PostgresPro

Инты, массивы, внешние ключи, наследование

Этюды о нестандартных фичах PostgreSQL

8 июля 2021

PGDay.RU

www.postgrespro.ru

Иван Панченко

Postgres Professional

Массивы

PostgresPro

Объявление

```
CREATE TABLE x (
    a int4,
    b int4[]
);
```

Вставка

```
INSERT INTO x VALUES (
    1,
    ARRAY[2,3,4]
);
```

```
INSERT INTO x VALUES (
    1,
    '{2,3,4}'
);
```

Изменение

```
UPDATE x SET b[1] = 7; -- можно поэлементно или целиком
```

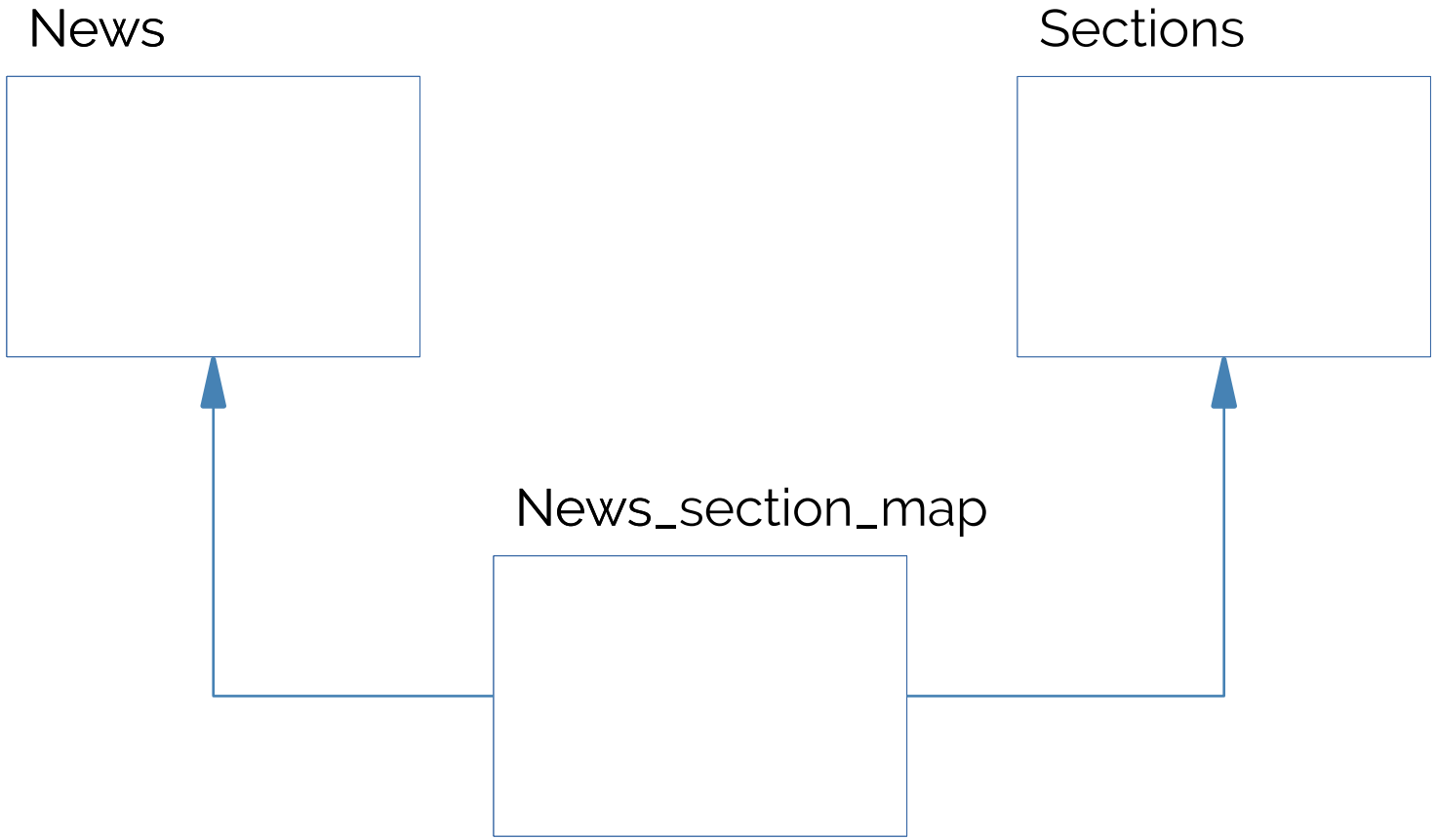
Извлечение

```
SELECT b[1], b FROM x; -- можно поэлементно или целиком
```

С этим мы пришли в XXI век..

