Your remote PostgreSQL DBA team

data egret

Индексы в Postgresql.
Как понять, что создавать.

Сальников Андрей

Info Кто такие Data Egret

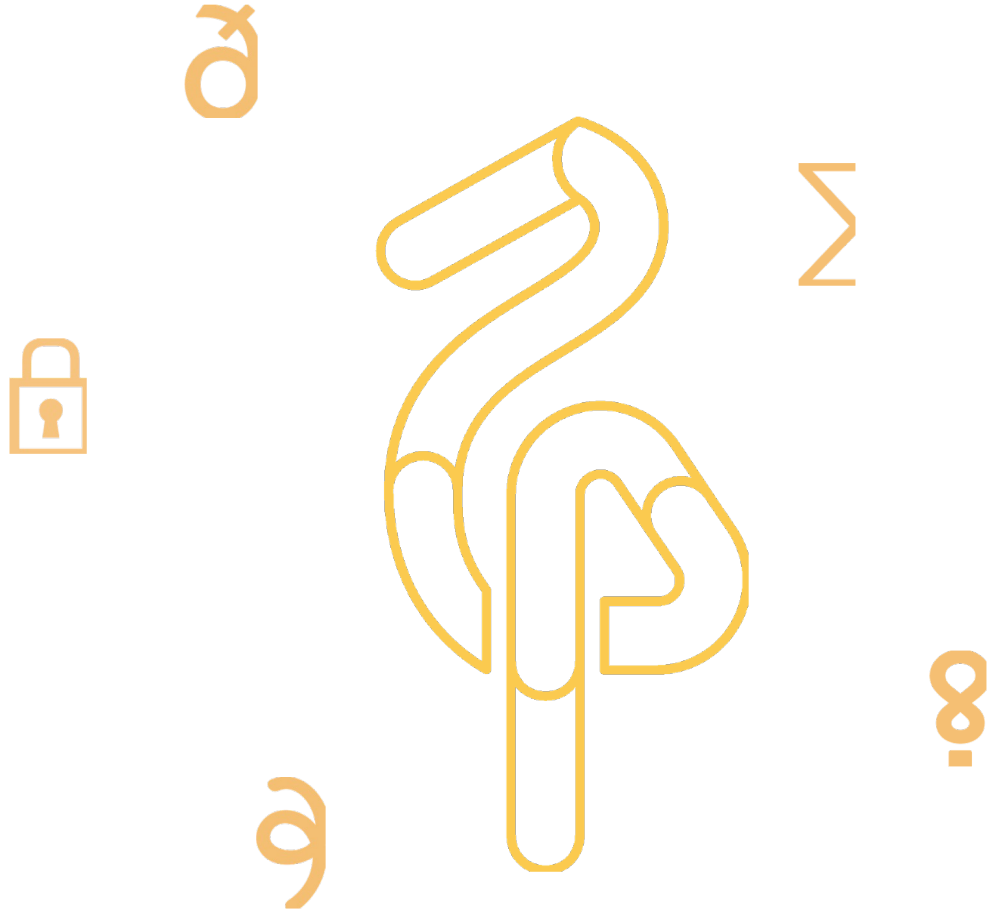
- Remote DBA PostgreSQL
- Консультанты PostgreSQL
- Постоянно готовим доклады на конференции
- Проводим мастер-классы

!!! Важные оговорки

- Будем говорить о OLPT нагрузке
  - WEB
  - API для WEB сервисов
  - Объемы баз от 20Гб до 10Тб
- Об OLAP не будем
  - Аналитика данных
  - Хранилища данных
  - Объемы баз больше 1Тб и до бесконечности

Indexes

С чего начать?



```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ]
  [ [ IF NOT EXISTS ] name ] ON [ ONLY ] table_name[ USING method ]
  ( { column_name | ( expression ) }
     [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
     [ NULLS { FIRST | LAST } ]
     [, ...] )
  [ INCLUDE ( column_name [, ...] ) ]
  [ WITH ( storage_parameter [= value] [, ... ] ) ]
  [ TABLESPACE tablespace_name ]
  [ WHERE predicate ]
```

ЗС Здравый смысл

- Что такое индексы?
  - Легализованные костыли для ускорения (pk)
- Какова расплата?
  - Замедление записи в таблицы (80% к 20%)
  - Дополнительные объемы дискового пространства
  - Усложненное техническое обслуживание (bloat)
- Все ли индексы полезны?
  - Нет, нет и нет.

