БАЗИ ДАНИХ, БАЗИ ЗНАНЬ ТА ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 681.3

Институт программных систем НАН Украины

В.А. Резниченко

Рекурсивный SQL

Рассматриваются возможности языка SQL по формулированию рекурсивных запросов. Наличие рекурсии в SQL позволяет выражать в этом языке произвольные интенсиональные правила языка Datalog дедуктивных баз данных.

Розглядаються можливості мови SQL по формулюванню рекурсивних запитів.

Наявність рекурсії в SQL надає можливість виражати у цій мові довільні інтенсіональні правила мови Datalog дедуктивних баз даних.

Possibilities of SQL language concerning recursive queries formulation are considered. Recursion availability in SQL gives possibilities to express arbitrary intestinal rules of deductive database Datalog language.

Ключевые слова: рекурсивный SQL , дедуктивная база данных, язык Datalog

Рекурсия была введена в SQL для поддержания возможности интенсиональных правил дедуктивной базы данных. Как известно [1], имеется три вида интенсиональных правил в дедуктивной базе данных: простые, рекурсивные и с отрицаниями. Простые правила и специальные правила с отрицаниями полностью выразимы в стандартном (без рекурсии) SQL. Включение рекурсии в SQL привело к тому, что все возможности дедуктивных интенсиональных правил выразимы в SQL.

В данной статье раскрываются возможности рекурсивного SQL на многочисленных примерах.

В стандарте SQL–99 было расширено определение запроса. Его синтаксис принял следующий вид:

```
[фраза_WITH] запрос
```

Здесь под запрос подразумевается стандартное понимание запроса. В свою очередь фраза WITH, которая может располагаться в начале запроса, используется для достижения следующих целей:

- описать подзапросы, которые многократно используются в самом запросе, с тем, чтобы к ним (подзапросам) можно было обращаться в запросе по имени;
- описать рекурсивное выполнение запроса.

Эта фраза имеет следующий немного упрощенный синтаксис:

```
  WITH [RECURSIVE]
имя_подзапроса_1 [(список_имен_столбцов_1)]
  AS (подзапрос_1) [характеристики_поиска_1]
[,имя_подзапроса_2 (список_имен_столбцов_2)
  AS (подзапрос_2) [характеристики_поиска_2]]..
  .
```

Давайте рассмотрим следующий фрагмент запроса:

```
WITH B AS
 (SELECT ... FROM A ...
    UNION ALL
  SELECT ... FROM A,V ...)
SELECT ....
```

Это нерекурсивный запрос. Если теперь мы в нем заменим «V» на «B» то он по форме преобразуется в рекурсивный:

```
WITH B AS
 (SELECT ... FROM A ...
    UNION ALL
  SELECT ... FROM A,B ...)
SELECT....
```

Чтобы явно указать, что это рекурсивный запрос, а не случайное совпадение, вводится ключевое слово RECURSIVE:

```
WITH RECURSIVE B AS
 (SELECT ... FROM A ...
    UNION ALL
  SELECT ... FROM A,B ...)
SELECT....
```

© Резниченко В.А., 2010

Если это ключевое слово используется в том случае, когда контекст запроса не является рекурсивным, то есть:

```
   WITH RECURSIVE B AS
    (SELECT ... FROM A ...
        UNION ALL
     SELECT ... FROM A,V ...)
SELECT....
```

то ключевое слово игнорируется. С другой стороны, если ключевое слово не используется, однако запрос является рекурсивным, то это рассматривается как ошибка.

При вычислении рекурсивного запроса предполагается, что сначала таблица B пустая. При первом рекурсивном выполнении этого запроса таблица B получит результат выполнения подзапроса `SELECT ... FROM A ...`, так как второй подзапрос даст пустое значение в связи с тем, что он ссылается на пустую таблицу B. Все последующие рекурсивные вычисления используют именно этот второй подзапрос, так как он имеет рекурсивное определение таблицы B через саму себя. На каждом итерационном цикле текущее состояние таблицы B используется для формирования ее нового состояния. Такое вычисление завершается тогда, когда очередной цикл построения таблицы B не приводит к порождению новых строк в этой таблице.

Рассмотрим ряд примеров использования рекурсивных запросов.

1.Примеры рекурсивных запросов: рейсовые полеты

Пусть имеется таблица рейсовых полетов `FLIGHTS`, которая помимо пар городов, между которыми осуществляются полеты, содержит информацию о стоимости полета. Определение таблицы и ее содержимое приведено ниже. На рис. 1 также приведена схема рейсовых полетов.

```
   CREATE TABLE FLIGHTS
  (FromCity varchar(15),
   ToCity   varchar(15),
   Cost     numeric(4));
   SELECT * FROM FLIGHTS;
   FROMCITY    TOCITY       COST
------------ ------------ ------
Москва       Минск          50
Москва       Киев           70
Москва       Кишинев        80
Москва       Ростов         80
Москва       Симферополь   100
Минск        Кишинев        40
Кишинев      Симферополь    10
Киев         Симферополь    40
Ростов       Симферополь    30
9 строк выбрано.
```

