

Миграция Int -> Bigint

Александр Никитин, DBA



PGMechanix

Наша команда PostgreSQL DBA

Работаем с Big Data

Поддерживаем тысячи БД объемом
свыше **30 ТБ** на хост.

Поддерживаем mission-critical БД

Обеспечиваем доступность команды
24/7/365.

SLA на все обращения

Гарантированные сроки реакции на
обращения. SLA на аварии от **30 мин.**

Делимся знаниями

За плечами у каждого
не менее **10 лет** опыта
работы с PostgreSQL.

Занимаемся разработкой

Патчи наших DBA внесены в
mainstream код PostgreSQL.

Проводим диагностику

Нашими DBA подано
около **300 bug reports**.

Наши клиенты

платежные системы



разработка

банки



ритейл

телеком



геймдев

логистика



образование



-  Что делать, если уже всё сломалось?
-  Как обнаружить, что скоро всё сломается?
-  Как предотвратить ситуацию переполнения типа?

Уже всё сломалось!

Мы не отследили момент
переполнения.

01



Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807

девять **квинтиллионов** двести двадцать три **квадриллиона** триста семьдесят два **триллиона** тридцать шесть **миллиардов** восемьсот пятьдесят четыре **миллиона** семьсот семьдесят пять **тысяч** восемьсот семь

```
CREATE TABLE pg_conf (  
    id    integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY  
(START WITH 2147483646 INCREMENT BY 1),  
    name  varchar(40) NOT NULL CHECK (name <> '')  
);
```

```
max(int) = 2147483647
```

```
insert into pg_conf (name) values ('a');
```

```
insert into pg_conf (name) values ('b');
```

```
insert into pg_conf (name) values ('c');
```

```
ERROR:  nextval: reached maximum value of sequence  
"pg_conf_id_seq" (2147483647)
```

```
postgres=# select * from pg_conf;
```

id	name
2147483646	a
2147483647	b

```
alter table pg_conf alter column id type bigint;
```

- 1) Эксклюзивная блокировка таблицы на всё время перестроения
- 2) Избавление от блоата
- 3) Перестроение всех индексов.



ВСЁ!

Предел где-то рядом

Разберёмся как отслеживать
подобные ситуации

02



Истина где-то рядом



Как бы определить то, что в таблицах скоро наступит переполнение?

Как бы определить то, что в таблицах скоро наступит переполнение?

1) Наверное, хорошо было бы вести список таблиц с полями типа `int`.

Как бы определить то, что в таблицах скоро наступит переполнение?

- 1) Наверное, хорошо было бы вести список таблиц с полями типа `int`.
- 2) По крону опрашивать все поля на значение максимума.

Как бы определить то, что в таблицах скоро наступит переполнение?

- 1) Наверное, хорошо было бы вести список таблиц с полями типа `int`.
- 2) По крону опрашивать все поля на значение максимума.
- 3) Отслеживать динамику роста и строить прогнозы.