PostgresPro

Связь «Многие ко многим». Рамблер-Медиа, 2000г



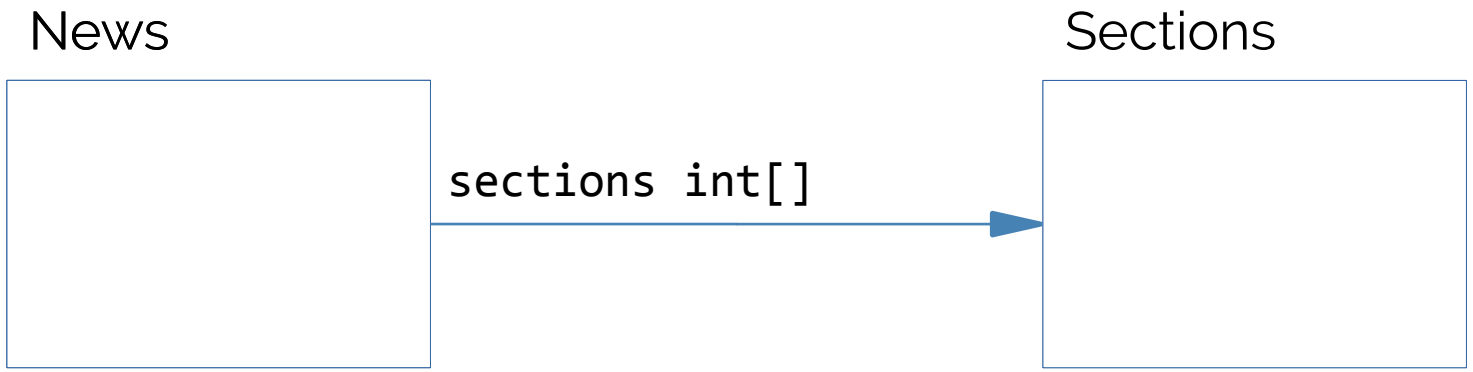
Pentium II/III 400-800MHz
Pentium III 800MHz
512M-1G RAM
33Gb HDD

Запрос на выборку новостей по рубрике содержал JOIN, и тормозил.
Запрос по пересечению рубрик — ещё хуже.

Альтернативное решение

PostgresPro

Связь через массив



От JOIN'а избавились. Но...

Выборка: `SELECT * FROM news WHERE sections && ARRAY[section_id]`

Что-то новое...

Поиск по массивам

PostgresPro

Федор Сигаев, Олег Бартунов 2000 г.

`contrib/intarray` — набор полезных функций для работы с массивами.
В т.ч. операторы &&, <@, @>, @@ и др.

```
&&  : (int[], int[] )→bool -- есть ли пересечение массивов
<@  : (int[], int[] )→bool -- входит ли первый во второй ⊆
@>  : (int[], int[] )→bool -- входит ли второй в первый  ⊇
@@  : (int[], query_int )→bool -- удовлетворяет ли массив запросу
                                  типа  1&2&3&(4|5)
```

А также индекс типа GiST, позже - GIN

```
CREATE INDEX ON x USING gist (a gist__intbig_ops)

CREATE INDEX ON x USING gin  (a)
```

Поиск по массивам

PostgresPro

Федор Сигаев, Олег Бартунов 2000 г.