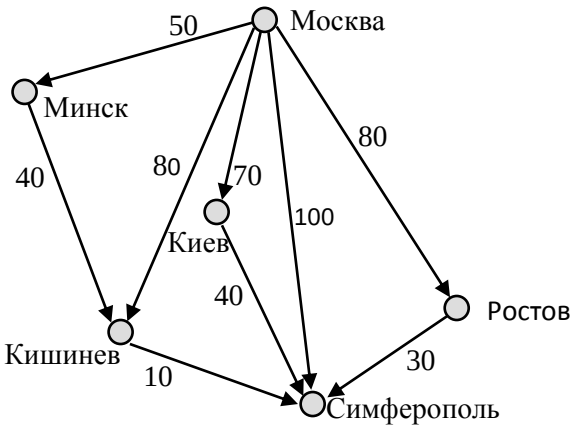

Рис. 1 Рейсовые полеты

1.1 Рекурсия с накоплением

Пусть необходимо найти цену самого дешевого перелета из Москвы в Симферополь. Для решения этой задачи сначала следует построить рекурсивный запрос, который бы вычислял все достижимые из Москвы города с указанием стоимости полета с учетом различных маршрутов перелета. И затем в основном запросе необходимо будет найти строку с конечным городом «Симферополь» и с минимальным значением стоимости. Вот как это выглядит в стандарте SQL:

```
   WITH RECURSIVE REACHABLE(FromCity, Destination, To-
tal_Cost) AS
   (SELECT FromCity, ToCity, Cost
    FROM   FLIGHTS
    WHERE  FromCity = 'Москва'
        UNION
    SELECT inn.FromCity, outt.ToCity, inn.Total_Cost +
outt.Cost
    FROM   REACHABLE inn, FLIGHTS outt
    WHERE  inn.Destination = outt.FromCity
   )
   SELECT FromCity, Destination, MIN(Total_Cost)
FROM   REACHABLE
WHERE  Destination = 'Симферополь'
GROUP  BY FromCity, Destination;
```

Чтобы показать, как этот рекурсивный запрос вычисляется, представим его в виде нескольких нерекурсивных, запоминая результаты промежуточных шагов в отдельных таблицах.

Создадим таблицу REACHABLE и запомним в ней результаты первого запроса рекурсии (стоящего до предложения UNION):

```
CREATE TABLE REACHABLE(FromCity, Destination, Total_Cost) AS
    (SELECT FromCity, ToCity, Cost
     FROM   FLIGHTS
     WHERE  FromCity = 'Москва');
FROMCITY        DESTINATION     TOTAL_COST
--------------- --------------- ----------
Москва          Минск                   50
Москва          Киев                    70
Москва          Кишинев                 80
Москва          Ростов                  80
Москва          Симферополь            100
```

Это начальная фаза, мы получили все города, имеющие непосредственные рейсы из Москвы. Теперь посмотрим, что дает запрос, стоящий после UNION:

```
  SELECT inn.FromCity, outt.ToCity, inn.Total_Cost + outt.Cost
FROM   REACHABLE inn, FLIGHTS outt
WHERE  inn.Destination = outt.FromCity;
  FROMCITY        TOCITY          INN.TOTAL_COST+OUTT.COST
  --------------- --------------- ------------------------
  Москва          Кишинев                              90
  Москва          Симферополь                         110
  Москва          Симферополь                          90
  Москва          Симферополь                         110
```

© Резниченко В.А., 2011

Это те города, к которым можно добраться из Москвы с одной пересадкой с подсчетом суммарной стоимости перелета из Москвы в этот город.

Создадим таблицу REACHABLE2, которая является объединением REACHABLE и последнего запроса.

```
CREATE TABLE REACHABLE2(FromCity, Destination, Total_Cost) AS
(SELECT * FROM REACHABLE
   UNION
 SELECT inn.FromCity, outt.ToCity, inn.Total_Cost + outt.Cost
 FROM   REACHABLE inn, FLIGHTS outt
 WHERE  inn.Destination = outt.FromCity);
```

Вот как она выглядит:

```
FROMCITY        DESTINATION     TOTAL_COST
--------------- --------------- ----------
Москва          Киев                    70
Москва          Кишинев                 80
Москва          Кишинев                 90
Москва          Минск                   50
Москва          Ростов                  80
Москва          Симферополь             90
Москва          Симферополь            100
Москва          Симферополь            110
```

Наконец, выполним еще один цикл рекурсии с созданием таблицы REACHABLE3 на основе REACHABLE2 и FLIGHTS. Получим города, достижимые из Москвы непосредственно, а также за одну и две пересадки.

```
CREATE TABLE REACHABLE3(FromCity, Destination, Total_Cost) AS
(SELECT * FROM REACHABLE2
    UNION
 SELECT inn.FromCity, outt.ToCity, inn.Total_Cost + outt.Cost
 FROM   REACHABLE2 inn, FLIGHTS outt
 WHERE  inn.Destination = outt.FromCity);
```

```
FROMCITY        DESTINATION     TOTAL_COST
--------------- --------------- ----------
Москва          Киев                    70
Москва          Кишинев                 80
Москва          Кишинев                 90
Москва          Минск                   50
Москва          Ростов                  80
Москва          Симферополь             90
Москва          Симферополь            100
Москва          Симферополь            110
```

Последующие рекурсивные вычисления ничего нового не дадут, так как в нашей схеме нет маршрутов с более чем двумя пересадками.

Теперь выполним основной запрос (но только по отношению к REACHABLE3):

```
   SELECT FromCity, Destination, MIN(Total_Cost)
FROM   REACHABLE3
WHERE  Destination = 'Симферополь'
GROUP  BY FromCity, Destination;
```

```
   FROMCITY        DESTINATION     MIN(TOTAL_COST)
--------------- --------------- ---------------
Москва          Симферополь                  90
```

Именно такой результат должен был бы выдать рекурсивный запрос. Минимальная стоимость перелета из Москвы в Симферополь — 90.

1.2 Рекурсия и соединение таблиц

Приведем пример рекурсии с нестандартным соединением таблиц.