Конечно, никто этим заниматься не будет 😊

Конечно, никто этим заниматься не будет 😊

Потому что есть статистика.

```
select * from pg_stats where tablename = 'BiG' and attname = 'Id1' \gx
```

```
schemaname      | Pub
tablename       | BiG
attname         | Id1
inherited       | f
null_frac       | 0
avg_width       | 4
n_distinct      | -1
most_common_vals|
most_common_freqs|
histogram_bounds|
(48,19664,41824,68398,79788,99450,119266,140949,162246,182856,203103,224314,241847,260581,279520,300747,320097,340705,361023,380822
,399876,418726,439367,459709,479830,500020,520472,541667,560350,579318,599900,619272,639818,662281,682896,703497,725302,744544,7676
06,787522,808224,828515,847867,866637,886496,906325,927126,946760,965922,987345,1005678,1027246,1046229,1068477,1087540,1105918,112
5533,1144762,1165713,1184532,1204372,1224411,1244806,1264502,1286363,1305370,1325105,1345259,1363680,1383702,1403000,1423359,144323
9,1463502,1486144,1504681,1522827,1544265,1564710,1585230,1604612,1625444,1643785,1664036,1683492,1702049,1722750,1740977,1761986,1
779599,1799320,1818998,1837386,1856913,1875871,1895298,1913951,1936424,1956173,1979090,1999968)
correlation     | 1
most_common_elems|
most_common_elem_freqs|
elem_count_histogram|
```

```
select * from pg_stats where tablename = 'BiG' and attname = 'Id1' \gx
```

```
schemaname      | Pub
tablename       | BiG
attname         | Id1
inherited       | f
null_frac       | 0
avg_width       | 4
n_distinct      | -1
most_common_vals|
most_common_freqs|
histogram_bounds|
(48,19664,41824,68398,79788,99450,119266,140949,162246,182856,203103,224314,241847,268581,279520,300747,320097,340705,361023,380822
,399876,418726,439367,459709,479830,500020,520472,541667,560350,579318,599900,619272,639818,662281,682896,703497,725302,744544,7676
06,787522,808224,828515,847867,866637,886496,906325,927126,946760,965922,987345,1005678,1027246,1046229,1068477,1087540,1105918,112
5533,1144762,1165713,1184532,1204372,1224411,1244806,1264502,1286363,1305370,1325105,1345259,1363680,1383702,1403000,1423359,144323
9,1463502,1486144,1504681,1522827,1544265,1564710,1585230,1604612,1625444,1643785,1664036,1683492,1702049,1722750,1740977,1761986,1
779599,1799320,1818998,1837386,1856913,1875871,1895298,1913951,1936424,1956173,1979090,1999968)
correlation     | 1
most_common_elems|
most_common_elem_freqs|
elem_count_histogram|
```

```
select schemaname, tablename, attname, histogram_bounds[array_length(histogram_bounds, 1)] as last_value_in_stat from
(SELECT n.nspname AS schemaname,
       c.relname AS tablename,
       a.attname,
       (CASE
        WHEN s.stakind1 = 2 THEN s.stavalues1
        WHEN s.stakind2 = 2 THEN s.stavalues2
        WHEN s.stakind3 = 2 THEN s.stavalues3
        WHEN s.stakind4 = 2 THEN s.stavalues4
        WHEN s.stakind5 = 2 THEN s.stavalues5
        ELSE NULL::anyarray
       END)::text::bigint[] AS histogram_bounds,
       s.stadistinct,
       s.stanullfrac
FROM pg_statistic s
JOIN pg_class c ON c.oid = s.starelid
JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum
LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
where s.stadistinct = -1 and s.stanullfrac = 0 and s.stawidth = 4 and a.atttypid = (select oid from pg_type where
typname = 'int4') offset 0) t
where histogram_bounds[array_length(histogram_bounds, 1)] > 2^30;
```

Мы узнали, что скоро будет переполнение

Что делать, какому алгоритму
следовать?

03



Борьба с переполнением

- добавляем новое поле с нужным типом, nullable без default значения

Борьба с переполнением

- добавляем новое поле с нужным типом, nullable без default значения
- создаем триггер, который будет заполнять новое поле при вставках и обновлениях

Борьба с переполнением

- добавляем новое поле с нужным типом, nullable без default значения
- создаем триггер, который будет заполнять новое поле при вставках и обновлениях
- обновляем строки в таблице пачками с паузами, пропуская уже заполненные значения. Размер пачки подбираем экспериментально

Борьба с переполнением

- добавляем новое поле с нужным типом, nullable без default значения
- создаем триггер, который будет заполнять новое поле при вставках и обновлениях
- обновляем строки в таблице пачками с паузами, пропуская уже заполненные значения. Размер пачки подбираем экспериментально
- проверяем, что значения в старом и новом полях везде совпадают.

Борьба с переполнением

- добавляем новое поле с нужным типом, nullable без default значения
- создаем триггер, который будет заполнять новое поле при вставках и обновлениях
- обновляем строки в таблице пачками с паузами, пропуская уже заполненные значения. Размер пачки подбираем экспериментально
- проверяем, что значения в старом и новом полях везде совпадают.
- проставляем not null, если необходимо и default значение

Борьба с переполнением

- добавляем новое поле с нужным типом, nullable без default значения
- создаем триггер, который будет заполнять новое поле при вставках и обновлениях
- обновляем строки в таблице пачками с паузами, пропуская уже заполненные значения. Размер пачки подбираем экспериментально
- проверяем, что значения в старом и новом полях везде совпадают.
- проставляем `not null`, если необходимо и `default` значение
- создаем индексы, аналогичные индексам со старым полем

