УДК 004.05

СПОСОБЫ РЕЛЯЦИОННОГО МОДЕЛИРОВАНИЯ ИЕРАРХИЧЕСКИХ СТРУКТУР ДАННЫХ

С. В. Тарасов,
ведущий инженер исследований и разработки
Bel Air Informatique, Париж, Франция
В. В. Бураков,
доктор техн. наук, профессор
Санкт-Петербургский государственный университет аэрокосмического приборостроения

Данные различных предметных областей часто имеют связи иерархического характера. Наиболее надежным способом хранения информации до сих пор являются реляционные базы данных. Реляционный способ моделирования в чистом виде не поддерживает иерархические типы данных. Распространенность иерархических структур в задачах автоматизации обосновывает актуальность поиска способов эффективного отображения древовидных данных в реляционную модель. В статье рассматриваются способы представления структур иерархического типа в реляционных базах данных и типовые запросы к этим структурам хранения. Описаны критерии количественной оценки способов хранения и их сравнительные характеристики.

Ключевые слова *— иерархические структуры данных, реляционные базы данных.*

Введение

В области баз данных обычно различают три вида проектирования: концептуальное, логическое и физическое [1, 2]. Логическое проектирование позволяет выделить и обобщить типовые способы, независимые как от предметных областей (концептуальное проектирование), так и от деталей реализации и операционных сред (физическое проектирование).

Проектировщику часто приходится сталкиваться с древовидными структурами, представляющими собой граф без циклов. В общем случае моделирование сводится к многоуровневой связи «главный — подчиненный», «предок — потомок», «общий — конкретный». Реляционный способ моделирования не обладает какими-либо специализированными механизмами для адекватного представления иерархической информации. Распространенность иерархических структур в реальном мире обосновывает актуальность поиска способов эффективного отображения древовидных данных в реляционную модель.

При рассмотрении общих способов представления древовидных структур в реляционных базах данных в качестве примера использована СУБД Microsoft SQL Server (для типовых операций используется синтаксис MS SQL Server 2005). Вместе с тем все предложенные способы являются независимыми от реализации, поскольку описываются на уровне логического проектирования и могут быть воспроизведены на других СУБД, поддерживающих реляционную модель.

Используемые понятия теории графов

Графы широко используются для моделирования различных концепций теории программирования [3, 4]. Основные понятия описаны в специальной литературе [5—8], здесь перечислены только определения, используемые далее в тексте статьи. Дерево — это связный ациклический граф [5]. Связность означает наличие путей (простой цепи) между любой парой вершин, ацикличность — отсутствие циклов, т. е. между любыми парами вершин имеется только по одному пути. Граф порядка N называется помеченным, если его вершинам присвоены некоторые метки, например, номера 1, 2, ..., N. Пара вершин u и v графа является смежной, если множество $\{u, v\}$ является ребром. Произвольный помеченный граф порядка N можно представить в виде квадратной бинарной матрицы $N \times N$, где на пересечении i-й строки и j-го столбца стоит 1, вершины i и j смежны, или 0 в противном случае. Такое

