PGConf.Russia 2023

PostgresPro

UUID v7 & v8

Зачем и когда

PostgresPro

Естественные/искусственные ключи

- Естественные — можно получить из самих данных, н.,
  - ФИО, дата и место рождения
  - Номер документа
  - Хеш всей строки или ее части
  - UUID v3,5
  - **Требуют неизменности данных**

- Искусственные — никак не связаны с данными
  - Последовательности, UUID v1,4,6,7
  - **Не требуют** неизменности данных

Числовые последовательности

Ѻче́нь традицїо́ннѡ

- Кстати, generated always as identity лучше, чем serial/bigserial
- Достоинства
  - Быстро
  - Небольшого размера
  - **Коррелирует со временем** — при последовательной вставке значения в индексе оказываются рядом (локальность)
  - **Монотонность в случае вставки пачкой** (значения в пачке упорядочены в порядке вставки)

PostgresPro

Числовые последовательности

Недостатки

- Неудобно и/или иногда медленно: в случае одной БД все быстро, в случае нескольких — либо разбивать на диапазоны/начальные значения + шаг, что неудобно и ведет к ошибкам, либо устраивать отдельный сервис с etcd/zookeeper и предварительным получением пачек на инстанс; нужен в той или иной форме единый источник
- Они **НЕУНИКАЛЬНЫ —** независимые последовательности генерируют одинаковые значения, order.id=order_item.id
- Они предсказуемы
- Они всегда искусственные

PostgresPro

Альтернативы последовательностям

- Случайные значения
  - UUID v4
- Хеши
  - Адреса, хеши блоков и транзакций в криптовалютах
  - Хеши от данных
- Недостатки
  - Размер — принципиально неустранимый
  - Нелокальность
  - Неупорядоченность
- Достоинства
  - Обычно непредсказуемы

PostgresPro

Анатомия UUID

```
cfb01158-4bb8-4817-8e8a-8201f761483a

xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx
```

4,M — версия, четыре старших бита

8,N — вариант, 1-3 старших бита

Всего 16 байт=128 бит, из них от 5 до 8 бит расходуются на версию и вариант, в типичном случае остается 122 бита для собственно значения

https://datatracker.ietf.org/doc/html/rfc4122

В Postgres UUID — стандартный тип

Версии UUID

Версия	Способ получения
1	Время + MAC, время специально сохраняется с младших байтов
2	DCE security version(?)
3	Hash, MD5
4	Random
5	Hash, SHA-1

PostgresPro

Вариант

- 0 — 0xxx - «now-obsolete Apollo Network Computing System»
- 1 — 10xx, RFC 4122
- 2 — 110x, «Microsoft Corporation backward compatibility», любопытно, что в Windows интерфейс `iUnknown={00000000-0000-0000-C000-000000000046}`
- 111x — зарезервировано

