Иерархические данные в MySQL

Отдел R&D

Маркетинговая группа Текарт

18 декабря 2009 г.

О чем речь

О чем речь

Иерархические данные в таблицах

- В каком виде хранить отношения между узлами дерева?
- Как **эффективно** выбирать данные с учетом отношений:
  - всех потомков узла;
  - путь к узлу;
  - соседей узла;
  - ...
- Как **эффективно** вставлять, обновлять, удалять данные с учетом отношений?
- Как совместить всё это с **MySQL**?

Известные решения

Adjacency **L**ist

Materialized **P**ath

Nested **S**ets

Nested **I**ntervals

О чем речь

Идея

```
1    CREATE TABLE (
2      id INT AUTO_INCREMENT PRIMARY KEY,
3      parent_id INT NOT NULL
4    )
```

В каждом узле хранится идентификатор предка.

Результат

- Хранение не вызывает проблем;
- Выборка приводит к рекурсивным запросам – медленно;
- Вставка и реализация на MySQL не вызывает проблем;
- Удобный способ задания отношений, но неэффективный;
- В некоторых СУБД существует поддержка – connect by в Oracle.

О чем речь

Идея

- Храним весь путь к корню дерева;
- `this.path = parent.path + this.id`;
- Выборка с помощью `LIKE`;
- Существуют модификации, когда хранится не идентификаторы, а номера соседей или граней.

Результат

- При реализации с помощью триггеров MySQL приходиться дополнительные поля выносить в отдельную таблицу;
- Выборка с помощью `LIKE`:
  - вся надежда на индекс;
  - ограничение на длину индекса – 255 символов. Обходится разбиением на несколько полей;
  - при большом количестве строк работает медленно.
- Быстрая выборка при приемлимом объеме данных.

Примеры выборок

Выборка потомков:

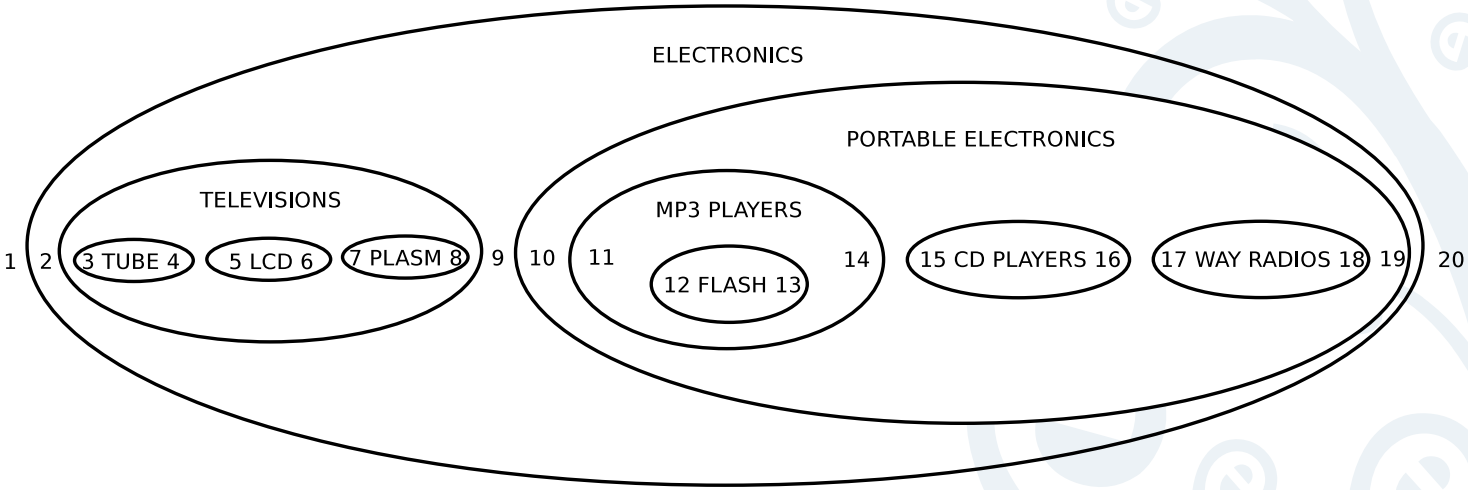
```
1  SET @search_path = '1.2.3';
2  SELECT c.*,m.* FROM categories as c, mp as m
3  WHERE m.entity_id = c.id and
4  m.path like concat(@search_path, '.%');
```