Запрос. Найти все маршруты между городами и количество промежуточных городов, имеющих рейсы за границу. Таблица ABROAD_CITIES содержит список городов, имеющих рейсы за границу.

```
 WITH RECURSIVE REACHABLE(FromCity, ToCity, Abroad_count) AS
(SELECT FromCity, ToCity, 0
 FROM   FLIGHTS
   UNION ALL
 SELECT inn.FromCity, outt.ToCity,
-- При необходимости добавляем город, имеющий рейсы за границу
 Abroad_count + CASE WHEN ABROAD_CITIES.Name IS NULL
                          THEN 0 ELSE 1
                    END AS Abroad_count
 FROM REACHABLE inn
      INNER JOIN (FLIGHTS outt LEFT OUTER JOIN ABROAD_CITIES
                  ON (outt.FromCity =ABROAD_CITIES.name))
      ON (inn.ToCity = outt.FromCity)
)
SELECT * FROM REACHABLE
```

Так как не все рейсы проходят через города, имеющие рейсы за границу, то вместо обычного соединения следует использовать внешнее соединение. Если предположить, что таблица ABROAD_CITIES содержит города Москва, Киев, Минск, Кишинев, то результат вычисления этого рекурсивного запроса будет следующим:

```
  FROMCITY        TOCITY          ABROAD_COUNT
--------------- --------------- ------------
Киев            Симферополь                0
Кишинев         Симферополь                0
Минск           Кишинев                    0
Минск           Симферополь                1
Москва          Киев                       0
Москва          Кишинев                    0
Москва          Кишинев                    1
Москва          Минск                      0
Москва          Ростов                     0
Москва          Симферополь                0
Москва          Симферополь                1
Москва          Симферополь                2
Ростов          Симферополь                0
  13 строк выбрано.
```

1.3 Отрицание и группирование в рекурсии

Как следует из теории дедуктивных баз данных, рекурсивные запросы с отрицанием не всегда являются вычислимыми. Не будем вдаваться в подробности этого вопроса, а просто сформулируем утверждение, которое указывает, когда именно отрицание в рекурсии является допустимым:

Отрицание в рекурсии разрешимо, если оно (отрицание) применяется к таблицам, которые являются полностью известными (то есть уже вычислены).

Приведем пример отрицания.

Запрос. Найти города, которые являются достижимыми из других городов кроме Москвы.

```
with RECURSIVE Reachable_From (FromCity, ToCity) AS
(SELECT FromCity, ToCity
 FROM   Flights
    UNION
 SELECT inn.FromCity, out.ToCity
 FROM   Reachable_From inn, Flights out
 WHERE  inn.ToCity = out.FromCity
)

   SELECT * FROM Reachable_From
EXCEPT
SELECT * FROM Reachable_From WHERE FromCity = 'Москва';
   FROMCITY      TOCITY
------------ -----------
Киев         Симферополь
Кишинев      Симферополь
Минск        Кишинев
Минск        Симферополь
Ростов       Симферополь
   5 строк выбрано.
```

Эти же замечания относятся к группированию и агрегатным функциям, а именно:

Группирование и агрегатные функции следует применять только к полностью известным (то есть уже вычисленным) таблицам.

2. Примеры рекурсивных запросов: комплектация изделий

Существует целый класс задач на структуре данных, которая описывает многоуровневую комплектацию изделий, когда изделия включают в свой состав другие изделия, которые, в свою очередь состоят из третьих изделий и так далее. Причем вхождение одного изделия в другое может характеризоваться различными свойствами. Во всех рассматриваемых далее примерах таким свойством будет количество входящих в изделие других изделий. Схема используемой комплектации приведена на рис. 2, а таблицы имеют следующее определение и содержимое:

```
   CREATE TABLE PART_PART
(Major CHAR(3), Minor CHAR(3), Qty NUMBER(2));
   MAJOR MINOR  QTY

----- ----- ----
p1    p2       2
p1    p3       3
p1    p4       2
p2    p3       3
p2    p5       2
p3    p5       4
p3    p6       2
p4    p3       5
p4    p6       2
p7    p8       3
p7    p9       2
   11 строк выбрано.
```

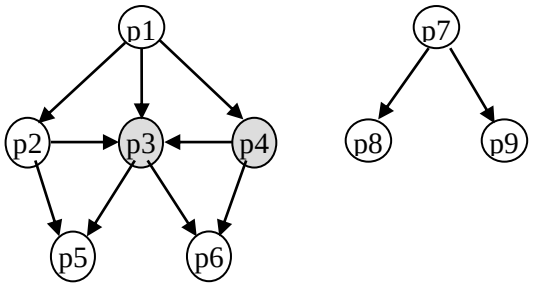

Рис. 2. Граф-схема комплектации изделий

2.1 Вычисление промежуточных количественных характеристик

Давайте на заданном графе комплектации изделий подсчитаем количество необходимых деталей (изделий) в зависимости от пути их достижения от корневой вершины:

```
   WITH RECURSIVE PX (Major, Minor, Qty) AS
   (SELECT Major, Minor, Qty
    FROM   PART_PART pp
    WHERE  pp.Major = 'P1'
      UNION ALL
    SELECT pp.Major, pp.Minor, PX.Qty * pp.Qty
    FROM   PX, Part_Part pp
    WHERE  pp.Major = PX.Minor
    )
SELECT Major, Minor, Qty FROM PX;
   Результат будет следующим:
   MAJOR MINOR  QTY
----- ----- ----
p1    p2       2
p1    p3       3
p1    p4       2
p2    p3       6
p2    p5       4
p3    p5      40
p3    p5      12
p3    p5      24
p3    p6      12
p3    p6      20
p3    p6       6
p4    p3      10
p4    p6       4
   13 строк выбрано.
```

Объясним смысл полученного результата на примере трех строк, имеющих в первых двух столбцах значения p3 и p5 (они выделены жирным шрифтом). От корня и до вершины p3 имеется три пути.

Если проследовать по первому пути, p1 → p2 → p3, то получим 6 экземпляров изделия p3, а так как каждый p3 содержит по 4 экземпляра изделия p5, то общее количество p5 становится равным 24.

Второй путь, p1 → p3 дает 3 экземпляра p3, а значит 12 экземпляров p5.

Наконец, третий путь, p1 → p4 → p3, дает 10 экземпляров p3, и следовательно 40 экземпляров p5.

2.2 Итоговые количественные вычисления

В предыдущем примере мы получили количество деталей, необходимых согласно существующим путям, ведущим к этим деталям. Теперь не представляет труда подсчитать общее количество деталей каждого типа, которые входят в состав всего изделия. При этом текст рекурсивного запроса во фразе WITH не изменяется. Меняется только текст основного запроса, как это показано ниже.