представление называется матрицей смежности [5]. Граф является ориентированным (орграф), если все его ребра ориентированы, т. е. смежные вершины разделены на начальную и конечную. Ориентированное ребро называется дугой. В ориентированном дереве имеется только одна вершина с нулевой степенью захода (в нее не ведут дуги), она называется корнем дерева, а все остальные вершины имеют степень захода 1 (в них ведет ровно по одной дуге). Вершины с нулевой степенью исхода (из которых не исходит ни одна дуга) называются концевыми вершинами или листьями. Если в дереве существует путь из вершины a к вершине b, то вершина a называется предком вершины b, а вершина b — потомком вершины a. Путем из вершины n_i к вершине n_j называется последовательность вершин, где для всех k, $i \le k \le j$, узел n_k является предком узла n_{k+1}. Длиной пути называется число, на единицу меньшее числа составляющих этот путь вершин. Высотой вершины называется длина самого длинного пути из этой вершины до какого-либо листа. Высота дерева совпадает с высотой корня. Глубина вершины определяется как длина пути (который единственный в дереве) от корня до этой вершины. N-арное дерево (ориентированное) — это ориентированное дерево, в котором число исходящих дуг для любой вершины не превосходит N. Вершина графа также имеет часто используемый синоним «узел». Д. Кнут приводит следующие критерии сбалансированности деревьев. N-арное дерево является сбалансированным, если выполнены следующие условия [7, с. 515]: каждый узел имеет не более N потомков; каждый узел, за исключением корня и листьев, имеет не менее $N/2$ потомков; корневой узел, если он не является листом, имеет не менее двух потомков; все листья расположены на одном уровне (имеют одинаковую глубину). Более краткое определение сформулировано Г. Адельсон-Вельским и Е. Ландисом [8, с. 248]. Дерево является сбалансированным тогда и только тогда, когда для каждого узла высота его двух поддеревьев различается не более чем на 1. Дерево называется идеально сбалансированным, если для каждой его вершины количества вершин в любом поддереве различаются не более чем на 1 [8, с. 228]. В настоящей статье рассматриваются только ориентированные деревья.

Способ «Список смежности»

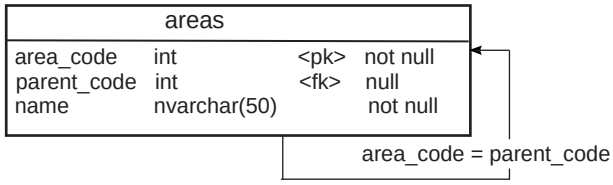
В реляционной модели матрица смежности может быть представлена в виде множества (списка) пар с номерами (идентификаторами, кодами) вершин, где каждая пара определяет ориентированную дугу между вершинами. Способ представляет собой интуитивно понятную организацию иерархии в виде таблицы с замкнутой на саму себя связью (рефлексивная связь).

Корневые вершины отличаются от других пар пустой (NULL) ссылкой на предка. В приведенном примере (рис. 1, табл. 1, 2) это поле «Код вышестоящей территории».

Число необходимых элементов для представления матрицы в виде списка и соответствующих ему строк таблицы будет равно порядку графа (числу вершин графа), так как в каждую вершину, кроме корневой, входит только одна дуга. Обозначим это число элементов как S_N, где N — число вершин в дереве:

$$S_N = N. \quad (1)$$

Для выполнения часто используемых выборок требуется поддержка рекурсивных запросов. Если СУБД не умеет выполнять такие запросы,



■ ***Рис. 1.*** Пример представления данных согласно способу «Список смежности»

■ ***Таблица 1.*** Пример заполнения таблицы areas («Список смежности»)

area_code	parent_code	name
1	NULL	Санкт-Петербург
2	1	Центральный район
3	1	Московский район
4	1	Невский район
5	3	МО Новоизмайловское
6	3	МО Кузнецовское
7	4	МО Рыбацкое

■ ***Таблица 2.*** Представление в виде матрицы смежности

	1	2	3	4	5	6	7
1	0	1	1	1	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

то выборки придется строить с использованием других механизмов, например, временных таблиц или хранимых процедур и функций. Рассмотрим примеры запросов.

Типовые запросы

Листинг 1. Выборка поддерева

Колонка level отображает глубину узла выбранного поддерева.

```
WITH subtree (area_code, parent_code, name, level)
AS (SELECT area_code, parent_code, name, 0 AS level
  FROM  areas
  WHERE  area_code = 3 /* код корня */
  UNION ALL
  SELECT areas.area_code, areas.parent_code, areas.name, level + 1
  FROM  areas INNER JOIN subtree
        ON areas.parent_code = subtree.area_code
  WHERE  areas.parent_code IS NOT NULL
)
SELECT area_code,  name, level
FROM  subtree;
```

Результат выборки поддерева:

area_code	name	level
3	Московский район	0
5	МО Новоизмайловское	1
6	МО Кузнецовское	1