`contrib/intarray` — набор полезных функций для работы с массивами.
В т.ч. операторы &&, <@, @>, @@ и др.

```
&&  : (int[], int[] )→bool -- есть ли пересечение массивов
<@  : (int[], int[] )→bool -- входит ли первый во второй ⊆
@>  : (int[], int[] )→bool -- входит ли второй в первый  ⊇
@@  : (int[], query_int )→bool -- удовлетворяет ли массив запросу
                                  типа  1&2&3&(4|5)
```

А также индекс типа GiST, позже - GIN

```
CREATE INDEX ON x USING gist (a gist__intbig_ops)

CREATE INDEX ON x USING gin  (a)
```

Не int ?

PostgresPro

```
&&  : (anyarray, anyarray )→bool -- есть ли пересечение массивов
<@  : (anyarray, anyarray )→bool -- входит ли первый во второй ⊆
@>  : (anyarray, anyarray )→bool -- входит ли второй в первый  ⊇
@@  : (int[], query_int )→bool -- удовлетворяет ли массив запросу
                                  типа  1&2&3&(4|5)
```

Индекс типа GIN

```
CREATE INDEX ON x USING gin (a)
```

Для `int[], text[]`

Для `int8[] , uuid[]` и т.п. - нет

Foreign Keys on Arrays

PostgresPro

Не поддерживается:

```
CREATE TABLE news (
    ...
    sections  int[] REFERENCES section(id)

);
```

Необходимо контролировать руками (триггерами или приложением).

```
CREATE TRIGGER news_upd BEFORE INSERT OR UPDATE ON news
       FOR EACH ROW
       EXECUTE PROCEDURE check_news_outgoing_refs();

CREATE TRIGGER section_del BEFORE DELETE ON section
       FOR EACH ROW
       EXECUTE PROCEDURE check_section_incoming_refs();
```

int4 или int8 ?

PostgresPro

Что выбрать в качестве PK ?

```
#define idtype int4
#define idtype int8
……
```

```
CREATE DOMAIN idtype int4;
```

Сразу перестанет работать всё, что мы сделали ранее с массивами.
Придётся определять для idtype всё, что потребуется.

Например, заставим работать оператор && для idtype[];

Оператор && на доменном типе

PostgresPro

```
CREATE FUNCTION _int_overlap(a idtype[], b idtype[]) RETURNS bool LANGUAGE SQL
    STRICT IMMUTABLE
    SECURITY DEFINER
--  Это трюк. Чтобы функция не инлайнилась, иначе заинлайнится основанный на ней
--  оператор и постгрес не сообразит, что индекс нужно использовать.
    PARALLEL SAFE
    AS $$ SELECT a::_int4 &&  b::_int4 $$;

CREATE OPERATOR && (
    LEFTARG = idtype[],
    RIGHTARG = idtype[],
    PROCEDURE = _int_overlap,
    COMMUTATOR = '&&',
    RESTRICT = _int_overlap_sel,
    JOIN = _int_overlap_joinsel );
```

...продолжение

PostgresPro

```
CREATE OPERATOR CLASS gist__idtypebig_ops FOR TYPE idtype[] USING gist AS
    OPERATOR    3   && ,
    OPERATOR    6   =  (anyarray, anyarray),
    OPERATOR    7   @> (_int4,_int4) ,
    OPERATOR    8   <@ (_int4,_int4),
    OPERATOR    13  @  (_int4,_int4),
    OPERATOR    14  ~  (_int4,_int4),
    OPERATOR    20  @@ (_int4, query_int),
    FUNCTION    1   g_intbig_consistent (internal, _int4, smallint, oid, internal),
    FUNCTION    2   g_intbig_union (internal, internal),
    FUNCTION    3   g_intbig_compress (internal),
    FUNCTION    4   g_intbig_decompress (internal),
    FUNCTION    5   g_intbig_penalty (internal, internal, internal),
    FUNCTION    6   g_intbig_picksplit (internal, internal),
    FUNCTION    7   g_intbig_same (intbig_gkey, intbig_gkey, internal),
    STORAGE     intbig_gkey;
```

Для int8, uuid ?

PostgresPro

В целом то же, но работы больше, т. к. нет contrib/int8array

Developer challenge!

Что ещё можно делать с массивами?

