Погружение в СУБД. Сезон 2017

Древовидные структуры и рекурсивные запросы

Дмитрий Барашев

Computer Science Center

Санкт-Петербург 2017

Сверстано в Папирии

papeeria.com

онлайн редактор для LaTeX и Markdown• совместное редактирование в реальном времени • интеграция с Git репозиториями • графики

подсветка синтаксиса • автодополнение • проверка орфографии • предпросмотр математических формул • галерея шаблонов

Хранение и обработка деревьев

Хранение и обработка деревьев

- Списки смежности и рекурсивные запросы

Хранение и обработка деревьев

- Структурные метки

Хранение и обработка деревьев

- Специальные нумерации вершин

Списки смежности и рекурсивные запросы

Списки смежности

- Известный и простой способ представления графов
- Каждая вершина знает список исходящих дуг или смежных вершин

Списки смежности

- Известный и простой способ представления графов
- Каждая вершина знает список исходящих дуг или смежных вершин
- Список ссылок на смежные вершины в ЯП общего назначения

```
class Vertex {
  private List<Vertex> adjacentVertices;
}
```

Списки смежности в таблицах

```
CREATE TABLE Vertex (
  id INT PRIMARY KEY,
  value TEXT,
  parent_id INT REFERENCES Vertex
)
```

Обход дерева

```
class Vertex {
  String value;
  List<Vertex> children;
}

void dfs(Vertex root) {
  out.println(root.value);
  for (Vertex child : root.children) {
    dfs(child);
  }
}
```

Как это сделать в SQL?

- Как проитерироваться по списку детей вершины?
- Как сделать рекурсию?
- Как это сделать в SQL?

Непосредственные потомки вершины

```
CREATE TABLE Vertex (
  id INT PRIMARY KEY,
  value TEXT,
  parent_id INT REFERENCES Vertex
);

SELECT P.id AS parent_id,
       C.id AS child_id
FROM Vertex P JOIN Vertex C
     ON C.parent_id = P.id;
WHERE P.id = 42;
```

Еще бы рекурсию...

```
-- for child_id IN
SELECT P.id AS parent_id,
       C.id AS child_id
FROM Vertex P JOIN Vertex C
     ON C.parent_id = P.id;
-- do the same as above
-- WHERE P.id = child_id !
```

Рекурсивный запрос

```
-- состоит из базы и рекурсивной части
WITH RECURSIVE Fibonacci AS (
  -- Результат базы (нерекурсивной части)
  SELECT 1 AS ord, 1 AS value, 1 AS next_value

  -- объединяется
  UNION ALL

  -- с рекурсивной частью
  SELECT ord + 1 AS ord,
         next_value AS value,
         value + next_value AS next_value
  -- в которой можно использовать результат
  -- предыдущей итерации
  FROM Fibonacci
  WHERE ord < 10
)
-- Итерации заканчиваются если очередная возвращает
-- пустой результат
SELECT * FROM Fibonacci;
```

Собираем поддерево

```
CREATE TABLE Vertex (
  id INT PRIMARY KEY,
  value TEXT,
  parent_id INT REFERENCES Vertex
);

WITH RECURSIVE Bfs AS (
  SELECT V.id AS vertex_id,
         V.parent_id AS parent_id,
         V.value,
         0 AS level
  FROM Vertex V
  WHERE id = 42

  UNION ALL

  SELECT V.id AS vertex_id, V.parent_id AS parent_id,
         V.value, Bfs.level + 1 AS level
  FROM Vertex V JOIN Bfs ON V.parent_id = Bfs.vertex_id
)
SELECT * FROM Bfs;
```

Структурные метки

materialized paths, outline numbers

Структурные метки в таблице

```
CREATE TABLE Vertex (
  id INT PRIMARY KEY,
  value TEXT,
  path INT[]
)
```

Выборка поддерева

```
CREATE OR REPLACE FUNCTION IsPrefix(
    prefix INT[], arr INT[])
RETURNS BOOLEAN AS $$
DECLARE len_prefix INT;
BEGIN
-- префикс меньше либо равен значению
IF prefix > arr THEN
  RETURN FALSE;
END IF;
-- и совпадает с началом значения
len_prefix = array_length(prefix, 1);
RETURN prefix = arr[1:len_prefix];
END
$$ LANGUAGE PLPGSQL;

SELECT * FROM Vertex
WHERE IsPrefix(
    (SELECT path FROM Vertex WHERE id=42),
    path
);
```

Структурные метки

недостатки

- Изменение структуры дерева – дорогая операция
- Нет гарантии целостности: элементы в пути могут вообще не существовать

Расширение ltree

- Тип данных `ltree` с большим множеством операций и индексными структурами
- Входит в «расширенную» поставку PostgreSQL
- Написано программистами из МГУ:

  `http://www.sai.msu.su/~megera/postgres/gist/`

Расширение ltree

- `ltree`: структурная метка, путь

  `'Животные.Хордовые.Млекопитающие'`
- `lquery`: навигационный запрос, похожий на регулярные выражения.