Листинг 2. Выборка всех предков (путь к узлу от корня)

```
WITH  subtree (area_code, parent_code, name, level)
AS   (SELECT area_code, parent_code, name, 0 AS level
     FROM  areas
     WHERE  area_code = 5 -- узел
     UNION ALL
     SELECT areas.area_code, areas.parent_code, areas.name, level + 1
     FROM  areas
          INNER JOIN subtree
          ON areas.area_code = subtree.parent_code)
SELECT area_code,  name,
    (SELECT MAX(level) FROM subtree) - level + 1 AS level
FROM  subtree
ORDER BY level;
```

Результат выборки предков:

area_code	name	level
1	Санкт-Петербург	0
3	Московский район	1
5	МО Новоизмайловское	2

Листинг 3. Проверка, входит ли узел в поддерево

```
WITH  subtree (area_code, parent_code)
AS   (SELECT area_code, parent_code
     FROM  areas
     WHERE  area_code = 5 /* узел, проверяемый на вхождение */
     UNION ALL
     SELECT areas.area_code, areas.parent_code
     FROM  areas
          INNER JOIN subtree
          ON areas.area_code = subtree.parent_code)
SELECT CASE
     WHEN EXISTS (SELECT 1
               FROM  subtree
               WHERE  area_code = 3 /* корень поддерева */)
     THEN N'Входит'
     ELSE N'Не входит'
END AS result;
```

Результат проверки вхождения узла:
result = Входит

Листинг 4. Подсчет количества всех потомков узла

Запрос похож на выборку поддерева с последующим подсчетом количества выбранных узлов.

```
WITH subtree (area_code, parent_code)
AS (SELECT area_code, parent_code
  FROM  areas
  WHERE  parent_code = 3 /* код корня */
  UNION ALL
  SELECT areas.area_code, areas.parent_code
  FROM  areas INNER JOIN subtree
        ON areas.parent_code = subtree.area_code
  WHERE  areas.parent_code IS NOT NULL
)
SELECT COUNT(1) AS qty
FROM  subtree;
```

Результат подсчета количества потомков узла:
qty = 2

Листинг 5. Определение высоты узла

```
WITH  subtree (area_code, parent_code, level)
AS   (SELECT area_code, parent_code, 0 AS level
     FROM  areas
     WHERE  area_code = 5 -- узел
     UNION ALL
     SELECT areas.area_code, areas.parent_code, level + 1
     FROM  areas INNER JOIN subtree
          ON areas.area_code = subtree.parent_code)
SELECT MAX(level) AS level
FROM  subtree;
```

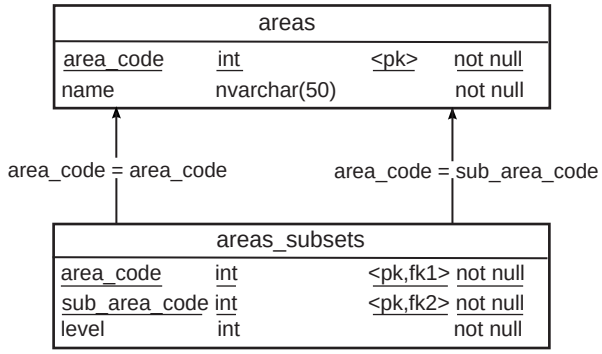
Результат определения абсолютного уровня:
level = 2

Способ «Подмножества»

В этом способе дерево представляется вложенными подмножествами. Корневой уровень включает в себя все подмножества — узлы первого уровня. Узлы первого уровня в свою очередь включают в себя все узлы второго уровня и т. д.

Например, иерархия административно-территориального деления муниципалитета может выглядеть следующим образом (рис. 2).