3C Начальная информация

- Ориентироваться только на продуктовое окружение
  - Тестовые окружения не соответствуют реальности
  - Обладать статистикой нагрузки на БД от запросов
    - pg_stat_statments – хорошо
    - pgBadger – с осторожностью
  - Иметь примеры запросов с параметрами
    - Для понимания входящих параметров запроса
    - Необходимо для проверки

3C Начальная информация

- Уметь читать статистику распределения данных (планировщик)
- Представление pg_stats
  - Уникальность значений: n_distinct
  - Упорядоченность значений: correlation
  - Объемы null: null_frac
  - Частые значения: most_common_vals и most_common_freqs
- Вручную собрать более полную статистику (sample 30k)
  - Подозрения, что в статистике есть существенные промахи

ТИ Типы индексов

- btree
  - Наиболее распространённый тип индексов
  - Алгоритмы работы и модель хранения улучшаются
  - Покрывает 90% задач
  - Легко создать ориентируясь на статистику по таблице
- hash
  - Мой любимый тип индексов
  - Абсолютно бесполезен, используйте btree

ти Типы индексов

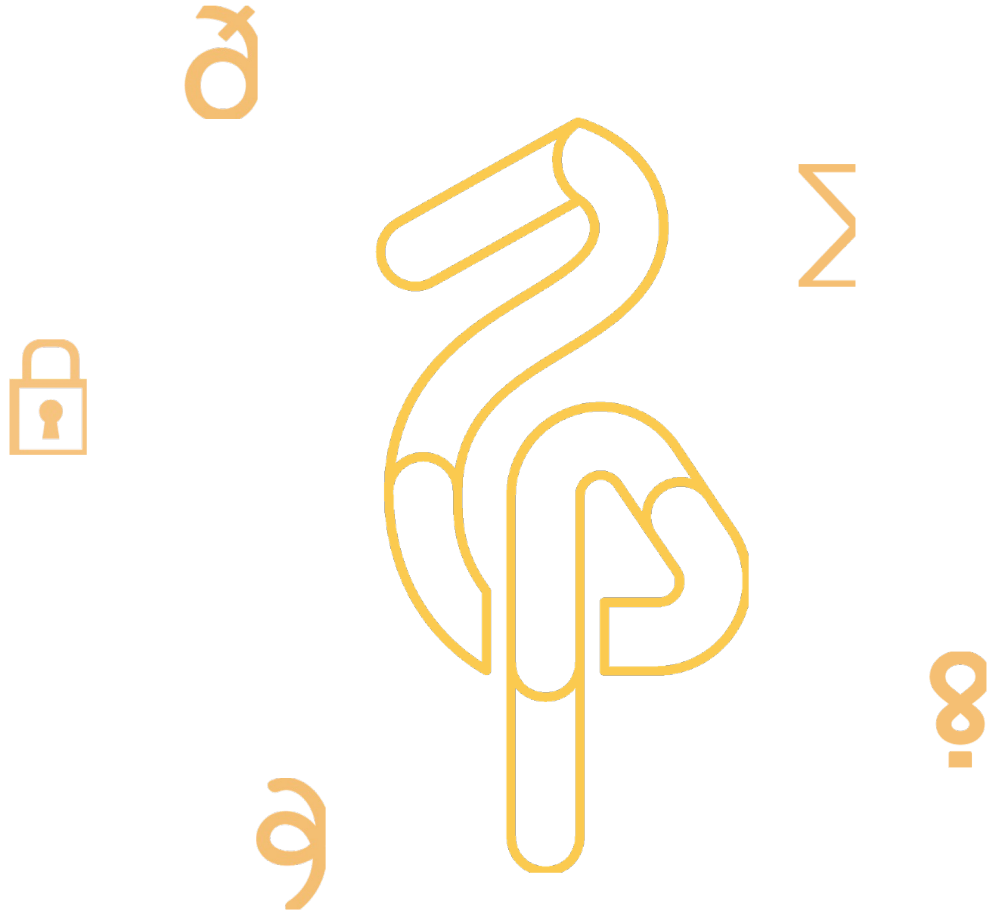
- gist
  - В чистом виде полезен для гео-данных
  - Расширения
    - pg_trgm – like, ilike, ~, ~* (regexp)
    - btree_gist – сложные constraints с интервалами
- sp_gist
  - … Loading …
  - Нет практических применений в OLTP