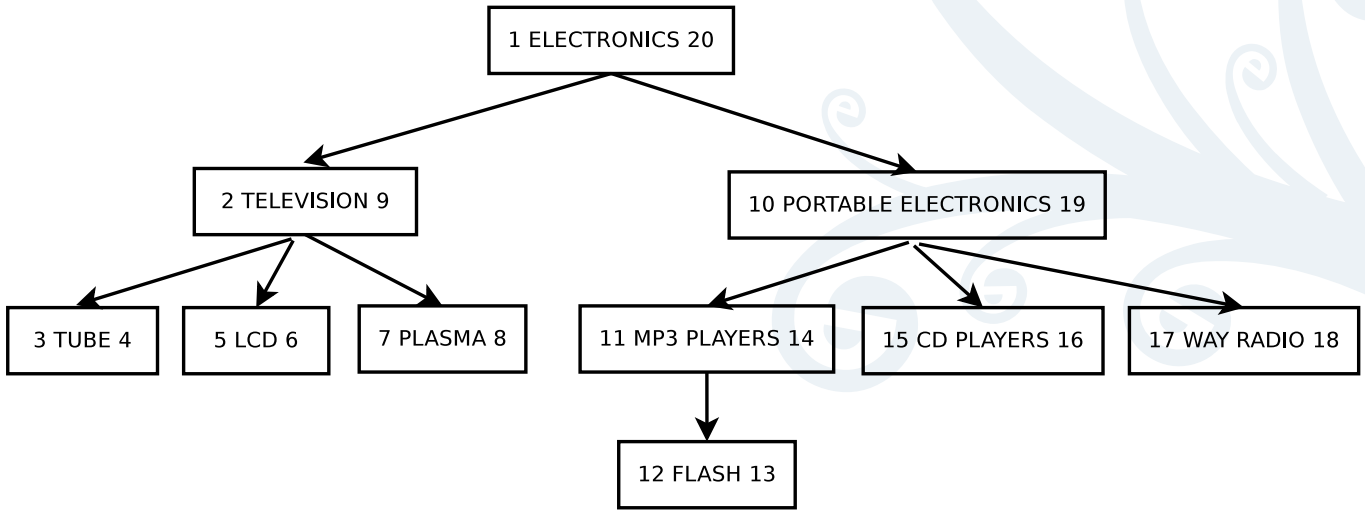
Выборка потомков с учетом разбиения на два поля:

```
1  DROP PROCEDURE IF EXISTS descendants;
2  CREATE PROCEDURE descendants (id BIGINT)
3  BEGIN
4    DECLARE lpath VARCHAR(255);
5    DECLARE lpath1 VARCHAR(255);
6
7    SELECT path, path1 FROM mp
8    WHERE entity_id=id INTO lpath, lpath1;
9
10   IF length(lpath1) > 0 THEN
11     SELECT c.*, m.path, m.path1, m.depth
12     FROM categories as c, mp as m
13     WHERE
14       m.entity_id = c.id and
15       m.path1 like concat(lpath1, '.%');
16   ELSE
17     SELECT  c.*, m.path, m.path1, m.depth
18     FROM categories as c, mp as m
19     WHERE
20       m.entity_id = c.id and
21       m.path like concat(lpath, '.%');
22  END IF;
23 END
```

О чем речь

Идея





Университет Текарт

В результате

- Быстрая выборка
- Хранение и реализация на MySQL не вызывает проблем
- Очень медленная вставка – необходимо обновлять часть таблицы