■ ***Рис. 2.*** Представление иерархии в виде вложенных подмножеств



■ ***Рис. 3.*** Пример представления данных согласно способу «Подмножества»

■ ***Таблица 3.*** Пример заполнения таблицы areas («Подмножества»)

area_code	name
1	Санкт-Петербург
2	Московский район
3	МО Новоизмайловское
4	МО Кузнецовское
5	Невский район
6	МО Рыбацкое
7	Центральный район

В терминах реляционной модели схема будет выглядеть, как на рис. 3 (пример заполнения — в табл. 3).

В способе «Подмножества» каждый элемент, кроме ссылки на непосредственных потомков, содержит ссылки и на потомков всех последующих уровней иерархии (табл. 4).

Если представить граф в матричном виде, то на пересечении i-й строки и j-го столбца стоит 1, если вершины i и j смежны; $n < N$, если существует путь от i к j длиной n, или 0 в остальных случаях. Назовем такую матрицу смежности расширенной (табл. 5).

Очевидно, что по сравнению с обычной матрицей смежности (см. табл. 2) расширенная матрица содержит избыточные значения $n > 1$. Количественная оценка избыточности будет напрямую зависеть от сбалансированности дерева.

■ ***Таблица 4.*** Пример заполнения таблицы area_subsets («Подмножества»)

area_code	sub_area_code	level
1	1	0
1	2	1
1	3	2
1	4	2
1	5	1
1	6	2
1	7	1
2	2	0
2	3	1
2	4	1
3	3	0
4	4	0
5	5	0
5	6	1
6	6	0
7	7	0

■ ***Таблица 5.*** Представление в виде расширенной матрицы смежности

	1	2	3	4	5	6	7
1	0	1	2	2	1	2	1
2	0	0	1	1	0	0	0
3	0	0	0	0	1	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

В случае совершенно несбалансированного дерева все вершины находятся на одном пути между корнем и единственным листом. Пусть вершины пронумерованы от 1 (корень) до N (лист). В этом случае расширенная матрица будет целиком заполнена выше главной диагонали. Количество элементов списка (и строк таблицы) будет равно сумме арифметической прогрессии от 0 (из листа не исходят дуги) до $N-1$ (из корня есть пути во все вершины кроме него самого):

$$S_N = ((N-1)/2)\,N. \qquad (2)$$

В случае идеально сбалансированного k-арного дерева каждая вершина, согласно определению, будет иметь не менее $(k/2)L$ и не более kL путей к потомкам, где L — высота вершины.

Распределение элементов по уровням показано в табл. 6.

■ ***Таблица 6.*** Распределение вершин в идеально сбалансированном дереве

Высота	Максимальное количество элементов уровня
0	$k^0 = 1$
1	$k^1 = k$
2	k^2
3	k^3
...	

Количество внешних вершин N_T не может превышать

$$N_T \le k^h, \quad (3)$$

где h — высота дерева. Тогда количество элементов списка (и строк таблицы) можно оценить так:

$$S_N \le (N - N_T) \cdot k/2. \quad (4)$$

Таким образом, оценка избыточности будет зависеть от сбалансированности дерева и варьироваться от (4) в наилучшем случае до (2) в наихудшем.

Избыточность данных приводит к необходимости дополнительной поддержки их целостности императивно, триггерами, перезаписывающими список и уровни предков данного узла при его вставке или перемещении. Для операции удаления достаточно декларативной ссылочной целостности (каскадное удаление).

Типовые запросы показывают преимущества, полученные от избыточности хранения: запросы стали короткими и простыми, не содержат соединения с другими таблицами, что обеспечивает их эффективность на любой СУБД.