ТИ Типы индексов

- gin
  - Не сильно радует dba
  - Хорош для текстового поиска (+ pg_tgrm)
  - jsonb
    - jsonb_ops – умолчание
    - jsonb_path_ops – малый размер
- brin
  - Крайне компактный индекс

Indexes

null or not null



00 Таблица для тренировок

```
Table "public.pgconf"
   Column   |           Type           | Collation | Nullable | Default
------------+--------------------------+-----------+----------+---------
 id         | bigint                   |           | not null |
 fk_id      | bigint                   |           |          |
 state      | text                     |           |          |
 amount     | numeric                  |           |          |
 item       | text                     |           |          |
 created_at | timestamp with time zone |           |          |
Indexes:
    "pgconf_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "pgconf_fk_id_fkey" FOREIGN KEY (fk_id) REFERENCES pgconf(id)
```

01 Таблица для тренировок

rows count: **10,000,000**

```
 Schema |  Name  | Type  | Owner  |  Size
--------+--------+-------+--------+--------
 public | pgconf | table | andrey | 816 MB
```

```
 Schema |     Name     | Type  | Owner  | Table  |  Size
--------+--------------+-------+--------+--------+--------
 public | pgconf_pkey | index | andrey | pgconf | 214 MB
```

02 Удаляем строку

```
delete from pgconf where id = 10;

Delete on pgconf
   ->  Index Scan using pgconf_pkey on pgconf
         Index Cond: (id = 10)
 Planning Time: 0.043 ms
 Trigger for constraint pgconf_fk_id_fkey: time=690.455
calls=1
 Execution Time: 690.515 ms
```

03 Удаляем строку (под капотом)

```
select * from pgconf where fk_id = 10;

Gather
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on pgconf
         Filter: (fk_id = 10)
         Rows Removed by Filter: 3333333
 Planning Time: 0.059 ms
 Execution Time: 281.468 ms
```

04 Ускоряем проверку FK

```
create index fk on pgconf (fk_id);

Delete on pgconf
   ->  Index Scan using pgconf_pkey on pgconf
         Index Cond: (id = 10)
 Planning Time: 0.063 ms
 Trigger for constraint pgconf_fk_id_fkey: time=0.047
calls=1
 Execution Time: 0.101 ms
```

05 Ускоряем проверку FK (под капотом)

```
select * from pgconf where fk_id = 10;

Index Scan using fk on pgconf
   Index Cond: (fk_id = 10)
 Planning Time: 0.067 ms
 Execution Time: 0.027 ms
```

06 Смотрим статистику

```
select * from pg_stats where tablename = 'pgconf' and
attname = 'fk_id';
```

```
tablename              | pgconf
attname                | fk_id
null_frac              | 0.92943335
n_distinct             | -0.070566654
most_common_vals       |
most_common_freqs      |
correlation            | 0.0095442245
```

07 Уменьшаем проверку FK

```
create index fk_not_null on pgconf (fk_id)
 where fk_id is not null;


Delete on pgconf
   ->  Index Scan using pgconf_pkey on pgconf
         Index Cond: (id = 10)
 Planning Time: 0.069 ms
 Trigger for constraint pgconf_fk_id_fkey: time=0.044
calls=1
 Execution Time: 0.100 ms
```

08 Уменьшаем проверку FK (под капотом)

```
select * from pgconf where fk_id = 10;

Index Scan using fk_not_null on pgconf
   Index Cond: (fk_id = 10)
 Planning Time: 0.080 ms
 Execution Time: 0.027 ms
```

09 Что стало лучше с FK

Time: 690.713 ms vs 0.336 ms

Ускорение в 2055 раз!