Примеры выборок

Выборка пути:

```
1  SELECT parent.name
2  FROM nested_category AS node,
3   nested_category AS parent
4  WHERE node.lft BETWEEN parent.lft AND parent.rgt
5  AND node.name = 'FLASH'
6  ORDER BY parent.lft;
```

Выборка потомков:

```
1  SELECT node.name
2  FROM nested_category AS node,
3   nested_category AS parent
4  WHERE node.lft BETWEEN parent.lft AND parent.rgt
5  AND parent.name = 'ELECTRONICS'
6  ORDER BY node.lft;
```

О чем речь

Идея

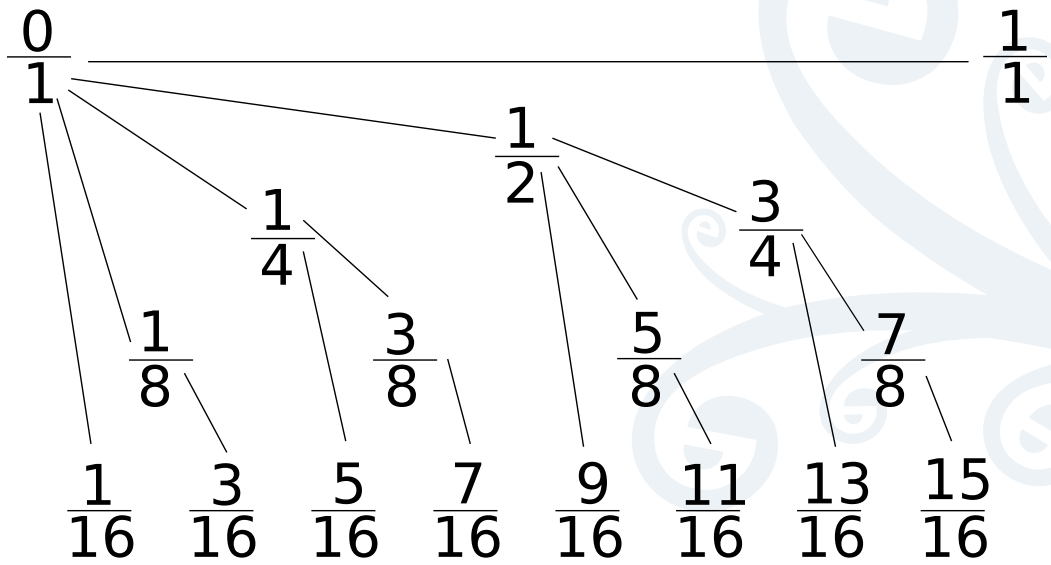
- Дальнейшее развитие Nested Sets:
- Используем рациональные числа вместо целых:
  - множество рациональных чисел бесконечно – всегда можно вставить третий элемент между двумя другими;
  - не надо обновлять таблицу при вставке.
- Значительную часть работы можно выполнить алгоритмически без использования БД (список родителей, расчет глубины и т.д.).

Как рассчитать промежуточный элемент?

- Двоичное кодирование (binary encoding);
- Ряды Фарея (farey fractions).

Двоичное кодирование

- Вложенные элементы рассчитываются с помощью деления отрезка пополам
- Применимо только для небольшого объема данных, т.к. знаменатель растет со скорость 2^n



Университет Текарт

Ряды Фарея I

Для расчета используются ряды Фарея:

$$F_1 = \left\{ \frac{0}{1}, \frac{1}{1} \right\};$$

$$F_2 = \left\{ \frac{0}{1}, \frac{1}{2}, \frac{1}{1} \right\};$$

$$F_3 = \left\{ \frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1} \right\};$$