Типовые запросы

Листинг 6. Выборка поддерева

```
SELECT  areas.area_code, areas.name, area_subsets.level
FROM   area_subsets INNER JOIN areas
       ON area_subsets.sub_area_code = areas.area_code
WHERE  area_subsets.area_code = 2 -- корень поддерева
ORDER BY level;
```

Результат выборки поддерева:

area_code	name	level
2	Московский район	0
3	МО Новоизмайловское	1
4	МО Кузнецовское	1

Листинг 7. Выборка предков

```
SELECT areas.area_code, areas.name,
    (SELECT MAX(s.level)
    FROM area_subsets s
    WHERE s.sub_area_code = area_subsets.sub_area_code)
     - area_subsets.level + 1 AS level
FROM  area_subsets
      INNER JOIN areas
      ON area_subsets.area_code = areas.area_code
WHERE  area_subsets.sub_area_code = 3 -- узел
ORDER BY level;
```

Результат выборки предков:

area_code	name	level
1	Санкт-Петербург	0
2	Московский район	1
3	МО Новоизмайловское	2

Листинг 8. Вхождение в поддерево

```
SELECT result = CASE
  WHEN EXISTS(
    SELECT 1 FROM area_subsets
     WHERE sub_area_code = 4 /* узел */
        AND area_code = 2 /* корень поддерева */)
  THEN N'Входит'
  ELSE N'Не входит'
END
```

Результат проверки вхождения узла:
result = Входит

Листинг 9. Подсчет количества всех потомков узла

```
SELECT COUNT(1) - 1 AS qty
FROM  area_subsets
WHERE area_code = 2 /* корень поддерева */
```

Результат подсчета количества потомков узла:
qty = 2

Листинг 10. Определение высоты узла

```
SELECT MAX(level) AS level
FROM  area_subsets
WHERE  sub_area_code = 4 /* узел */
```

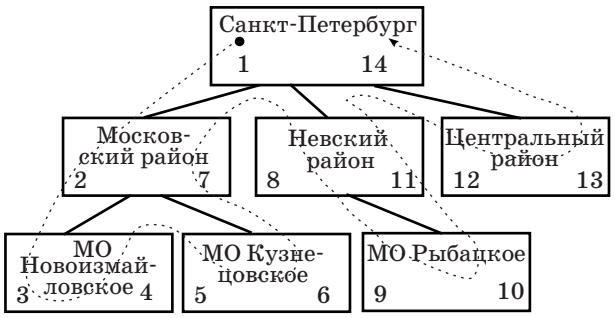
Результат определения абсолютного уровня:
level = 2

Способ «Маршрут обхода»

Согласно теории графов, для обхода дерева существует три способа: можно проходить узлы в префиксном, в инфиксном и в суффиксном порядке.

Префиксный порядок обхода дерева рекурсивно определяется так: сначала корень дерева, потом узлы левого поддерева в префиксном порядке, наконец, узлы правого поддерева в префиксном порядке. Пример обхода в префиксном порядке приведен на рис. 4.

Хранение маршрута обхода дерева в префиксном порядке также встречается у Джо Селко [9],



■ ***Рис. 4.*** Маршрут обхода дерева в префиксном порядке

areas			
area_code	int	<pk>	not null
input_code	int		not null
output_code	int		not null
name	nvarchar(50)		not null

■ ***Рис. 5.*** Пример представления данных согласно способу «Маршрут обхода»

■ ***Таблица 7.*** Пример заполнения таблицы areas («Маршрут обхода»)

area_code	name	input_code	output_code
1	Санкт-Петербург	1	14
2	Московский район	2	7
3	МО Новоизмайловское	3	4
4	МО Кузнецовское	5	6
5	Невский район	8	11
6	МО Рыбацкое	9	10
7	Центральный район	12	13

однако там способ носит не соответствующее его сути название «Вложенные множества» (nested sets).

Каждый квадрат на рис. 4 обозначает узел, цифра в левом его углу является порядковым номером этапа маршрута при входе в узел, а цифра справа — номером при выходе, когда тем же способом пройдены все потомки. Соответствующая структура таблицы показана на рис. 5, а пример содержания — в табл. 7. Нетрудно заметить, что номера потомков всегда располагаются в интервале между соответствующими номерами предка, сколь угодно дальнего. Храня порядок обхода дерева, этим свойством можно воспользоваться в типовых запросах, избежав рекурсии.