PostgresPro

Агрегация. (осторожно, размер!)

```
SELECT array_agg(some_field) FROM some_table WHERE ….
```

Но! Порядок элементов массива будет случайным

```
SELECT array_agg(some_field) FROM (
    SELECT * FROM some_table WHERE …
        ORDER BY  …
) foo;
```

Или

```
SELECT array_agg (some_field ORDER BY ….) FROM some_table WHERE ...
```

Другая польза от массивов

PostgresPro

Сокращение используемого места и трафика путем объединения последовательных записей в одну.

```
CREATE TABLE  ts (
       time timestamptz,
       value  double
);

CREATE TABLE ts_grouped (
    time timestamptz,
    offsets int[], -- например, в мс
    values double[]
);
```

JSONB наступает на пятки

PostgresPro

Всё то же, что с массивами, можно делать и с JSON(b) — и гораздо больше

Но:

- Производительность
- Объём

Наследование таблиц

PostgresPro

```
CREATE TABLE foo (
 id  idtype PRIMARY KEY,
 …..
);

CREATE TABLE bar1 (
 f1 text
) INHERITS (foo);

CREATE TABLE bar2 (
 f2 text
) INHERITS (foo);
```

Попробуем

```
INSERT INTO bar1 VALUES (NULL, 'q');
-- ошибка (NOT NULL унаследовалось)
```

```
INSERT INTO bar1 VALUES (1,'q');
INSERT INTO bar1 VALUES (1,'q');
INSERT INTO bar1 VALUES (1,'q');
-- не ошибка (индексы и уникальность не унаследовались).
```

Фича отмирает?

Наследование таблиц

PostgresPro

```
CREATE TABLE foo (
 id  idtype PRIMARY KEY,
 …..
);

CREATE TABLE bar1 (
 f1 text
) INHERITS (foo);

CREATE TABLE bar2 (
 f2 text
) INHERITS (foo);
```

Попробуем

```
INSERT INTO bar1 VALUES (NULL, 'q');
-- ошибка (NOT NULL унаследовалось)

INSERT INTO bar1 VALUES (1,'q');
INSERT INTO bar1 VALUES (1,'q');
INSERT INTO bar1 VALUES (1,'q');
-- не ошибка (индексы и уникальность не унаследовались).
```

Надо:

```
CREATE TABLE bar1 (
 id idtype PRIMARY KEY, --warning
 f1 text
) INHERITS (foo);
```

Но и этого мало!

Получение списка всех объектов

PostgresPro

```
SELECT * FROM foo;    -- неправильно

SELECT * FROM         -- правильно
 (
     SELECT 'bar1', id, f1, NULL AS f2
     UNION ALL
     SELECT 'bar2', id,  NULL AS f1, f2
);
```

FK на унаследованную таблицу

PostgresPro

```
CREATE TABLE some (
    foo idtype REFERENCES foo(id)
);

Надо:
CREATE TABLE some (
    foo idtype
);
CREATE TRIGGER …

-- не забываем, что наследование может быть многоуровневым. Поэтому….
```

Как получить всех потомков?

PostgresPro

```
CREATE FUNCTION get_inheritance_tree(schema_ text, tablename_ text)
    RETURNS SETOF inheritance_tree_item LANGUAGE SQL STABLE PARALLEL SAFE AS $$
  WITH RECURSIVE  tree AS (
   SELECT c.oid AS relid, s.nspname, c.relname, ARRAY[c.oid] AS path,
             0 AS depth, 0 AS pos FROM pg_class c
         JOIN pg_namespace s ON c.relnamespace = s.oid
         WHERE relname=tablename_ AND nspname=schema_
   UNION ALL
   SELECT c.oid AS relid, s.nspname, c.relname, array_cat(ARRAY[c.oid],tree.path),
depth + 1 AS depth, inhseqno AS pos
        FROM pg_class c
       JOIN pg_namespace s ON c.relnamespace = s.oid
       JOIN pg_inherits i  ON inhparent = c.oid
       JOIN tree           ON i.inhrelid = tree.relid
  )
  SELECT relid AS id, nspname AS schema, relname AS tablename,
         path[array_length(path,1)-2] AS parent, path, depth, pos
  FROM tree ORDER BY depth, pos;
$$;
```

СПРОСИТЕ POSTGRES PRO!

www.postgrespro.ru

info@postgrespro.ru

И приходите на наши конференции!