PostgresPro

Версии UUID

Стандартные UUID вполне подходят для первичных ключей

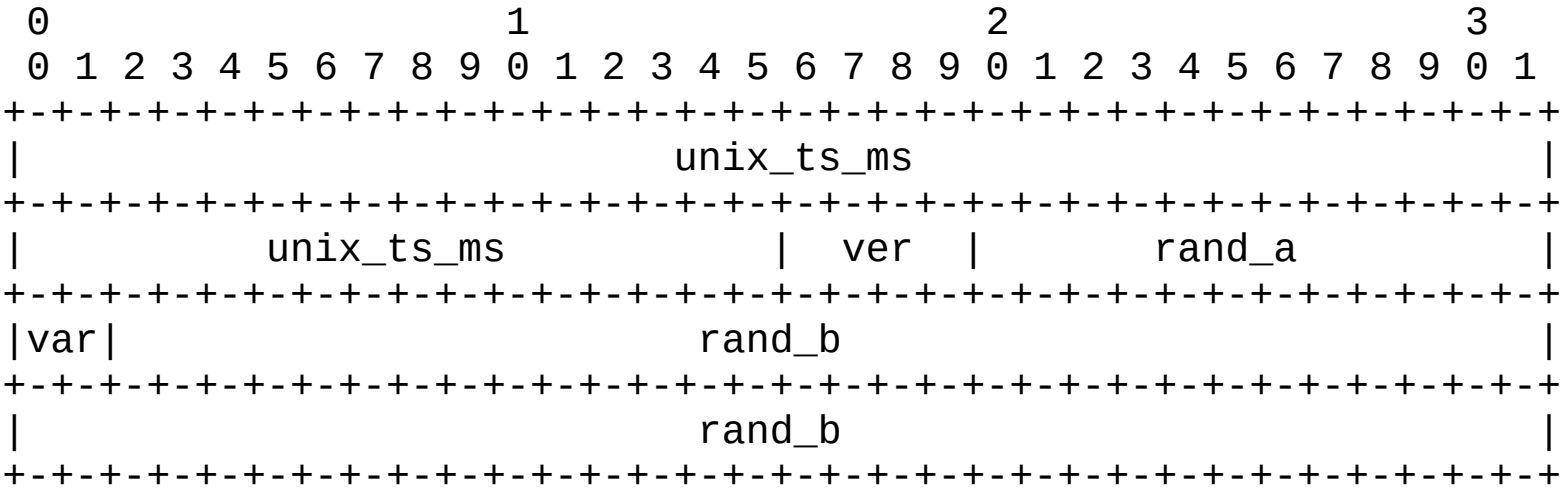
- 4 — искусственные ключи
- 3 и 5 — естественные ключи
- Недостатки:
  - Нелокальность
  - Неупорядоченность

UUID v6, v7 и v8

https://uuid6.github.io/uuid6-ietf-draft/

6	A re-ordering of UUID version 1 so it is sortable as an opaque sequence of bytes.
7	An entirely new time-based UUID bit layout sourced from the widely implemented and well known Unix Epoch timestamp source.
8	RFC-compatible format for experimental or vendor-specific use cases.

V7 — искусственные ключи



- 48 бит — unix ts
- rand_a, rand_b — случайные значения

PostgresPro

v7

Достоинства

- **Локальность** — UUID’ы, полученные примерно в одно время, имеют близкие значения
- **Непредсказуемость** — случайных данных 62 бита, перебрать практически невозможно, как и угадать время генерации с точностью в 48 бит

PostgresPro

Пример реализации

```
CREATE OR REPLACE FUNCTION wb.gen_v7_uuid() RETURNS uuid LANGUAGE plpgsql
 PARALLEL SAFE AS $function$
declare
 tp text = lpad(to_hex(floor(extract(epoch from clock_timestamp())*1000)::int8),
                12,'0')||'7';
 entropy text = md5(gen_random_uuid()::text);
begin
    return (tp ||
            substring(entropy from 1 for 3) || to_hex(8+(random()*3)::int) ||
            substring(entropy from 4 for 15)
           )::uuid;
 end
$function$;
```

PostgresPro

V7 — последовательные значения

«Additionally, care SHOULD be taken to ensure UUIDs generated in batches are also monotonic»

Монотонность

- Значения, создаваемые в группах, должны быть монотонно возрастающими

PostgresPro

V7 — последовательные значения

```
CREATE OR REPLACE FUNCTION wb.gen_v7_uuid_seq() RETURNS uuid LANGUAGE plpgsql AS $function$
declare
 cts timestamptz;
 state text = current_setting('uuidv7.old_tp',true);
 old_tp text = split_part(state, ':',1);
 base int = coalesce(nullif(split_part(state,':',4),'')::int,(random()*16777215/2-1)::int);
 tp text;
 entropy text;
 seq text=base;
 seqn int=split_part(state,':',2);
 ver text = coalesce(split_part(state,':',3),to_hex(8+(random()*3)::int));
 begin
    base = (random()*16777215/2-1)::int;
    cts = clock_timestamp();
    tp = lpad(to_hex(floor(extract(epoch from cts)*1000)::int8),12,'0')||'7';
    if tp is distinct from old_tp then
      old_tp = tp;
      ver = to_hex(8+(random()*3)::int);
      base = (random()*16777215/2-1)::int;
      seqn = base;
    else
      seqn = seqn+(random()*1000)::int;
    end if;
    perform set_config('uuidv7.old_tp',old_tp||':'||seqn||':'||ver||':'||base, false);
    entropy = md5(gen_random_uuid()::text);
    seq = lpad(to_hex(seqn),6,'0');
    return (tp || substring(seq from 1 for 3) || ver || substring(seq from 4 for 3)
               || substring(entropy from 1 for 12))::uuid;
 end
$function$;
```