Оценка избыточности хранения по сравнению со способом «Список смежности» очевидна: вместо одного номера вершины предка каждый элемент матрицы будет хранить упорядоченную пару номеров:

$$S_N = N \cdot 2. \quad (5)$$

Избыточность хранения делает необходимым пересчет порядка обхода при добавлении новых или перемещении существующих узлов (удаление можно игнорировать). В триггере придется реализовать последовательный порядок обхода. Но, например, если добавляется элемент самого нижнего уровня, то придется пересчитать все номера «выше» или «правее», что может быть сравнимо с затратами на пересчет маршрута по всему дереву.

Если нумеровать входы и выходы из узлов с некоторым интервалом, например 100 или 1000, что в значительной степени зависит от предварительных оценок количества хранимых узлов дерева, то вставка новых элементов будет происходить без полной перенумерации всех последующих.

Типовые запросы

Листинг 11. Выборка поддерева

В запросе не вычисляется глубина узла. Добавление этой функции приведет к введению выполняющегося для каждого элемента агрегирующего подзапроса (см. листинг 15).

```
SELECT a1.area_code, a1.input_code, a1.name
FROM  areas a1
    INNER JOIN areas a2
    ON a1.input_code BETWEEN a2.input_code AND a2.output_code
WHERE  a2.area_code = 2 /* корень поддерева */
ORDER BY a1.input_code
```

Результат выборки поддерева:

area_code	input_code	name
2	2	Московский район
3	3	МО Новоизмайловское
4	5	МО Кузнецовское

Листинг 12. Выборка предков

Выборка всех предков симметрична предыдущему запросу относительно BETWEEN. В запросе не вычисляется глубина узла.

```
SELECT a1.area_code, a1.input_code, a1.name
FROM  areas a1 INNER JOIN areas a2
    ON a2.input_code BETWEEN a1.input_code AND a1.output_code
WHERE  a2.area_code = 4 /* узел */
ORDER BY a1.input_code
```

Результат выборки предков:

area_code	input_code	name
1	1	Санкт-Петербург
2	2	Московский район
4	5	МО Кузнецовское

Листинг 13. Вхождение в поддерево

```
SELECT result = CASE
  WHEN EXISTS(SELECT 1 FROM areas as a1, areas as a2
    WHERE a1.area_code = 4 /* узел */
      AND a2.area_code = 2 /* корень поддерева */
      AND a1.input_code BETWEEN a2.input_code AND a2.output_
code)
  THEN N'Входит'
  ELSE N'Не входит'
END
```

Результат проверки вхождения узла:
result = Входит

Листинг 14. Подсчет количества всех потомков узла

```
SELECT COUNT(1) AS qty
FROM (SELECT a1.area_code
FROM  areas a1 INNER JOIN areas a2
    ON a1.input_code BETWEEN a2.input_code AND a2.output_code
WHERE a2.area_code = 2 /* узел */ ) t
```

Результат подсчета количества потомков узла:
qty = 2

Листинг 15. Определение высоты узла

```
SELECT COUNT(1) AS level
FROM (SELECT a1.area_code
FROM  areas a1 INNER JOIN areas a2
    ON a2.input_code BETWEEN a1.input_code AND a1.output_code
WHERE a2.area_code = 4 /* узел */ ) t
```

Результат определения абсолютного уровня:
level = 1

■ ***Таблица 8***. Пример заполнения таблицы areas («Материализованные пути»)

area_path	name
1	Санкт-Петербург
1.1	Московский район
1.1.1	МО Новоизмайловское
1.1.2	МО Кузнецовское
1.2	Невский район
1.2.1	МО Рыбацкое
1.3	Центральный район

areas			
area_path	varchar(50)	<pk>	not null
name	nvarchar(50)		not null

■ ***Рис. 6***. Пример представления данных способом «Материализованные пути»

Способ «Материализованные пути»

Суть способа заключается в хранении пути от вершины до данного узла в явном виде и в качестве ключа. Например, ранее приведенная на рис. 2 иерархия территорий могла бы выглядеть следующим образом, схожим с нумерацией частей, разделов и глав в книге (табл. 8).