Борьба с переполнением

убеждаемся, что нет долгих транзакций и в транзакции:

- переименовываем старое поле в `old_column`, новое поле в `column_name`

Борьба с переполнением

убеждаемся, что нет долгих транзакций и в транзакции:

- переименовываем старое поле в `old_column`, новое поле в `column_name`
- удаляем триггер

Борьба с переполнением

убеждаемся, что нет долгих транзакций и в транзакции:

- переименовываем старое поле в `old_column`, новое поле в `column_name`
- удаляем триггер
- если есть `fk` на это поле - удаляем `fk`

Борьба с переполнением

убеждаемся, что нет долгих транзакций и в транзакции:

- переименовываем старое поле в `old_column`, новое поле в `column_name`
- удаляем триггер
- если есть `fk` на это поле - удаляем `fk`
- если заменяемое поле `pk` - удаляем `pk` со старого поля и создаем на новом из существующего уникального индекса

Борьба с переполнением

убеждаемся, что нет долгих транзакций и в транзакции:

- переименовываем старое поле в `old_column`, новое поле в `column_name`
- удаляем триггер
- если есть `fk` на это поле - удаляем `fk`
- если заменяемое поле `pk` - удаляем `pk` со старого поля и создаем на новом из существующего уникального индекса
- если были `fk` - создаем их заново с `not valid`

Борьба с переполнением

убеждаемся, что нет долгих транзакций и в транзакции:

- переименовываем старое поле в `old_column`, новое поле в `column_name`
- удаляем триггер
- если есть `fk` на это поле - удаляем `fk`
- если заменяемое поле `pk` - удаляем `pk` со старого поля и создаем на новом из существующего уникального индекса
- если были `fk` - создаем их заново с `not valid`
- убираем со старого поля `not null`

Борьба с переполнением

- проверяем, что все работает
- если были `fk` - делаем им `validate constraint`
- если был сиквенс на поле, то меняем `owned by` сиквенса на новое поле
- удаляем старое поле `columnname_old`
- при необходимости сжимаем таблицу `pg_repack`

Борьба с переполнением

Если вас посетила мысль о том, что всё это сложно, то это так и есть 😊

Борьба с переполнением

Если вас посетила мысль о том, что всё это сложно, то это так и есть 😊

Поэтому мы разработали небольшую утилиту, которая поможет вам при подобного рода миграциях.

Борьба с переполнением

```
create table "big"("id1" integer not null, "id2" integer not null, created_at date, content text);
insert into "big" select i, 2000000-i, now() - interval '1 day'*i, 'lorem ipsum' from generate_series (1,
2000000) as i;
create sequence "big_id1_seq" start 2000001;
create unique index "big_id_idx" on "big"("id1");
create unique index "big_id2_id1_idx" on "big"("id2","id1");
create index "big_created_at_part" on "big"(created_at) where "id1"<=5000;
alter table "big" add primary key using index "big_id2_id1_idx";
alter table "big" alter "id1" set default nextval('big_id1_seq')*2;
alter table "big" add constraint "big_id2_more_0" check ("id2" >=0);
alter table "big" add constraint "big_complex1" check ((id2>=0 and id1<30000000) or (created_at <= now())) not
valid;
alter table "big" add constraint "big_complex2" check ((id2>=0 and 30000000>id1) or (created_at <= now())) not
valid;
alter table "big" add constraint "big_complex3" check (id2>=0 and 30000000>id1) not valid;

create table big_child(id integer, id1_big int, constraint fk_big foreign key (id1_big) references big(id1) on
delete set null);
```

Борьба с переполнением

Table "public.big"

Column	Type	Collation	Nullable	Default
id1	integer		not null	(nextval('big_id1_seq'::regclass) * 2)
id2	integer		not null	
created_at	date			
content	text			

Indexes:

```
"big_id2_id1_idx" PRIMARY KEY, btree (id2, id1)
"big_created_at_part" btree (created_at) WHERE id1 <= 5000
"big_id_idx" UNIQUE, btree (id1)
```

Check constraints:

```
"big_complex1" CHECK (id2 >= 0 AND id1 < 300000000 OR created_at <= now()) NOT VALID
"big_complex2" CHECK (id2 >= 0 AND 300000000 > id1 OR created_at <= now()) NOT VALID
"big_complex3" CHECK (id2 >= 0 AND 300000000 > id1) NOT VALID
"big_i2_more_0" CHECK (id2 >= 0)
```