© Резниченко В.А., 2011

```
   WITH RECURSIVE PX (Major, Minor, Qty) AS
  (SELECT Major, Minor, Qty
   FROM   PART_PART pp
   WHERE  pp.Major = 'p1'
      UNION ALL
   SELECT pp.Major, pp.Minor, PX.Qty * pp.Qty
   FROM   PX, PART_PART.pp
   WHERE  pp.Major = PX.Minor
  )
   SELECT Minor AS PART, SUM(Qty) AS TOTAL_QUANTITY
FROM   PX
GROUP  BY Minor;
   Окончательный результат следующий:
   PART TOTAL_QUANTITY
---- --------------
p2                2
p3               19
p4                2
p5               80
p6              126
```

2.3 Игнорирование фиктивных изделий

Комплектация изделий может предполагать, исходя из тех или иных соображений, включение так называемых фиктивных изделий. Их реально нет, но они вводятся, например, для удобства путем объединения нескольких деталей в искусственный агрегат (комплект). В связи с этим в некоторых случаях появляется необходимость произвести количественные вычисления без учета таких искусственных агрегатов. Можно также сказать, что это задача изменения структуры граф-схемы, когда некоторые из ее вершин удаляются, как, например, на рис. 3.

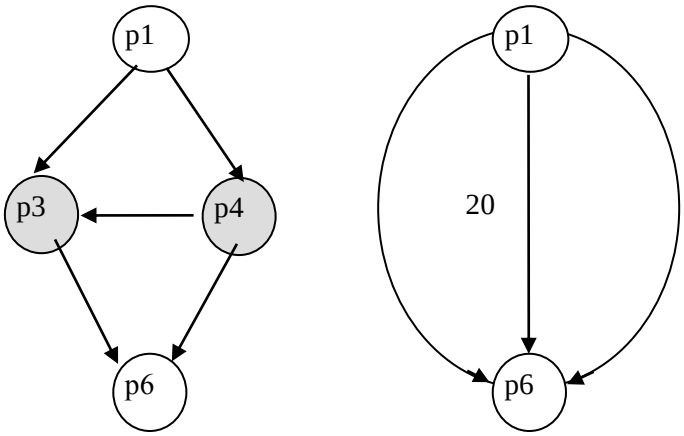

Рис. 3. Изменение структуры граф-схемы

Приведем пример, как можно выразить такие изменения в рекурсивных запросах. Для этого воспользуемся приведенной ниже таблицей, в которой, в частности, имеется указание того, какие именно детали являются фиктивными (столбец `Dummy`, детали p3 и p4).

```
   CREATE TABLE PARTMASTER
      (Pno CHAR(3), Pname CHAR(10), Plevel CHAR(8), Dummy
   CHAR(5));
   PNO PNAME      PLEVEL   DUMMY
--- ---------- -------- -----
p1  главная    корень   нет
p2  полка               нет
p3  комплект1           да
p4  комплект2           да
p5  подпорка   лист     нет
p6  уголок     лист     нет
   6 строк выбрано.
```

Вот как выглядит вычисление количества деталей с удалением промежуточных фиктивных вершин.

```
   WITH RECURSIVE PPX (Major, IsNewArc, Minor, Qty) AS
   (SELECT Major, pm.Dummy, Minor, Qty
    from   PART_PART pp, PARTMASTER pm
    where  pp.Major = 'p1' and pp.Major = pm.Pno
           AND pm.Dummy = 'нет'
      UNION ALL
    SELECT CASE pm.Dummy
                WHEN 'да' THEN ppx.Major ELSE pp.Major
           END,                                        --- Major

              pm.Dummy,                                  ---
IsNewArc
           pp.Minor,                                   --- Minor
           CASE pm.Dummy
                WHEN 'да' THEN ppx.Qty * pp.Qty ELSE pp.Qty
           END                                         --- Qty
    FROM   ppx, PART_PART pp, PARTMASTER pm
    WHERE pp.Major = ppx.Minor and pp.Major = pm.Pno
     )
   SELECT DISTINCT ppx.Major, ppx.Minor, ppx.Qty,
ppx.IsNewArc
FROM   ppx, Partmaster pm
WHERE  ppx.Minor = pm.Pno AND pm.Dummy = 'нет'
```

Результат вычисления рекурсивного запроса следующий:

© Резниченко В.А., 2011

```
MAJOR ISNEWARC MINOR   QTY
----- -------- ----- -----
p1    нет      p2        2
p1    да       p3       10
p1    нет      p3        3
p1    нет      p4        2
p1    да       p5       40
p1    да       p5       12
p1    да       p6        4
p1    да       p6        6
p1    да       p6       20
p2    нет      p3        3
p2    нет      p5        2
p2    да       p5       12
p2    да       p6        6
13 строк выбрано.
```

Здесь каждая пара деталей стоится по следующему принципу:

- первая деталь не является фиктивной, а вторая может быть фиктивной;
- пары деталей имеют непосредственную связь, или между ними находится любое количество фиктивных деталей.

Столбец ISNEWARC указывает, имеются ли между соответствующими парами фиктивные вершины. Столбец QTY дает количество второй детали в первой с учетом одного из возможных путей их связи.

Наконец, основной запрос отбирает из результата рекурсивного запроса такие строки, у которых вторая деталь не является фиктивной:

```
MAJOR MINOR   QTY ISNEWARC
----- ----- ----- --------
p1    p2        2 нет
p1    p5       40 да
p1    p5       12 да
p1    p6        4 да
p1    p6       20 да
p1    p6        6 да
p2    p5       12 да
p2    p5        2 нет
p2    p6        6 да
9 строк выбрано.
```

2.4 В каких агрегатах используется деталь?

Интересной задачей является определение того, в каких именно агрегатах используется заданная деталь или агрегат. Фактически это означает построения такого подмножества заданной структуры агрегирования, которая содержит все пути, ведущие от корневой вершины в заданную.