Данный способ является наиболее наглядным с точки зрения кодификации элементов: каждый узел получает значение, которое пригодно для непосредственного восприятия пользователем, сам код и его части несут смысловую нагрузку. Подобные свойства являются важными в классификациях, предназначенных для широкого использования, например, в стандартизованных справочниках территорий (ОКАТО), отраслей экономики (ОКВЭД, NAICS), медицинских диагнозов (МКБ — международный классификатор болезней) и во многих других областях.

Оценка избыточности хранения аналогична таковой для способа «Подмножества»: расширенная матрица смежности содержит те же элементы (рис. 6).

Ограничив максимальное количество уровней иерархии и число прямых потомков, можно обойтись без разделителей, используя символьные коды с фиксированной разбивкой на группы разрядов. Пустые лидирующие разряды в группе заполняются нулями.

Однако запросы не всегда могут быть эффективно реализованы на уровне СУБД, так как, например, поиск подстроки вызывает сканирование таблицы вместо поиска по ключу или его начальному фрагменту.

Листинг 16. Выборка поддерева

В запросе не вычисляется глубина узла.

```
SELECT  *
FROM    areas
WHERE   area_path LIKE '1.1%' /* корень поддерева */
ORDER BY area_path
```

Результат выборки поддерева:

area_path	name
1.1	Московский район
1.1.1	МО Новоизмайловское
1.1.2	МО Кузнецовское

Листинг 17. Выборка предков

В запросе не вычисляется глубина узла.

```
SELECT  *
FROM    areas
WHERE   '1.2.1' /* узел */ LIKE area_path + '%'
ORDER BY area_path
```

Результат выборки предков:

area_path	name
1	Санкт-Петербург
1.2	Невский район
1.2.1	МО Рыбацкое

Листинг 18. Вхождение в поддерево

```
WITH
  node(area_path) AS (SELECT '1.2.1'),
  subtree(area_path) AS (SELECT '1.2')
SELECT result = CASE
    WHEN EXISTS(SELECT 1
      FROM node, subtree
      WHERE SUBSTRING(node.area_path, 1, LEN(subtree.area_path))
          = subtree.area_path)
    THEN N'Входит'
    ELSE N'Не входит'
END
```

Результат проверки вхождения узла:
result = Входит

Листинг 19. Подсчет количества всех потомков узла

```
SELECT COUNT(1) - 1 AS qty
FROM  areas
WHERE area_path LIKE '1.1%' /* корень поддерева */
```

Результат подсчета количества потомков узла:
qty = 2

■ ***Таблица 9.*** Сравнительные характеристики рассмотренных способов

Критерии сравнения	Списки смежности	Подмножества	Хранение маршрута обхода	Материализованные пути
Сложность схемы базы данных (количество)				
Таблицы	1	2	1	1
Ссылки	1	2	0	0
Колонки	3	5	4	2
Запросы на извлечение данных (число соединений)				
Выборка поддерева, число соединений	$H-N+1$ (*)	2	2	1
Выборка пути от узла до корня (предков)	N (*)	2	2	1
Вхождение в поддерево	$H-N+1$ (*)	1	2	0 (сравнение значений двух строк)
Подсчет количества всех потомков узла	$H-N+1$ (*)	1	2	1
Определение высоты узла	N (*)	1	2	1
Соответствие порядка следования узлов сортировке по ключу	Нет	Нет	Нет	Да
Запросы на изменение данных				
Прямая вставка новых узлов для листа	Да	Нет	Да, если есть свободный номер, иначе пересчет диапазонов	Да
Прямая вставка новых узлов для внутренней вершины	Нет, изменение ссылки потомков	Нет, перегруппировка подмножеств	Нет, пересчет диапазонов	Нет, пересчет номеров узлов
Прямое перемещение поддерева	Да, изменение ссылки на предка	Нет, перегруппировка подмножеств	Нет, пересчет диапазонов	Нет, пересчет номеров узлов
Прямое удаление поддерева	Нет, рекурсивное (каскадная DRI)	Нет, перегруппировка подмножеств	Да, заданием диапазона	Да, заданием шаблона подстроки
Избыточность хранения	Нет	Да	Да	Да
Оценка избыточности хранения дерева из N вершин	N	Не хуже $((N-1)/2)N$	$2N$	Не хуже $((N-1)/2)N$
Ограничения высоты дерева	Нет	Нет	Нет	Да
Императивная поддержка целостности	Нужна	Нужна	Нужна	Нужна

