELECT. Такая форма является сокращением для списка всех атрибутов таблицы *T1*, *в порядке слева направо.*
- Если элемент select-list содержит имя переменной области значений, тогда select-item представляет список всех атрибутов указанной переменной *в порядке слева направо.* Как мы помним, в качестве переменной области значений не явно может быть использовано имя таблицы.

- Если данный элемент select-list используется при определении представлений или ограничений, тогда список атрибутов вычисляется в момент определения данных ограничений или представлений. *Следовательно, если впоследствии в соответствующие таблицы будут добавлены новые атрибуты, то это не повлияет на данные представления или ограничения.*

Рекомендации:

- Всегда указывайте опцию DISTINCT, чтобы позаботиться об удалении строк дубликатов.

Раздел FROM

FROM table-reference-commalist

Предположим, список ссылок на таблицу представлен в виде таблиц *A, B , C*. Тогда результатом вычисления раздела FROM будет таблица, которая является декартовым произведением таблиц *A, B , C*. Декартовое произведение от одной таблицы *T* по определению равно *T*.

Раздел WHERE

WHERE search-condition

Пусть *T* будет таблица, полученая после вычисления раздела FROM. Тогда результатом раздела WHERE будет таблица, полученная из *T* удалением всех строк, для которых условное выражение (search-condition) не равно *true*. Если раздел WHERE опущен, результатом будет таблица *T*.

Раздел GROUP BY

GROUP BY [ALL | DISTINCT] column-commalist

Пусть *T* будет результат вычисления разделов FROM и WHERE. Каждый атрибут в списке column-commalist в разделе GROUP BY должен быть атрибутом таблицы *T*. Результатом раздела GROUP BY будет сгруппированная таблица, то есть множество из групп строк, полученное из *T*, где строки в группе имеют одинаковое значение для атрибутов, указанных в разделе GROUP BY. Если не указан спецификатор ALL или DISTINCT, ALL предполагается по умолчанию. Если указано ключевое слово DISTINCT, тогда группы строк не будут содержать дубликатов. Такие группы сортированы внутри множества по количеству строк в группе от большего к меньшему. Отметим, что результат GROUP BY не является правильной таблицей, поэтому раздел GROPU BY никогда не используется без соответствующего

раздела SELECT, задача которого получить правильную таблицу из промежуточного результата.

Если select выражение включает в себя раздел GROUP BY, то дополнительное ограничение накладывается на раздел SELECT. Каждый элемент списка SELECT должен иметь общее значение для группы. Предполагается, что такие элементы не должен включать в себя никакие ссылки на атрибуты из таблицы T, которые не являются общими для группы, если только такие ссылки не являются аргументом (или частью выражения аргумента) для функции агрегации (функции над множеством), которая вычисляет единственное скалярное значение для коллекции скалярных значений.

Раздел HAVING

HAVING search-condition

Пусть *G* будет сгруппированная таблица, полученная после вычисления FROM, WHERE и GROUP BY разделов. Если раздел GROUP BY опущен, тогда *G* представляет собой сгруппированную таблицу, полученную после вычисления разделов FROM и WHERE, которая содержит только одну группу. Таким образом, полагается, что объявлен неявный раздел GROUP BY, который не содержит ни одного атрибута. Результатом раздела HAVING будет сгруппированная таблица, полученная из *G* удалением всех групп, для которых условное выражение (search-condition) не равно *true*. Скалярное выражение в разделе HAVING должно оперировать общими значениями для группы (как скалярные выражения в разделе SELECT, если присутствует раздел GROUP BY). Если раздел HAVING опущен, а раздел GROUP BY присутствует – тогда результатом будет таблица *G*. Если оба раздела HAVING и GROUP BY опущены – тогда результатом будет правильная (несгруппированная) таблица *T*, полученная вычислением разделов FROM и WHERE.

Замечание по поводу избыточности

Следует отметить, что разделы GROUP BY и HAVING являются в действительности избыточными. Имеется ввиду, что для каждого select-выражения, включающего данные разделы, существует эквивалентное по смыслу выражение, которое не будет включать эти разделы.