Referenced by:

```
TABLE "big_child" CONSTRAINT "fk_big" FOREIGN KEY (id1_big) REFERENCES big(id1) ON DELETE SET NULL
```

Борьба с переполнением

Table "PuB.BiG"

Column	Type	Collation	Nullable	Default
Id1	integer		not null	nextval('PuB"."BiG_id1_seq"::regclass)
Id2	integer		not null	
created_at	date			
content	text			

Indexes:

```
"BiG_Id2_Id1_idx" PRIMARY KEY, btree ("Id2", "Id1")
"BiG_created_at_part" btree (created_at) WHERE "Id1" <= 5000
"BiG_id_idx" UNIQUE, btree ("Id1")
```

Check constraints:

```
"BiG_I2_more_0" CHECK ("Id2" >= 0)
"big_Complex1" CHECK ("Id2" >= 0 AND "Id1" < 300000000 OR created_at <= now()) NOT VALID
"big_Complex2" CHECK ("Id2" >= 0 AND 300000000 > "Id1" OR created_at <= now()) NOT VALID
"big_Complex3" CHECK ("Id2" >= 0 AND 300000000 > "Id1") NOT VALID
```

Referenced by:

```
TABLE ""PuB"."BiG_child"" CONSTRAINT "fk_big_pub" FOREIGN KEY ("Id1_big") REFERENCES "PuB"."BiG"("Id1") ON DELETE SET NULL
```

Борьба с переполнением

```
psql -d db_name -f int_to_bigint.sql
```

Борьба с переполнением

```
psql -d db_name -f int_to_bigint.sql
Schema name: [public]: PuB
Table name: BiG
Затем выводится \dt+ от выбранной таблицы
int4 column name: Id1
Blocks in batch [1000]:
Sleep after N batches [50]:
Sleep duration N sec [1]:
Vacuum after every N% rows complited [10]:
vacuum_cost_delay (0..100 ms) [0]:
```

Это для версий 14 и выше, если версия ниже, то нужно ввести поле, по которому бить на пачки.

Борьба с переполнением

Information about table:

n_live_tuples	n_pages	tbl_size	indexes_size	total_size
2000000	12739	100 MB	86 MB	185 MB

(1 row)

index_name	index_size
BiG_Id2_Id1_idx	43 MB
BiG_id_idx	43 MB
BiG_created_at_part	128 kB

Борьба с переполнением

```
Created 'migr_BiG_step_1.sql' - Add column, trigger and function
Created 'migr_BiG_step_2.sql' - Copy data from old column to new
Created 'migr_BiG_step_3.sql' - Indexes and constraints
...
Created 'migr_BiG_step_4.sql' - Change columns
Created 'migr_BiG_step_5.sql' - Create commands for delete obsolete column
and indexes'
```

STEP 1

```
begin;
  set local statement_timeout to '1000ms';
  alter table "PuB"."BiG" add column "new_Id1" bigint;

  CREATE FUNCTION "PuB"."BiG_migr_f"()
  returns trigger as $$
  begin
    new."new_Id1" := new."Id1";
    return new;
  end $$ language plpgsql;

  CREATE TRIGGER "BiG_migr_t"
  before insert or update on "PuB"."BiG"
  for each row
  execute function "PuB"."BiG_migr_f"();
commit;
select 'The next step may take a long time. Try running it in tmux or screen!' as "Notice";
```

Борьба с переполнением

```
-bash-4.2$ psql -f migr_BiG_step_1.sql
BEGIN
SET
ALTER TABLE
CREATE FUNCTION
CREATE TRIGGER
COMMIT
```

Notice

The next step may take a long time. Try running it in tmux or screen!
(1 row)

STEP 1

Борьба с переполнением

```
set lock_timeout to '100ms';
set session_replication_role to 'replica';
set deadlock_timeout to '600s';
set vacuum_cost_delay to 0;
select now() as start_time \gset
\set cnt_err_vac 0
```