PostgresPro

V7 - генерация последовательностей

```
if@work=# select n, wb.gen_v7_uuid_seq()
from generate_series(1,10) as gs(n) order by 2;

 n  |            gen_v7_uuid_seq
----+--------------------------------------
  1 | 018736d9-9f73-7234-87bf-1d56dbec07a4
  2 | 018736d9-9f74-75bb-a872-56e07b627b35
  3 | 018736d9-9f74-75bb-a88e-631a156f91c6
  4 | 018736d9-9f74-75bb-aa1f-e1eac0ca3424
  5 | 018736d9-9f74-75bb-acbc-b32b92339949
  6 | 018736d9-9f74-75bb-ad0d-442a4af061d7
  7 | 018736d9-9f74-75bc-a0de-44ee373f4d07
  8 | 018736d9-9f74-75bc-a2d8-9809c12b0434
  9 | 018736d9-9f74-75bc-a491-220f9ac46242
 10 | 018736d9-9f74-75bc-a7f2-0e7cfa9b4dd5

(10 rows)

Time: 1.661 ms
```

V7 — последовательные значения

```
if@work=# explain analyze select n, wb.gen_v7_uuid_seq() from generate_series(1,62500) as gs(n);

                                        QUERY PLAN
-----------------------------------------------------------------------------------------------
 Function Scan on generate_series gs  (cost=0.00..16250.00 rows=62500 width=20) (actual
time=15.475..962.407 rows=62500 loops=1)
 Planning Time: 0.060 ms
 Execution Time: 967.891 ms

(3 rows)
```

PostgresPro

Пример реализации для фиксированного timestamp - секционирование

```
CREATE OR REPLACE FUNCTION wb.gen_v7_uuid(ts timestamp without time zone)
RETURNS uuid LANGUAGE plpgsql PARALLEL SAFE AS $function$
declare
 tp text = lpad(to_hex(floor(extract(epoch from ts)*1000)::int8),12,'0')||'7';
 begin
    return (tp || '0008000000000000000'):::uuid;
 end
$function$;
```

PostgresPro

V7

Позволяет создавать ключи

- Уникальные
- Монотонные
- Локальные
- Удобные для партиционирования
  - Отдельно определен особый тип — Max UUID `FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF`
  - Тип NIL UUID был всегда
    - `00000000-0000-0000-0000-000000000000`
- Для UUID в Postgres есть специальный встроенный тип

PostgresPro

V8 — естественные ключи

«RFC-compatible format for experimental … use cases»
Можно использовать для генерации естественных ключей:

```
CREATE OR REPLACE FUNCTION wb.gen_v8_uuid(ts timestamp without time zone, v text)
 RETURNS uuid
 LANGUAGE plpgsql IMMUTABLE PARALLEL SAFE AS $function$
declare
 hash text = encode(sha256(v::bytea),'hex');
 tp text = lpad(to_hex((extract(epoch from ts)::int8*1000)),12,'0');
 begin
    return (tp ||'8'|| substring(hash from 2 for 3)||'8'
              ||substring(hash from 6 for 15)
            )::uuid;
 end
$function$;
```

PostgresPro

Дополнение

```
if@work=# select gen_random_uuid();
           gen_random_uuid
--------------------------------------
 f8aeb511-9213-400d-a2ef-a487e1ce7c6c

if@work=# select md5('uuid')::uuid;
                  md5
--------------------------------------
 ef7c876f-00f3-acdd-d00f-a671f52d0b1f
```

- uuid-ossp

PostgresPro

Выводы

UUID v7 позволяет **распределенно** создавать **уникальные**, **монотонные**, **локальные** первичные ключи, хорошо подходящие для использования в БД

Представленная реализация UUID v8 позволяет создавать уникальные ключи, основываясь на времени и некоторой строке, что может быть использовано как генератор естественных ключей

Postgres уже имеет встроенный тип, предназначенный для хранения именно этих значений

PGConf.Russia 2023

PostgresPro

Спасибо!