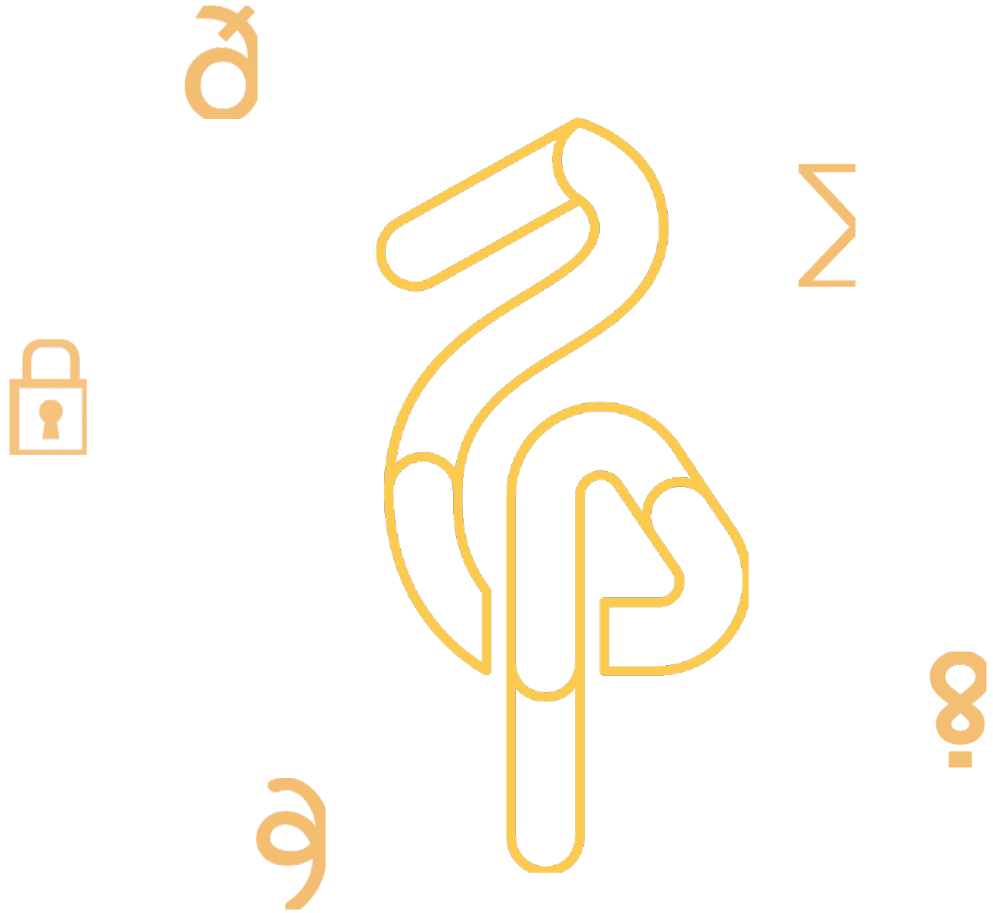
pgconf_pkey : 214 MB

fk : 215 MB

fk_not_null : 15 MB

Уменьшение размеров в 14 раз!

Indexes
partial



10 Поиск необработанных событий

```
select * from pgconf
 where state = 'ожидает'
 order by created_at
 limit 100;
```

11 Запрос в лоб

```
Limit
   ->  Gather Merge
         Workers Planned: 2
         Workers Launched: 2
         ->  Sort
               Sort Key: created_at
               Sort Method: top-N heapsort  Memory: 35kB
               Worker 0:  Sort Method: top-N heapsort  Memory: 35kB
               Worker 1:  Sort Method: top-N heapsort  Memory: 35kB
               ->  Parallel Seq Scan on pgconf
                     Filter: (state = 'ожидает'::text)
                     Rows Removed by Filter: 3003483
 Planning Time: 0.140 ms
 Execution Time: 362.268 ms
```