STEP 2

```
update "PuB"."BiG" set "new_Id1" = "Id1" where "Id1" is distinct from "new_Id1" and ctid
>='(0,0)' and ctid<'(1000,0)';
update "PuB"."BiG" set "new_Id1" = "Id1" where "Id1" is distinct from "new_Id1" and ctid
>='(1000,0)' and ctid<'(2000,0)';
```

Борьба с переполнением

```
select n_dead_tup >= 200000 as res from pg_stat_all_tables where relid = 213226 \gset
\if :res
  reset lock_timeout;
  vacuum "PuB"."BiG";
  select n_dead_tup < 200000 as res from pg_stat_all_tables where relid = 213226 \gset
  \if :res
    \set cnt_err_vac 0
  \else
    select :cnt_err_vac::int + 1 as cnt_err_vac \gset
    select :cnt_err_vac >= 3 as res \gset
    \if :res
      \echo 'Can not perform a vacuum on the table. There may be a competing long-running
transaction. Get rid of it and start over from step 2.'
      \q
    \endif
  \endif
  set lock_timeout to '100ms';
\endif
```

STEP 2

Борьба с переполнением

```
select date_trunc('sec',now()) as now, '4900/22739(21.0%)' as pages_processed,  
date_trunc('sec',now()-:'start_time'::timestamp) as elapsed,  
    date_trunc('sec',(now()-:'start_time'::timestamp)/round(4900*100/22739,1)*100 - (now()-  
:'start_time'::timestamp)) as estimate;  
select pg_sleep(1);
```

STEP 2

Борьба с переполнением

STEP 2

```
reset lock_timeout;
vacuum "PuB"."BiG";
select now()-:'start_time':::timestamp as total_elapsed;
select 'The non-updated rows are being counted, please wait.' as "Information";
select count(*) cnt from "PuB"."BiG" where "Id1" is distinct from "new_Id1" \gset

select :cnt = 0 as res \gset
\if :res
  \echo 'You can proceed to the next step.'
\else
  \echo 'The number of rows that have not been updated: ':cnt
  \echo 'It might be better if you repeat step 2.'
\endif

select 'Please check index and constraint list on step 3!' as "Notice";
```

Борьба с переполнением

STEP 2

```
-bash-4.2$ psql -f migr_BiG_step_2.sql
SET
SET
SET
SET
UPDATE 15700
UPDATE 15700
UPDATE 15700
```

```
UPDATE 15700
      now          | pages_processed | elapsed | estimate
-----+-----+-----+-----
 2024-04-05 13:19:04-04 | 4900/22739(21.0%) | 00:00:18 | 00:01:08
(1 row)

pg_sleep
-----
(1 row)
```

Борьба с переполнением

STEP 2

```
RESET
VACUUM
  total_elapsed
-----
 00:00:50.14351
(1 row)

                Information
-----
The non-updated rows are being counted, please wait.
(1 row)

You can proceed to the next step.

                Notice
-----
Please check index and constraint list on step 3!
(1 row)
```

STEP 3

```
CREATE INDEX CONCURRENTLY "_BiG_created_at_part" ON "PuB"."BiG" USING btree (created_at) WHERE ("new_Id1" <= 5000);
CREATE UNIQUE INDEX CONCURRENTLY "_BiG_Id2_Id1_idx" ON "PuB"."BiG" USING btree ("Id2", "new_Id1");
CREATE UNIQUE INDEX CONCURRENTLY "_BiG_id_idx" ON "PuB"."BiG" USING btree ("new_Id1");
CREATE INDEX CONCURRENTLY "_BiG_99304" on "PuB"."BiG"("Id1") where "Id1" is distinct from "new_Id1";
--Please check constraint list:
begin;
  set local statement_timeout = '1s';
  alter table "PuB"."BiG" add constraint big_Complex1_new CHECK ("Id2" >= 0 AND "new_Id1" < 30000000 OR created_at
<= now()) not valid;
  alter table "PuB"."BiG" add constraint big_Complex2_new CHECK ("Id2" >= 0 AND 30000000 > "new_Id1" OR created_at
<= now()) not valid;
  alter table "PuB"."BiG" add constraint big_Complex3_new CHECK ("Id2" >= 0 AND 30000000 > "new_Id1") not valid;
commit;
begin;
  set local statement_timeout = '1s';
  alter table "PuB"."BiG" add constraint BiG_new_Id1_not_null check ("new_Id1" is not null) not valid;
commit;
alter table "PuB"."BiG" validate constraint BiG_new_Id1_not_null;
```