Ниже приводится рекурсивный запрос, вычисляющий все пары вершин, лежащие на путях из корневой вершины в вершину p5:

© Резниченко В.А., 2011

```
WITH RECURSIVE PX (Major, Minor) AS
  (SELECT Major, Minor
   FROM   PART_PARP pp
   WHERE  pp.Minor = 'p5'
     UNION
   SELECT pp.Major, pp.Minor
   FROM   PX, PART_PART pp
   WHERE  pp.Minor = PX.Major)
SELECT Major, Minor FROM PX;
```

```
Результат вычисления запроса следующий:
MAJOR MINOR
----- -----
p2    p5
p3    p5
p1    p3
p2    p3
p4    p3
p1    p2
p1    p4
8 строк выбрано.
```

В запросе мы использовали UNION, чтобы в ответе отсутствовали одинаковые пары вершин, если одни и те же ребра присутствуют в различных путях. Если дубликаты нужны, то следует использовать UNION ALL.

3. Характеристики поиска

Конструкция характеристики_поиска задается во фразе WITH только при определении рекурсивного запроса. Она позволяет специфицировать направление поиска по иерархической структуре и/или реагирование на наличие циклов и имеет следующий синтаксис:

```
фраза_фиксации_цикла |
фраза_направления_поиска |
фраза_фиксации_цикла
фраза_направления_поиска
```

Рассмотрим эти фразы.

3.1 Фиксация наличия циклов

Рекурсия может иметь бесконечные циклы. Приводимая ниже фраза служит для предотвращения циклов:

```
CYCLE список_столбцов_цикла
SET имя_столбца_отметки_цикла TO значение_отметки_цикла
DEFAULT значение_отметки_отсутствия_цикла USING столбец_пути
```

Параметры этой фразы означают следующее:

- параметр список_столбцов_цикла указывает, на значениях каких столбцов следует выявлять условие наличие цикла.
- с помощью имя_столбца_отметки_цикла задается имя дополнительного явно не определенного столбца результирующей таблицы, в котором будет формироваться признак того, что обнаружен цикл.
- значение_отметки_цикла указывает, какое именно значение в этом столбце соответствует этой отметке. Кроме того, с помощью фразы DEFAULT задается значение столбца отметки цикла, которое свидетельствует, что цикл не обнаружен.
- с помощью фразы USING задается имя еще одного дополнительного столбца результирующей таблицы, который будет использоваться для сохранения информации, необходимой для выявления наличия цикла.

Приведенный далее пример может иметь бесконечную петлю, если сами данные носят циклический характер (обратите внимание, что в запросе мы используем предложение UNION ALL).

© Резниченко В.А., 2011

```
WITH RECURSIVE PX(Major, Minor) AS
(SELECT Major, Minor FROM PART_PART WHERE Major = 'p1'
   UNION ALL
 SELECT PX.Major, PART_PART.Minor
 FROM   PX, PART_PART WHERE PART_PART.Major = PX.Minor)
SELECT Major, Minor FROM PX
```

Как показано в следующем запросе, фраза CYCLE может использоваться для предотвращения зацикливания выполнения запроса:

```
WITH RECURSIVE PX(Major, Minor) AS
(SELECT Major, Minor FROM PART_PART WHERE Major = 'p1'
   UNION ALL
 SELECT PX.Major, PART_PART.Minor
 FROM   PX, PART_PART WHERE PART_PART.Major = PX.Minor)
CYCLE Minor SET CycleMark TO '1' DEFAULT '0' USING Path
SELECT Major, Minor FROM PX
```

Чтобы понять смысл запросов, подобных предыдущему, содержащих фразу CYCLE, авторы рекурсивного SQL предложили синтаксические правила определения эквивалентных запросов без фразы CYCLE. Для предыдущего запроса эквивалентным является приведенный ниже. В нем добавленный текст приведен жирным шрифтом:

```
WITH RECURSIVE PX(Major, Minor, CycleMark, Path) AS
(SELECT Major, Minor, '0', LIST{ROW(Minor)}
 FROM PART_PART WHERE Major = 'p1') AS CRN1(Major, Minor)
   UNION ALL
 SELECT Major, Minor,
        CASE WHEN ROW(Minor)IN (SELECT P.* FROM
TABLE(Path) P)
             THEN '1' ELSE '0' END,
        CONCATENATE(Path, LIST {ROW(Minor)})
 FROM   (SELECT PART_PART.Major, PART_PART.Minor,
                PX.CycleMark, PX.Path
         FROM   PX, PART_PART
         WHERE  PART_PART.Major = PX.Minor) AS
                   CRN2(Major, Minor, CycleMark, Path)
 WHERE CycleMark <> '1'
)
SELECT Major, Minor, CycleMark FROM PX
```

Например, если таблица PPA имеет следующее содержимое:

MAJOR	MINOR
p1	p2
p1	p3
p1	p4
p3	p5
p3	p6
p6	p2
p6	p3
p6 p4	

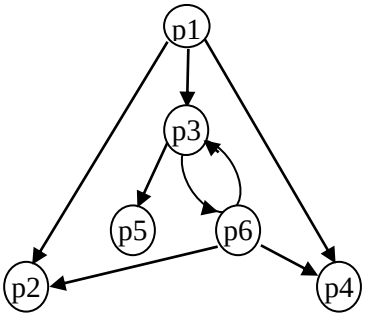

Рис. 4 Граф-схема таблицы PPA

то приводимый далее запрос:

```
   WITH RECURSIVE PX (Major, Minor, CycleMark, Path) AS
   (SELECT Major, Minor, '0', LIST {(Major, Minor)}
    FROM   PPA WHERE PPA.Major = 'p1'
      UNION ALL
    SELECT PPA.Major, PPA.Minor,
           CASE WHEN (PPA.Major,PPA.Minor) IN
                       (SELECT *
                        FROM   TABLE(LIST_APPEND{Path,
                                     (PPA.Major,PPA.Minor)}))
                THEN '1' ELSE '0'
           END,
           LIST_APPEND{Path, (PPA.Major, PPA.Minor)}
    FROM  PX, PPA
    WHERE PPA.Major = PX.Minor AND PX.Major <> 'p2' AND
          PX.CycleMark = '0'
)
SELECT Major, Minor, CycleMark, Path FROM PX
ORDER BY Major, Minor DESC;
```