12 Странный вариант

```
create index strange on pgconf ((state = 'ожидает'));

Limit
   ->  Gather Merge
         ->  Sort
               Sort Key: created_at
               Sort Method: top-N heapsort  Memory: 35kB
               ->  Parallel Seq Scan on pgconf
                     Filter: (state = 'ожидает'::text)
                     Rows Removed by Filter: 3003483
 Planning Time: 0.093 ms
 Execution Time: 370.419 ms
```

13 Хороший вариант

```
create index normal on pgconf (created_at, state);
```

```
 Limit
   ->  Index Scan using normal on pgconf
         Index Cond: (state = 'ожидает'::text)
 Planning Time: 0.107 ms
 Execution Time: 0.138 ms
```

14 А если взглянуть на статистику?

```
attname          | created_at
null_frac        | 0
n_distinct       | -0.4638518
most_common_vals |
most_common_freqs|
histogram_bounds | {"2021-10-23 00:16:35.571674+03",
                 |  …, …}
correlation      | 1
```

15 А если взглянуть на статистику?

attname	state
null_frac	0
n_distinct	3
most_common_vals	**{обработано,ожидает,ошибка}**
most_common_freqs	**{0.89863336,0.09996667,0.0014}**
correlation	0.81656545

16 Почти идеал

```
create index perfect on pgconf (created_at)
 where state != 'обработано';

 Limit
   ->  Index Scan using perfect on pgconf
         Filter: (state = 'ожидает'::text)
         Rows Removed by Filter: 2
 Planning Time: 0.120 ms
 Execution Time: 0.105 ms
```

17 Что стало лучше

```
Time: 690.713 ms vs 0.105 ms
```

Ускорение в 3450 раз!

```
pgconf_pkey : 214 MB
strange     : 215 MB
normal      : 464 MB <-
perfect     : 21 MB  <-
```

Уменьшение размеров в 22 раза!

18 Бонусный вариант

~~create index normal on pgconf (created_at, state);~~

create index expected on pgconf (state, created_at);

```
Limit
   ->  Index Scan using expected on pgconf
         Index Cond: (state = 'ожидает'::text)
 Planning Time: 0.087 ms
 Execution Time: 0.112 ms
```

19 Стало лучше?