Борьба с переполнением

STEP 3

```
-bash-4.2$ psql -f migr_BiG_step_3.sql
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
BEGIN
SET
ALTER TABLE
ALTER TABLE
ALTER TABLE
COMMIT
BEGIN
SET
ALTER TABLE
COMMIT
ALTER TABLE
```

STEP 4

```
set enable_seqscan to 0;  
update "PuB"."BiG" set "new_Id1" = "Id1" where "Id1" is distinct from "new_Id1";  
select now() as start_time \gset  
\timing on
```

```
BEGIN;
```

```
    set local statement_timeout to '20s';  
    lock table "PuB"."BiG" in access exclusive mode;
```

Производится замена колонок и т.д.

```
    drop trigger "BiG_migr_t" on "PuB"."BiG";  
    drop function "PuB"."BiG_migr_f"();
```

```
COMMIT;
```

```
\timing off
```

```
select now()-:'start_time'::timestamp as total_elapsed;
```

Предупреждение о том, что если вы используете пулер соединений, то, возможно, понадобится выполнить RECONNECT.

Ну и просьба всё проверить перед заключительным шагом

Борьба с переполнением

```
-bash-4.2$ psql -f migr_BiG_step_4.sql
SET
UPDATE 0
Timing is on.
BEGIN
Time: 0.185 ms
SET
Time: 0.048 ms
LOCK TABLE
Time: 0.103 ms
ALTER TABLE
Time: 12.873 ms
```

STEP 4

```
-----
Notice
-----
If using connection pooler you may need to perform reconnect since table definition was changed and the cached plan result type may also have changed.
(1 row)

Notice
-----
Please check the result before proceeding to step 5.
```

Борьба с переполнением

STEP 5

На самом деле является генератором команд, которые нужно сначала проверить и если всё правильно, то выполнить.

Удаление старых индексов

Удаление старой колонки

Валидация ограничений целостности

Борьба с переполнением

```
bash-4.2$ psql -f migr_BiG_step_5.sql
Pager is used for long output.

          Table "PuB.BiG"
  Column | Type          | Collation | Nullable |          Default          | Storage | Compression | Stats target | Description
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
old_id1 | integer       |           |          |                          | plain   |              |              |
id2     | integer       |           | not null |                          | plain   |              |              |
created_at | date         |           |          |                          | plain   |              |              |
content | text          |           |          |                          | extended|              |              |
id1     | bigint        |           | not null | nextval('PuB."BiG_id1_seq"::regclass) | plain   |              |              |
Indexes:
    "BiG_id2_id1_idx" PRIMARY KEY, btree ("id2", "id1")
    "BiG_created_at_part" btree (created_at) WHERE "old_id1" <= 5000
    "BiG_id_idx" UNIQUE, btree ("old_id1")
    "BiG_99304" btree ("old_id1") WHERE "old_id1" IS DISTINCT FROM "id1"
    "BiG_created_at_part" btree (created_at) WHERE "id1" <= 5000
    "BiG_id_idx" UNIQUE, btree ("id1")
Check constraints:
    "BiG_I2_more_0" CHECK ("id2" >= 0)
```

```
--Check the list of indexes to be deleted and execute the commands
drop index concurrently "PuB"."BiG_id_idx";
drop index concurrently "PuB"."BiG_created_at_part";
drop index concurrently "PuB"."_BiG_99304";
SET statement_timeout to '1000ms';
alter table "PuB"."BiG" drop column "old_id1";
```

Борьба с переполнением

```
n_live_tuples | n_pages | tbl_size | indexes_size | total_size
-----+-----+-----+-----+-----
      2000000 |    17438 | 136 MB   | 103 MB       | 239 MB
(1 row)
```

```
postgres=# select pgc.relname as index_name, pg_size_pretty(pg_rel
relid = pgc.oid where pgi.indrelid = 213226;
   index_name      | index_size
-----+-----
  BiG_Id2_Id1_idx  | 60 MB
  _BiG_created_at_part | 128 kB
  _BiG_id_idx       | 43 MB
(3 rows)
```

Борьба с переполнением

Table "PuB.BiG"