  `'Животные.*'`
- Операции, сравнивающие структурную метку с запросом или создающие объекты `ltree`

  `path ~'Животные.*'`

Пути

- Компоненты состоят из алфавитно-цифровых символов (набор зависит от локали) и подчеркиваний
- Компоненты разделены точками
- Длина одной компоненты не более 256 байт
- Длина всей метки не более 65 килобайт

Навигационные запросы

- Шаблон из слов, разделенных точками
- Простое слово соответствует метке в пути
- * соответствует любой метке или ппоследовательности меток
- Специальные модификаторы ограничивают область действия *
- Логические операции ! и || позволяют делать негативные запросы и объединение запросов

Примеры запросов

```
'*'
```

все пути

```
'Животные.*'
```

все пути, начинающиеся с метки 'Животные'

```
'*.Хищные*.*'
```

все пути, в которых есть метки, начинающиеся с префикса 'Хищные'

```
'Животные.*|Грибы.*'
```

все пути, начинающиеся с метки 'Животные' или 'Грибы'

Основные операции

- Сравнивающие пути

  =, <>, <, >, <=, >=, @>, <@

- Преобразующие пути

  ||, subtree, subpath, text2ltree, ltree2text, lca

- Сравнивающие путь и запрос

  path ~ query

Нумерации вершин

Нумерации вершин

- Каждая вершина дерева нумеруется несколькими числами
- Отношение родитель-потомок выясняется арифметическими операциями с числами
- Числа могут быть целыми (вложенные множества, нумерация Диеца) или рациональными (вложенные множества, дроби Фарея)

Вложенные множества

Nested Sets

- Вершина i нумеруется двумя целыми числами L_i и R_i, назначаемыми при обходе в глубину
- Оба числа берутся из одной и той же монотонно возрастающей последовательности

Вложенные множества

Nested Sets

- Вершина i нумеруется двумя целыми числами L_i и R_i, назначаемыми при обходе в глубину
- Оба числа берутся из одной и той же монотонно возрастающей последовательности
- Число L_i назначается сразу после перехода из родителя в вершину
- Число R_i назначается непосредственно перед возвращением обратно к родителю

Вложенные множества

пример

Вложенные множества

выборка поддерева

```
CREATE TABLE Vertex (
  id INT PRIMARY KEY,
  value TEXT,
  lft INT,
  rgt INT,
  CHECK(rgt > lft)
);
-- выборка поддерева
SELECT Child.* FROM
Vertex Child JOIN Vertex Parent
    ON Child.lft BETWEEN Parent.lft AND Parent.rgt
WHERE Parent.id = 42;
-- или так --
SELECT * FROM
Vertex V
WHERE V.lft >= (SELECT lft FROM Vertex WHERE id=42)
AND V.lft < (SELECT rgt FROM Vertex WHERE id=42);
```

Вложенные множества

выборка ограниченного поддерева

```
-- Количество отрезков-контейнеров для каждой вершины
WITH Level AS (
  SELECT 0 AS id, 0 AS level
  UNION
  SELECT Child.id, COUNT(Parent.id) as level
  FROM Vertex Child JOIN Vertex Parent
      ON Child.lft > Parent.lft AND Child.lft < Parent.rgt
  GROUP BY Child.id
)
-- Ищем потомков вершины 42, у которых родителей на 3 больше,
-- чем у вершины 42
SELECT V.* FROM
Vertex V JOIN Level L ON V.id = L.id
WHERE V.lft >= (SELECT lft FROM Vertex WHERE id=42)
AND   V.lft <  (SELECT rgt FROM Vertex WHERE id=42)
AND L.level <= (
    SELECT level FROM Level WHERE id=42
) + 3;
```

Вложенные множества

добавление узла

- Найти место вставки нового узла j и значения L_j и R_j которые он бы получил
- Увеличить на 2 все существующие $L_i >= L_j, R_i >= R_j$
- Перестроить примерно пол-дерева

Вложенные множества

интервалы с запасом

- Можно назначать номера R_i с шагом $S > 1$
- В каждый узел тогда можно будет добавить $\lfloor \frac{S-1}{2} \rfloor$ потомков без перестройки всего дерева

Другие нумерации

Нумерация Диеца: две разных последовательности для L и R, правила назначения те же

Другие нумерации

Рациональные числа, непрерывные дроби

Тест производительности

нагрузка

- Запрос, выбирающий все поддерево заданной вершины
- Запрос, выбирающие поддерево заданной вершины глубины 3
- Запросы выполняются для каждой вершины в дереве

Тест производительности

не цель теста

Мы не ищем универсального
ответа на вопрос
«как лучше хранить дерево?»

Тест производительности

лучший способ зависит от...

- Железа
- СУБД
- Конфигурации СУБД
- Природы и особенностей данных
- Природы и особенностей нагрузки
- Наличия или отсутствия индексов
- Реализации запросов

Тест производительности

цель теста

Подтвердить или опровергнуть некоторые предположения:

- Рекурсивные запросы делают много соединений, а значит работают медленно
- Вложенные множества тупо сравнивают числа, а значит быстрые
- Реализации структурных меток массивами и Itree существенно не отличаются

Выборка всего поддерева

```
CREATE OR REPLACE FUNCTION RunDeepAdjLists()
RETURNS VOID AS $$
DECLARE i INT;
DECLARE root INT;
BEGIN
  FOR i IN 0..10000 LOOP
    PERFORM GetSubtreeAdjList(i);
  END LOOP;
END
$$ LANGUAGE plpgsql;
```

Выборка неглубокого поддерева

```
CREATE OR REPLACE FUNCTION RunShallowAdjLists()
RETURNS VOID AS $$
DECLARE i INT;
BEGIN
  FOR i IN 0..10000 LOOP
    PERFORM GetShallowSubtreeAdjList(i, 3);
  END LOOP;
END
$$ LANGUAGE plpgsql;
```

Что нужно запомнить

- Есть несколько вариантов хранения дерева
- Разная функциональность, гарантии согласованности, сложность модификации дерева

Что нужно запомнить

при чтении деревьев

- Списки смежности и рекурсивные запросы работают удивительно хорошо

Что нужно запомнить

при чтении деревьев

- Структурные метки, использующие массивы, работают не очень хорошо

Что нужно запомнить

при чтении деревьев

- ltree работает вполне достойно

Что нужно запомнить

при чтении деревьев

- Вложенные множества работают ожидаемо плохо когда нужна глубина вершины

Your mileage may vary