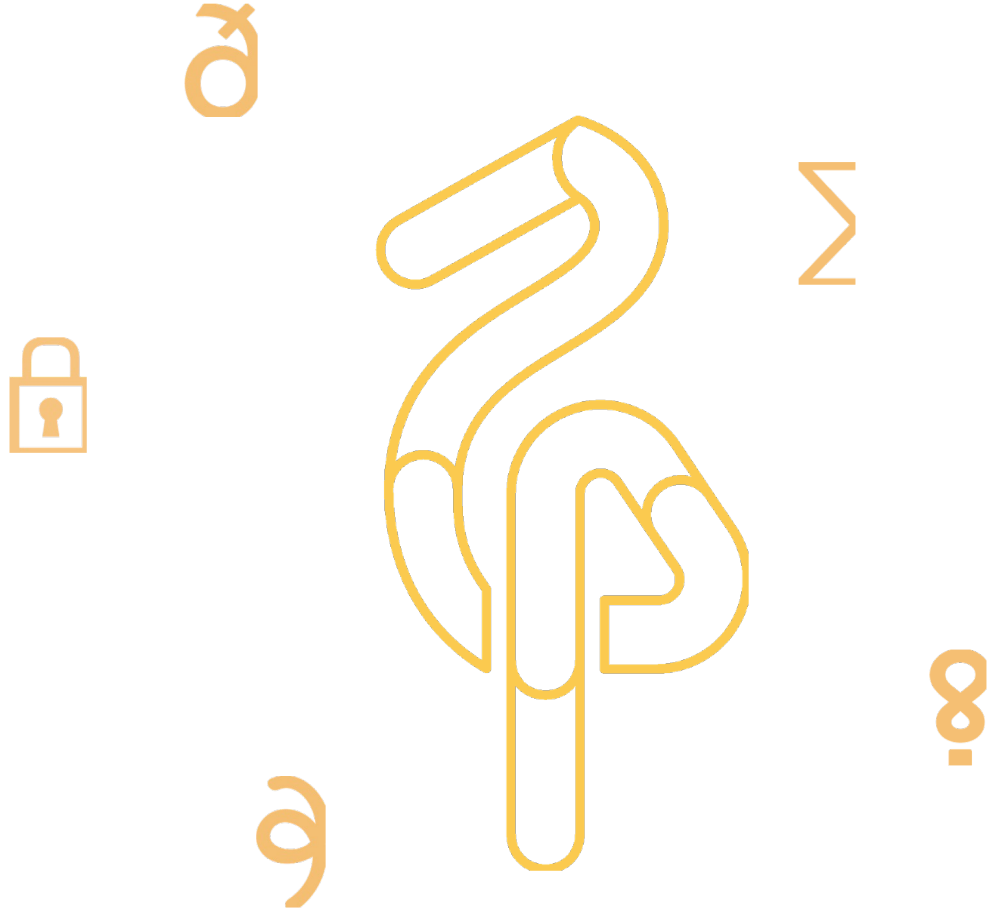
```
Time: 690.713 ms vs 0.105 ms
```

Ускорение в 3450 раз!

```
pgconf_pkey : 214 MB
strange     : 215 MB
normal      : 464 MB <-
expected    : 466 MB <-
perfect     : 21 MB  <-
```

Уменьшение размеров в 22 раза!

Indexes
Когда очень,
очень нужно!



20 Онлайн статистика

```
select sum (amount)
  from pgconf
 where state = 'обработано'
   and item = 'апельсин'
   and created_at between '2021-10-23 00:00'
                      and '2021-10-24 00:00';
```

21 План без индексов

```
Finalize Aggregate
  ->  Partial Aggregate
        ->  Parallel Seq Scan on pgconf
              Filter: ((created_at >= '2021-10-23 00:00') AND
                       (created_at <= '2021-10-24 00:00') AND
                       (state = 'обработано'::text) AND
                       (item = 'апельсин'::text))
              Rows Removed by Filter: 3323792
Planning Time: 0.087 ms
Execution Time: 357.851 ms
```

22 А если взглянуть на статистику?

```
attname           | state
null_frac         | 0
n_distinct        | 3
most_common_vals  | {обработано,ожидает,ошибка}
most_common_freqs | {0.89863336,0.09996667,0.0014}
correlation       | 0.81656545
```

23 А если взглянуть на статистику?

```
attname            | item
null_frac          | 0
n_distinct         | 5
most_common_vals   | {дыня,тыква,апельсин,
                   |  яблоко,персик}
most_common_freqs  | {0.59943336,0.27633333,
                   |  0.0994,0.022,0.0028333333}
correlation        | 0.44420442
```

24 Хороший индекс

```
create index normal on pgconf (created_at, item);

 Aggregate
   ->  Index Scan using normal on pgconf
         Index Cond: ((created_at >= '2021-10-23 00:00') AND
                      (created_at <= '2021-10-24 00:00') AND
                      (item = 'апельсин'::text))
         Filter: (state = 'обработано'::text)
         Rows Removed by Filter: 3122
 Planning Time: 0.124 ms
 Execution Time: 32.788 ms
```

25 Шаг к колоночным базам

```
create index special on pgconf (created_at, item)
 include (amount) where state = 'обработано';

 Aggregate
   ->  Index Only Scan using special on pgconf
         Index Cond: ((created_at >= '2021-10-23 00:00') AND
                      (created_at <= '2021-10-24 00:00') AND
                      (item = 'апельсин'::text))
         Heap Fetches: 28624
 Planning Time: 0.144 ms
 Execution Time: 30.084 ms
```

17 Что стало лучше

```
Time: 357.851 ms vs 30.084 ms
```

Ускорение в 11 раз!

```
pgconf_pkey : 214 MB
normal      : 395 MB
special     : 379 MB
```

Special хоть и перегружен, но меньшего размера.

Спасибо!

Андрей Сальников

andrey.salnikov@dataegret.com