создает следующую таблицу:

```
   MAJOR MINOR CYCLEMARK PATH
   p1    p4    0         {(p1, p4 )}
   p1    p3    0         {(p1, p3 )}
   p1    p2    0         {(p1, p2 )}
   p3    p6    0         {(p1, p3 ),(p3, p6 )}
   p3    p6    1         {(p1, p3 ),(p3, p6 ),(p6, p3 ),(p3,
   p6 )}
   p3    p5    0         {(p1, p3 ),(p3, p5 )}
   p3    p5    0         {(p1, p3 ),(p3, p6 ),(p6, p3 ),(p3,
   p5 )}
   p6    p4    0         {(p1, p3 ),(p3, p6 ),(p6, p4 )}
   p6    p3    0         {(p1, p3 ),(p3, p6 ),(p6, p3 )}
   p6    p2    0         {(p1, p3 ),(p3, p6 ),(p6, p2 )}
```

Таким образом, по наличию 1 в CYCLEMARK для пары (p3, p6) можно определить наличие цикла.

3.2Направление поиска

Рекурсивные запросы работают с иерархическими структурами данных. В этом случае имеется два способа поиска данных в такой структуре (обхода данных в структуре): в глубину и в ширину, как это показано на рис. 5.

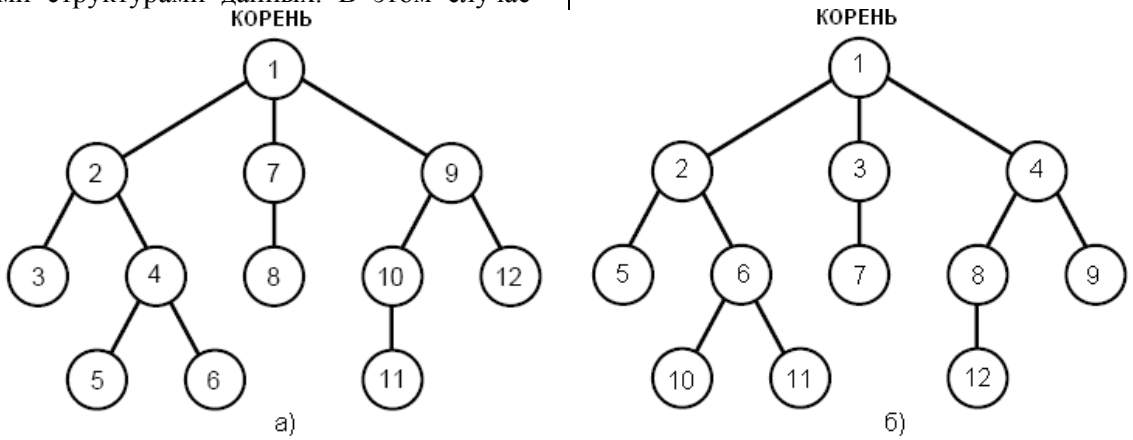

Рис. 5. Варианты направлений поиска данных: а) в глубину; б) в ширину

Синтаксис рекурсивного запроса предоставляет возможность указать, какой из этих двух вариантов следует применить. Для этого используется следующая конструкция направления поиска:

```
SEARCH {DEPTH | BREADTH} FIRST BY список_спецификации_сортировки
SET столбец_уровня
```

Ключевые слова `{DEPTH | BREADTH} FIRST` указывают, будет ли производиться поиск в глубину или ширину. Фраза `BY` используется для указания упорядоченности строк таблицы, используемой для перехода от одной строки к другой в пределах одного уровня (поиск в ширину) или выбора строки на следующем уровне иерархии (поиск в глубину). Список спецификации сортировки идентичен списку сортировки фразы `ORDER BY`. Наконец, фраза SET предоставляет возможность указать имя дополнительного столбца результирующей таблицы, в которой будет формироваться номер уровня расположения очередной вершины иерархической структуры данных.

Приведем пример:

```
WITH RECURSIVE PX(Major, Minor) AS
 (SELECT Major, Minor FROM PPA WHERE Major = 'p1'
    union all
  SELECT PPA.Major, PPA.Minor
  FROM PX, PPA WHERE PPA.Major = PX.Minor)
  SEARCH BREADTH FIRST BY MAJOR, MINOR SET
ORDERCLMN
SELECT Major, Minor FROM PX
ORDER BY ORDERCLMN;
```

4. Более сложная структура рекурсивного запроса

До сих пор мы рассматривали довольно простую структуру рекурсивных запросов. Приведем примеры более сложных структур.

Первый вид усложнения заключается в использовании в рекурсивных запросах многих операторов над множествами, как это показывается ниже:

```
WITH RECURSIVE result_table (...) AS
  (операнд_0
     <оператор_над_множествами_1>
   операнд_1
     <оператор_над_множествами_2>
   операнд_2
     <оператор_над_множествами_3>
   ...
   операнд_n-1
     <оператор_над_множествами_n>
   операнд_n
  )
SELECT ... FROM result_table;
```

Здесь предполагается, что операнд_0 обязательно является не рекурсивной частью запроса.