Column	Type	Collation	Nullable	Default
Id2	integer		not null	
created_at	date			
content	text			
Id1	bigint		not null	nextval('PuB"."BiG_id1_seq"::regclass)

Indexes:

```
"BiG_Id2_Id1_idx" PRIMARY KEY, btree ("Id2", "Id1")
"_BiG_created_at_part" btree (created_at) WHERE "Id1" <= 5000
"_BiG_id_idx" UNIQUE, btree ("Id1")
```

Check constraints:

```
"BiG_I2_more_0" CHECK ("Id2" >= 0)
"big_complex1_new" CHECK ("Id2" >= 0 AND "Id1" < 300000000 OR created_at <= now())
"big_complex2_new" CHECK ("Id2" >= 0 AND 300000000 > "Id1" OR created_at <= now())
"big_complex3_new" CHECK ("Id2" >= 0 AND 300000000 > "Id1")
```

Referenced by:

```
TABLE ""PuB"."BiG_child"" CONSTRAINT "fk_big_pub" FOREIGN KEY ("Id1_big") REFERENCES
"PuB"."BiG"("Id1") ON DELETE SET NULL
```

Борьба с переполнением

Скачать утилиту сейчас:

https://github.com/Nikitin-Alexandr/utils/blob/main/int_to_bigint.sql

Скачать утилиту (через пару недель после доклада):

https://github.com/dataegret/pg-utils/tree/master/bin/int_to_bigint.sql

Все наши скрипты:

```
git clone https://github.com/dataegret/pg-utils ~/stuff
```

Борьба с переполнением

Ограничения:

- Пока она думает, что мы работаем только с первичным ключом
- Пока не работает с `identity`
- Пока не работает с секционированными таблицами
- Пока не обрабатываю ситуацию в которой на поле завязан триггер
- Пока не обрабатываю ситуацию с длинными именами (более 59 символов, поскольку добавляю к имени `_new` и `_old`).

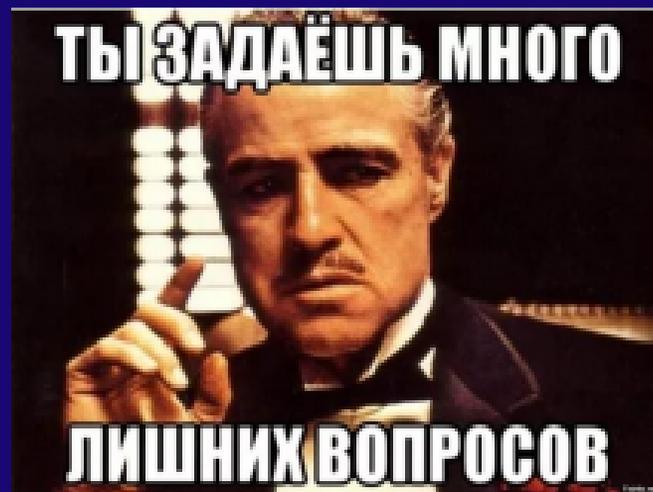
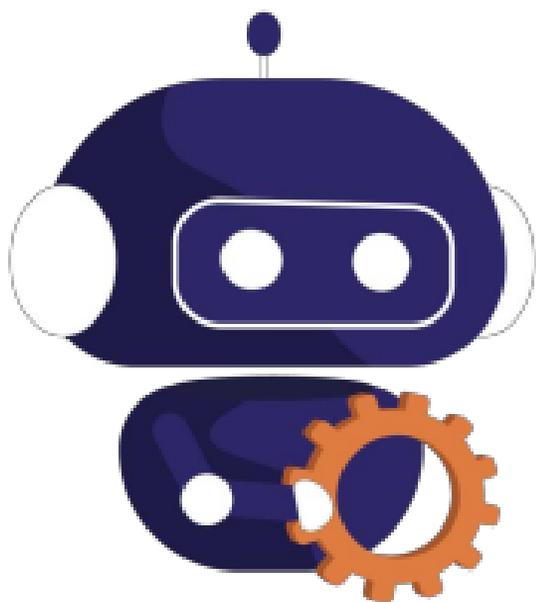
Планов по развитию, на самом деле много.

Спасибо за внимание!

contact@pgmech.ru



PGMechanix



Вопросы?

Александр Никитин

DBA

tg: <https://t.me/anikitindba>

<https://pgmech.ru/>