6. Заключение

В данной статье рассмотрено поведение табличных выражений стандарта SQL в сравнении с реляционными операторами. В реляционной модели понятие отношений гарантирует что: отсутствуют строки дубликаты, отсутствуют дублирующиеся столбцы, отсутствуют неименованные столбцы, отсутствует

упорядоченность столбцов, отсутствует упорядоченность строк, отсутствуют неопределённые значения (NULL). В статье были показаны случаи, в которых операторы SQL ведут себя отличным от реляционной модели образом, то есть возвращают таблицы, которые не являются отношениями. Также были описаны практики, которые позволяют вам избежать таких случаев и использовать SQL исключительно реляционным образом.

Использование SQL реляционным образом возможно в большинстве случаев. Однако, поскольку существующие реализации далеки от идеальных, временами вы обнаружите, что это приводит к потери производительности. Если в таких случаях вы склонитесь к не «истинно реляционному» использованию, тогда следует понимать, что вы идете на такие компромиссы с позиций концептуальной целостности. Всегда следует понимать правильную в теории ситуацию, и вы должны иметь веские основания уклониться от неё. Вам также следует описать эти причины, для того чтобы отказаться от них в будущем (к примеру, если новая версия продукта, который вы используете, лучше решает интересующую вас задачу), тогда вы сможете вернуться к оригинальным идеям.

7. Список литературы

[1]. C.J. Date. Hugh Darwen. A gude to the SQL standard. 4th edition. Addison-Wesley. (1997).

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

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

[4]. C. J. Date, Hugh Darwen. "Foundation for Future Database Systems: The Third Manifesto", Addison-Wesley Pub Co; 2nd edition (2000).

[5]. 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.

[6]. К. Дж. Дейт. Введение в системы баз данных. Изд. 8-е./ Перев. с англ. – М.: Вильямс, 2005.

[7]. С.Д. Кузнецов. Базы данных: языки и модели. – М.: ООО «Бином-Пресс», 2008 .

Features of SQL table expressions and their compliance with the concepts of relational model.

Ivan. V. Bludov
ivan.bludov@gmail.com
MSU, CMC Department, Moscow, Russia

Abstract. The paper analyses a behavior of the SQL standard's table expressions in comparison with relational equivalents. The concept of relation of relational model presumes nonexistence of duplicate tuples, attributes of the same name, ordering of attributes, ordering of tuples, no-typed NULLs. The paper demonstrate cases where behavior of constructions of the SQL-standard violates the rules of the relational model: these constructions produce tables that are not relations (join expressions, table references, set-theoretical operators, simple tables, and select). The paper also describes practices that allow to avoid such cases and to use SQL in truly relational manner. Relational use of SQL is possible nearly always. However, existing implementations of SQL are far from ideal, and sometimes such use leads a poor performance. Non-relational use of SQL means a tradeoff between performance and conceptual consistence. It is always useful to understand a theoretically perfect variant of use and to have solid reasons to abandon it. It is also useful to document these reasons to return to the perfect variant with some new version of SQL-based DBMS that does not demonstrates problems of performance.

Keywords: SQL, the relational model, SQL query, a table expression

References

[1]. C.J. Date. Hugh Darwen. A gude to the SQL standard. 4th edition. Addison-Wesley. (1997).
[2]. C.J. Date. SQL and Relational Theory: How to Write Accurate SQL Code. O'Reilly Media, Inc. (2009).
[3]. C. J. Date. Database in Depth: Relational Theory for Practitioners. Sebastopol, Calif.: O'Reilly Media, Inc. (2005).
[4]. C. J. Date, Hugh Darwen. Foundation for Future Database Systems: The Third Manifesto, Addison-Wesley Pub Co; 2nd edition (2000).
[5]. C.J. Date. Vvedenie v sistemy baz dannyh [An Introduction to Database Systems]. Izd. 8-e./ Perev. s angl. – M.: Vil'jams, 2005 (in Russian).
[6]. S.D. Kuznetsov. Bazy dannyh: jazyki i modeli [Databases: languages and models]. – M.: OOO «Binom-Press», 2008 (in Russian).