Далее, рекурсивные запросы могут вкладываться, например, так, как это демонстрируется далее:

```
WITH RECURSIVE T1 (...) AS
 (SELECT ... FROM ... operand 0
    UNION                    set operator 1
  SELECT ... FROM T1, T2 ... operand 1
    UNION                    set operator 2
   (WITH RECURSIVE T3 (..) AS
       (SELECT ... FROM T4 operand 3
          UNION                  set operator 4
        SELECT ... FROM T3, T5 operand 4
          UNION                  set operator 5
        SELECT ... FROM T1, T6 operand 6
        )
    )
 )
SELECT ... FROM T1;
```

5.Рекурсивные представления

Рекурсивными также могут быть представления. Для указания того, что представление является рекурсивным, следует следующим образом использовать слово RECURSIVE:

```
CREATE RECURSIVE VIEW имя_представления [(список_имен_столбцов)]
AS запрос [WITH [CASCADE | LOCAL] CHECK OPTION];
```

Чтобы он действительно стал рекурсивным, его запрос должен иметь вид запроса фразы WITH рекурсивного запроса. Например, представление, которое соответствует вычислению запроса: «Найти все города, к которым можно долететь из заданного города», имеет следующий вид:

```
CREATE RECURSIVE view Reachable_From (FromCity, ToCity)
AS
   SELECT FromCity, ToCity
   FROM   Flights
     UNION
   SELECT inn.FromCity, out.ToCity
   FROM   Reachable_From inn, Flights out
   WHERE  inn.FromCity = out.ToCity;
```

Теперь, если необходимо найти города, к которым можно долететь из Киева, надо следующим образом воспользоваться созданным представлением:

```
SELECT *
FROM   Reachable_From
WHERE  Source = 'Киев';
```

Приведем альтернативный синтаксический вариант описания рекурсивного представления. Он имеет следующую форму:

```
CREATE VIEW имя_представления AS
WITH RECURSIVE имя_таблицы (список_имен_столбцов)AS (запрос)
SELECT список_имен_столбцов FROM имя_таблицы
```

По сути это означает, что можно не использовать фразу RECURSIVE в заголовке определения представления, но при этом в качестве его запроса следует использовать полный синтаксис рекурсивного запроса. Так, например, запрос: «Найти наиболее дорогостоящие перелеты между парами городов» может быть представлен с помощью следующего представления:

```
CREATE view REACHABLE_VIEW (FromCity, ToCity, MaxCost) AS
WITH RECURSIVE REACHABLE_FROM (FromCity, ToCity, Total_Cost) AS
   (SELECT FromCity, ToCity, Cost
    FROM   FLIGHTS
      UNION
    SELECT inn.FromCity, out.ToCity, inn.Total_Cost + out.Cost
    FROM   REACHABLE_FROM inn, Flights out
    WHERE  inn.ToCity = out.FromCity
   )
SELECT FromCity, ToCity, MAX(Total_Cost)
FROM   REACHABLE_FROM
GROUP BY inn.FromCity, out.ToCity;
```

Тогда для получения максимальной стоимости перелета из Москвы в Симферополь следует воспользоваться следующим запросом:

```
SELECT MaxCost FROM REACHABLE_VIEW
WHERE  FromCity = 'Москва' AND ToCity = 'Симферополь';
```

6. Различные виды рекурсии

Существуют две категории рекурсии: категория взаимности и категория линейности. Категория взаимности включает прямую и взаимную рекурсию, а категория линейности: линейную и нелинейную. Все эти виды рекурсии представимы в SQL и они обсуждаются далее.

6.1 Прямая и взаимная рекурсия

Все рассмотренные до сих пор запросы относятся к разновидности прямой (непосредственной) рекурсии. Суть *прямой рекурсии* заключается в том, что таблица рекурсивно определяется через саму себя. Определение таблиц посредством *взаимной рекурсии* заключается в том, что, например, таблица A определяется

через таблицу B, а таблица B, в свою очередь, через таблицу A.

Например, определение двух представлений EVEN(N) и ODD(M), содержащих четные и нечетные числа, выражается через следующие взаимно-рекурсивные определения:

```
CREATE RECURSIVE view EVEN(N) AS
  (VALUES(0) UNION SELECT M + 1 FROM ODD)
CREATE RECURSIVE view ODD(M) AS
  (SELECT N + 1 FROM EVEN )
```

Примечание VALUES(0) – это конструктор таблицы, состоящей из одного столбца, содержащего единственное значение – 0.

Тогда при следующем обращении к представлению EVEN(N):

```
SELECT * FROM EVEN WHERE N < 12;
```

мы получим числа: 0, 2, ..., 10.

То же самое можно записать с помощью фразы WITH следующим образом:

```
with RECURSIVE
 EVEN (N) AS (VALUES(0) UNION SELECT M + 1 FROM ODD),
 ODD(M)   AS SELECT N + 1 FROM EVEN)
SELECT * FROM EVEN WHERE N < 12;
```

Давайте рассмотрим еще один более традиционный пример. Предположим, что имеются две следующие таблицы:

EMP (EmpId, EmpName, DeptId) - таблица служащих, содержащая идентификатор служащего (EmpId), его имя (EmpName) и ссылку на его отдел (DeptId),

DEPT (DeptId, MgrId) - таблица отделов, содержащая идентификатор отдела (DeptId) и ссылку на его менеджера (MgrId) из таблицы служащих.

Таблицы определены таким образом, что для нахождения ID менеджера служащего, перед которым он (служащий) подотчетен (таблица REPORTS_TO), необходимо обратиться к таблице отделов. Далее, для получения следующей итерации подотчетности удобно иметь таблицу BELONGS_TO, которая для каждого менеджера содержит все отделы, за которые он отвечает. Для построения этой таблицы на основании таблицы DEPT необходима также таблица REPORTS_TO. Таким образом, эти две таблицы определяются рекурсивно. Обратим ваше внимание на то, что для вычисления рекурсии необходимы обе таблицы, даже если результирующий запрос использует информацию только из одной из них. Вот как выглядит эта рекурсия:

```
  with RECURSIVE
REPORTS_TO (EmpId, EmpName, MgrId) AS
   (SELECT EmpId, EmpName, MgrId
    FROM   DEPT, EMP
    WHERE  DEPT.DeptId = EMP.DeptId
        UNION
    SELECT e.EmpId, e.EmpName, b.MgrId
    FROM EMP e, BELONGS_TO b
    WHERE e.DeptId = b.DeptId
      ),
```

```
BELONGS_TO(DeptId, MgrId) AS
 (SELECT d.DeptId, r.MgrId
  FROM DEPT d, REPORTS_TO r
  WHERE d.MgrId = r.EmpId
 )
SELECT * FROM REPORTS_TO;
```