Примечание: (*) — рекурсивный запрос, H — высота дерева, N — текущий уровень.

Листинг 20. Определение высоты узла

```
SELECT COUNT(1) AS level
FROM  areas
WHERE '1.1.2' /* узел */ LIKE area_path + '%'
```

Результат определения абсолютного уровня: **level** = 3

Критерии сравнения

В качестве критериев сравнения приведенных способов предлагается использовать следующие характеристики:

— сложность схемы базы данных;
— запросы на извлечение данных;
— запросы на изменение данных;
— избыточность хранения данных;
— поддержка целостности данных.

Сложность схемы базы данных определяется как количество достаточных для реализации таблиц, ссылок (связей) между ними и колонок, содержащих данные о структуре графа (матрице смежности). Запросы на извлечение данных характеризуются количеством необходимых соединений. Наиболее сложным вариантом является рекурсивный запрос, в котором число соединений в цикле соответствует глубине иерархии. Например, выборка поддерева с пятью уровнями будет осуществляться в цикле из пяти итераций, результат каждой из которых соединяется с предыдущим. Запросы на изменение данных, такие как вставка и удаление узлов, характеризуются необходимостью дополнительных операций со связанными узлами и обновлением избыточных данных. Поддержка целостности данных характеризуется необходимостью дополнительного императивного кода (триггеров) помимо декларативной ссылочной целостности.

Сведя перечисленные характеристики в одну общую таблицу (табл. 9), мы получим сравнительную картину, предназначенную для выбора одного из способов реализации.

Заключение

В статье рассмотрены основные способы организации иерархических структур в реляционных базах данных и их характеристики. Важно отметить, что нет «плохих» или «хороших» способов: проектировщик сможет сделать выбор оптимального решения, исходя из условий конкретной задачи на основании предлагаемого множества критериев.

Несмотря на относительно небольшое количество шаблонов логического уровня по сравнению, например, с объектно-ориентированным проектированием, где их выделено более 40 [10], систематизация таковых в применении к базам данных является необходимым шагом как для индустриализации процесса проектирования, так и для обучения.

Предлагаемую классификацию способов плоского представления иерархических данных предполагается использовать в исследованиях, финансируемых грантом Федерального государственного бюджетного учреждения «Российский фонд фундаментальных исследований» № 13-08-01250.

Литература

1. **Ульман Дж.** Основы систем баз данных. – М.: Финансы и статистика. 1983. – 334 с.
2. **Карпова Т. С.** Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
3. **Бураков В. В.** Управление качеством программных средств: монография. – СПб.: ГУАП, 2009. – 287 с.
4. **Бураков В. В.** Концептуальное моделирование качества программных средств // Авиакосмическое приборостроение. 2008. № 7. С. 54–60.
5. **Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И.** Лекции по теории графов. – М.: Наука, 1990. – 384 с.
6. **Ахо Д., Хопкрофт Д., Ульман Д.** Структуры данных и алгоритмы. – М.: Вильямс, 2000. – 384 с.
7. **Кнут Д.** Искусство программирования. 2-е изд. Т. 3. Сортировка и поиск. – М.: Вильямс, 2007. – 824 с.
8. **Вирт Н.** Алгоритмы + Структуры данных = Программы. – М.: Мир, 1985. – 408 с.
9. **Celko J.** Trees and Hierarchies in SQL for Smarties. – Morgan-Kaufmann, 2012. – 296 p.
10. **Гамма Э.** и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2013. – 368 с.