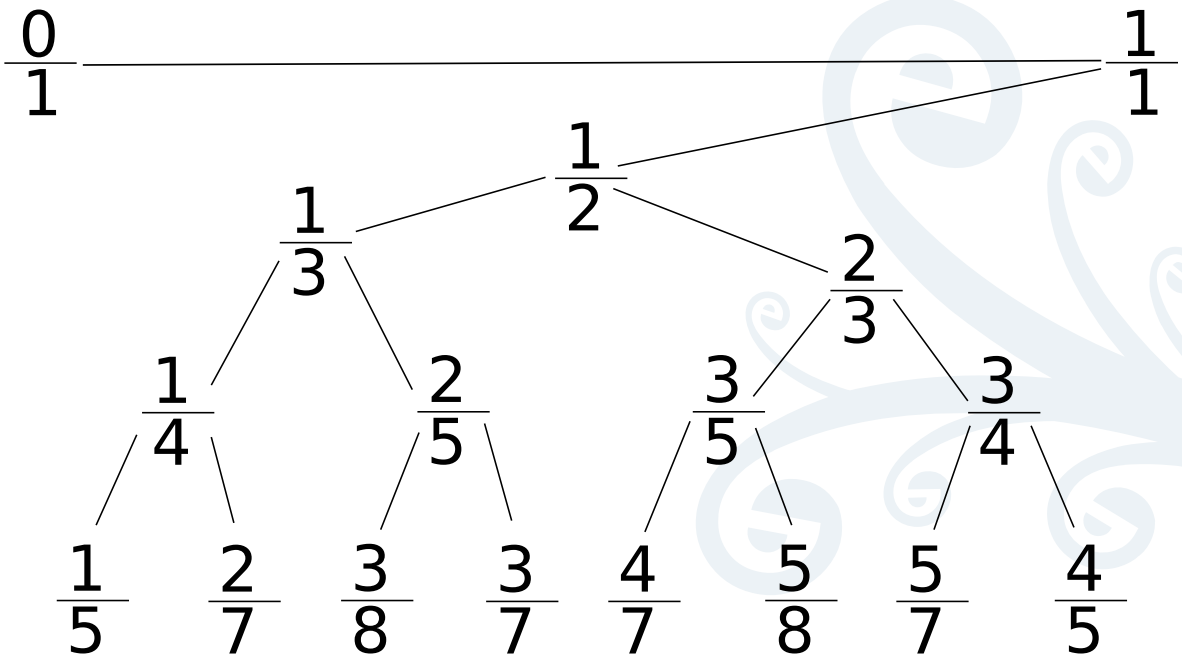
$$F_4 = \left\{ \frac{0}{1}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1} \right\};$$

Ряды Фарея II

- Элементы рассчитываются с помощью медианты:

$$a/b, c/d : \quad \frac{a+c}{b+d}$$

- Применим для гораздо больших объемов данных.



Университет Текарт

Тонкости реализации I

- Дополнительные поля хранятся в основной таблице

Структура таблицы:

```
1  CREATE TABLE categories (
2    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
3    name VARCHAR(20) NOT NULL,
4    parent_id BIGINT UNSIGNED NOT NULL,
5
6    lft_num BIGINT UNSIGNED NOT NULL,
7    lft_den BIGINT UNSIGNED NOT NULL,
8    rgt_num BIGINT UNSIGNED NOT NULL,
9    rgt_den BIGINT UNSIGNED NOT NULL,
10   depth BIGINT UNSIGNED NOT NULL,
11   lft DOUBLE UNSIGNED ZEROFILL NOT NULL,
12   rgt DOUBLE UNSIGNED ZEROFILL NOT NULL,
13
14   UNIQUE INDEX lft_uniq (lft_num, lft_den),
15   INDEX lft (lft),
16   INDEX rgt (rgt),
17   INDEX parent (parent_id)
18 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Тонкости реализации II

- `lft_num, lft_den, rgt_num, rgt_den` числители и знаменатели концов отрезков;
- `lft, rgt, depth` – вспомогательные поля для дополнительной фильтрации при выборке;
- Уникальным индексом является только пара `(lft_num, lft_den)`;
- Индекс `depth` пользы не приносит, т.к. обладает низкой селективностью;
- Поля `rgt,lft` и индексы по ним не являются достаточными – `DOUBLE` поля не хватает, возникают интервалы с одинаковыми значениями `lft` или интервалы с `lft = rgt`;
- Сравнение по `lft,rgt` полям – лишь дополнительная фильтрация, точный результат дает сравнение дробей;

Тонкости реализации III

- Для сравнения дробей вида $\frac{lft\_num}{lft\_den} : \frac{rgt\_num}{rgt\_den}$ приходиться использовать вычисляемую функцию;

Функция сравнения дробей:

```
1  DROP FUNCTION IF EXISTS compare;
2  CREATE FUNCTION compare (
3    num2 BIGINT, den2 BIGINT,
4    num1 BIGINT, den1 BIGINT) RETURNS BIGINT
5  BEGIN
6    RETURN num1*den2 - den1*num2;
7  END
```

Проблемы интеграции с PHP

- Магия при передаче *float* значений: драйвер MySQL поступает очень умно: переводит значение в строку, а потом конвертирует обратно – теряется точно и весь смысл *float*
- При реализации алгоритмических расчетов требуется выбрать строки по значениям (lft_num, lft_den)

```
1  SELECT * FROM categories  WHERE (lft_num, lft_den) IN
2  ((2035, 114476),(1684, 94731),(1333, 74986))
```

Но не подхватывается индекс, выход – покрывающие индексы

```
1  SELECT * FROM categories join
2    (SELECT categories.id id from categories
3      WHERE (lft_num, lft_den) IN
4      ((2035, 114476),(1684, 94731),(1333, 74986))
5    ) as p ON p.id = categories.id;
```

Примеры выборок I

Процедура для выборки потомков:

```
1  DROP PROCEDURE IF EXISTS descendants;
2  CREATE PROCEDURE descendants (id BIGINT UNSIGNED)
3  BEGIN
4   SELECT descen.* FROM
5     categories as node,
6     categories as descen use index (lft)
7     WHERE
8       node.id = id and
9       descen.depth > node.depth and
10      descen.lft between node.lft and node.rgt and
11      compare(
12        node.lft_num, node.lft_den,
13        descen.lft_num, descen.lft_den) > 0 and
14      compare (
15        descen.lft_num, descen.lft_den,
16        node.rgt_num, node.rgt_den) > 0;
17 END
```

Примеры выборок II

Процедура для выборки пути:

```
1  DROP PROCEDURE IF EXISTS path;
2  CREATE PROCEDURE path (id BIGINT UNSIGNED)
3  BEGIN
4    SELECT parent.*
5      from
6        categories as node,
7        categories as parent
8        where
9          node.id=id and
10         node.lft between parent.lft and parent.rgt and
11         compare(
12           node.lft_num, node.lft_den,
13           parent.lft_num, parent.lft_den) <= 0
14         and compare(
15           parent.rgt_num, parent.rgt_den,
16           node.lft_num, node.lft_den) < 0;
17 END
```

О чем речь

Практика: Materialized Path

- Простота алгоритма
- Быстрая вставка
- На тестовых данных в 30000 строк – отличные результаты по скорости выборки потомков
- На тестовых данных в 1000000 строк – `LIKE` тормозит
- Возможность алгоритмически получать путь

Практика: Nested Intervals

- Алгоритм сложнее – тяжелее в поддержке;
- Более медленная вставка, но в пределах 0.2 сек. для 1M записей;
- Приемлемая скорость выборки как при 30000, так и при 1M записей – 0.2 сек.;
- Возможность алгоритмических вычислений.

Общие замечания

- При наличии `LIMIT` в запросе выборка происходит быстро для обоих алгоритмов.
- Скорость того или иного алгоритма сильно зависит от структуры данных;
- В 80% случаях вполне подходит **Materialized Path**.

Что читать

- Joe Celko's Trees and Hierarchies in SQL for Smarties
- Trees in SQL: Nested Sets and Materialized Path by Vadim Tropashko
- Nested Intervals with Farey Fractions by Vadim Tropashko
- Managing Hierarchical Data in MySQL By Mike Hillyer
- Nested Intervals Tree Encoding in SQL by Vadim Tropashko
- Some answers to some common questions about SQL trees and hierarchies by Joe Celko