6.2 Линейная и нелинейная рекурсии

Линейная рекурсия

Все рассмотренные до сих пор рекурсивные запросы были линейными. Рекурсивный запрос является *линейным*, если таблица, определяемая рекурсивно, соединяется в запросе только один раз (то есть она перечисляется во фразе FROM только один раз). Кроме того, линейная рекурсия предполагает, что рекурсивно определяемая таблица не может одновременно использоваться во фразах FROM как самого рекурсивного запроса, так и во всех его подзапросах. Эти же правила применяются для взаимно определяемых рекурсивных таблиц. Если одно из этих условий нарушается, то рекурсия является нелинейной. Отметим, что многократная ссылка на рекурсивную таблицу не обязательно означает, что рекурсия нелинейная. Приведем пример. Пусть имеется таблица TRAINS, имеющая столбцы FromCity и ToCity и указывает пары городов, между которыми имеется железнодорожная связь. Тогда запрос: «Найти все города, достижимые из г. Васильков либо по железной дороге, либо авиатранспортом» имеет следующий вид:

```
WITH RECURSIVE REACHABLE_BY_TRAIN_PLANE (FromCity, ToCity) AS
(SELECT f.FromCity, f.ToCity
 FROM   FLIGHTS f
 WHERE  f.FromCity = 'Васильков'
     UNION ALL
  (SELECT t.FromCity, t.ToCity
   FROM TRAINS t
   WHERE t.FromCity = 'Васильков'
       UNION ALL
   SELECT rtp.FromCity, f.ToCity
   FROM REACHABLE_BY_TRAIN_PLANE rtp, FLIGHTS f
   WHERE rtp.ToCity=f.FromCity
       UNION ALL
   SELECT rtp.FromCity, t.ToCity
   FROM REACHABLE_BY_TRAIN_PLANE rtp, TRAINS t
   WHERE rtp.ToCity=t.FromCity
  )
 )
SELECT * FROM REACHABLE_BY_TRAIN_PLANE;
```

Это линейный запрос не смотря на то, что в нем имеется две ссылки на рекурсивно определяемую таблицу REACHABLE_BY_TRAIN_PLANE.

Обратим ваше внимание еще на одну особенность этого запроса, которая не имеет отношение к линейности рекурсии. Все до сих пор приводимые рекурсивные запросы, относящиеся к таблице FLIGHTS, имеют типичное рекурсивное определение. Имеется в виду, что вычисления начинаются с некоторой исходной совокупности строк таблицы FLIGHTS, которые выполняют роль так называемого базиса рекурсивного вычисления. Затем этот базис расширяется дополнительными строками в каждом цикле рекурсии. Это обычный случай рекурсии. Тем не менее, как видно из предыдущего запроса, возможны варианты, когда базис рекурсивного вычисления пустой. Так как г. Васильков не имеет пассажирского аэропорта, то первый маршрут путешествия будет выполнен железнодорожным транспортом, и базис рекурсивного запроса окажется пустым.

Нелинейная рекурсия

Примером нелинейной рекурсии может быть запрос, строящий последовательность чисел Фибоначчи. Напомним, что последовательность чисел называется последовательностью Фибоначчи, если каждый член этой последовательности равен сумме двух предшествующих членов. Начальные числа такой последовательности равны `1, 1, 2, 3, 5, 8, 13,…`

Рекурсивное представление, строящее такую последовательность, имеет следующий вид:

```
CREATE RECURSIVE view Fibonacci (N, F) AS
 VALUES(0,0), (1,1)
    UNION
 SELECT N + 1, p.F + pp.F
 FROM   Fibonacci p, Fibonacci pp
 WHERE  (p.N - 1) = pp.N;
```

Создаваемая таблица `Fibonacci` в первом столбце содержит порядковые номера строк, которые используются в рекурсивном вычислении, а второй столбец – собственно числа Фибоначчи.

Это рекурсивное определение является нелинейным, так как рекурсивно создаваемая таблица Fibonacci соединяется сама с собой.

Отметим, что нелинейные рекурсивные запросы являются довольно редкими и в большинстве случаев они могут выражаться линейными запросами. Например, вычисление чисел Фибоначчи может быть произведено с помощью следующего линейного рекурсивного представления:

```
CREATE RECURSIVE view Fibonacci2(N1, F1, N2, F2) AS
 VALUES(0,0, 1,1)
     UNION
 SELECT N1 + 1, F1 + F2, N1 +2, F1 + 2 * F2
 FROM   Fibonacci2 p
```

Наконец, следует заметить, что описанные выше две категории рекурсий, взаимности и линейности, являются полностью независимыми друг от друга. Рекурсивный запрос может быть прямо или взаимно рекурсивным и в то же самое время – либо линейным либо нелинейным.

Литература

1. *Пасічник В.В., Резніченко В.А.* Організація баз даних та знань. – К.: Видавнича група BHV, 2006. – 384 с.

2. Expressing Recursive Queries in SQL, ANSI Document X3H2-96-075r1, 1996. (with S. Finkelstein, N. Mattos, and H. Pirahesh) . - http://www.kirusa.com/mumick/papers/pspapers/ansiRevisedRecProp96-075r1.ps.Z

Сведения об авторе:

Резниченко Валерий Анатольевич - к.ф-.м.н., с.н.с., ведущий научный сотрудник Института программных систем НАН Украины, область научных интересов - информационные системы, базы данных и знаний, электронные библиотеки
E-mail: vreznichenko_47@mail.ru

Статья поступила в редакцию 15.11.2010 г.

© Резниченко В.А., 2011