ВНИИЖТ
НЦ ЦМПЭ

Вакуумотерапия: лечим хронические заболевания БД

Анатолий Анфиногенов

Краткое содержание предыдущих серий

2020 Как я перестал беспокоиться и перенес 60K строк из 150 процедур PL/SQL в Postgres
https://pgconf.ru/2020/274661

2021 Миграция приложения Oracle PL/SQL на Postgres pl/pgSQL: взгляд два года спустя
https://pgconf.ru/202110/308499

2021 Миграция приложения Oracle PL/SQL на Postgres pl/pgSQL: планирование, подготовка, переход и два года жизни с новой БД
https://conf.ontico.ru/lectures/3829870

2022 Жизнь после импортозамещения: некоторые особенности настройки БД и хранимых процедур
https://pgconf.ru/2022/316102

2022 Ускоряем хранимые процедуры на Postgres pl/pgSQL по гистограммам или жизнь после импортозамещения
https://highload.ru/moscow/2022/abstracts/9367

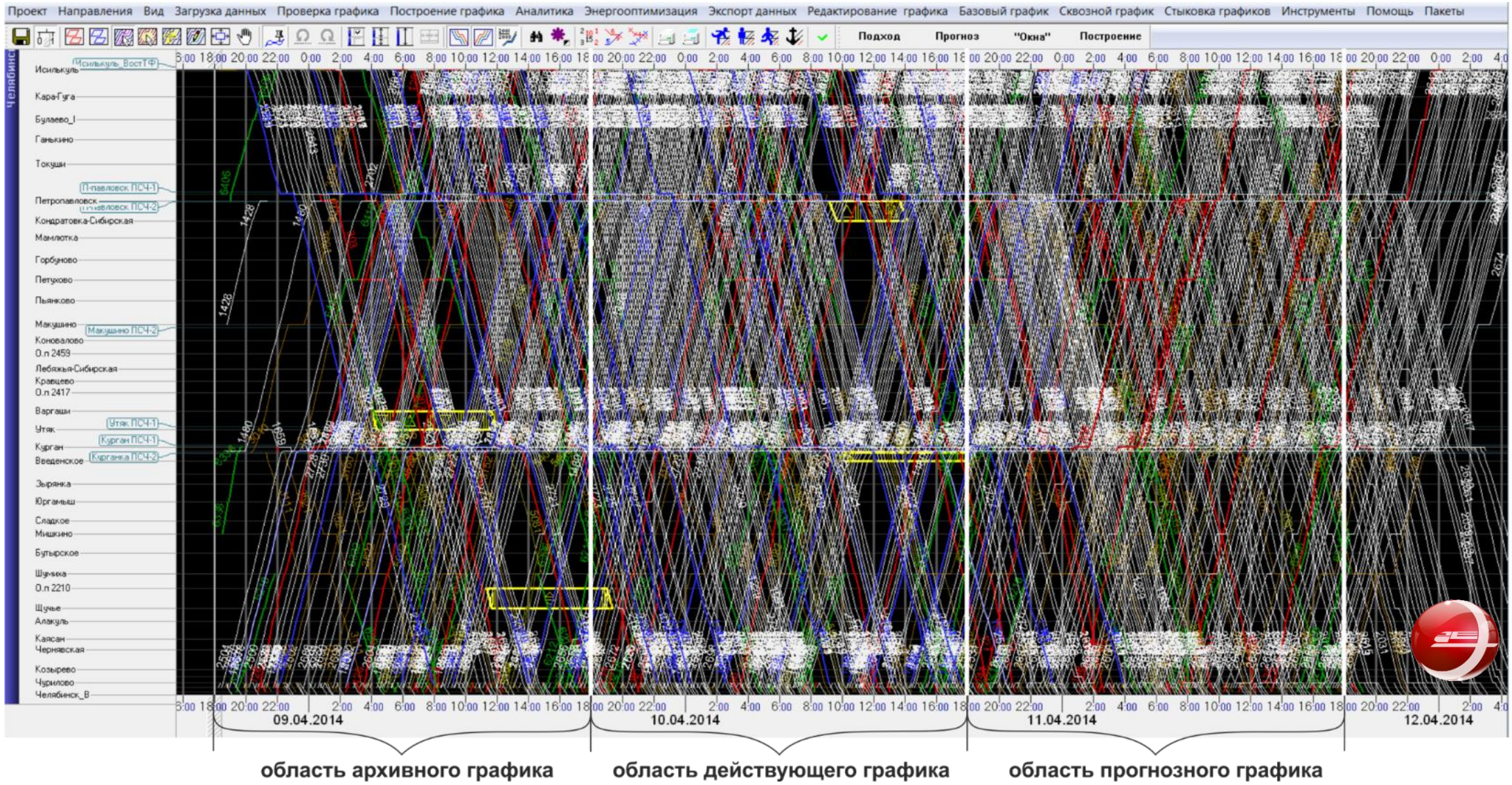
Что это и зачем это нужно?

- ЭЛЬБРУС (2006 г. – н.в.) – система планирования движения грузовых поездов по энергооптимальным расписаниям

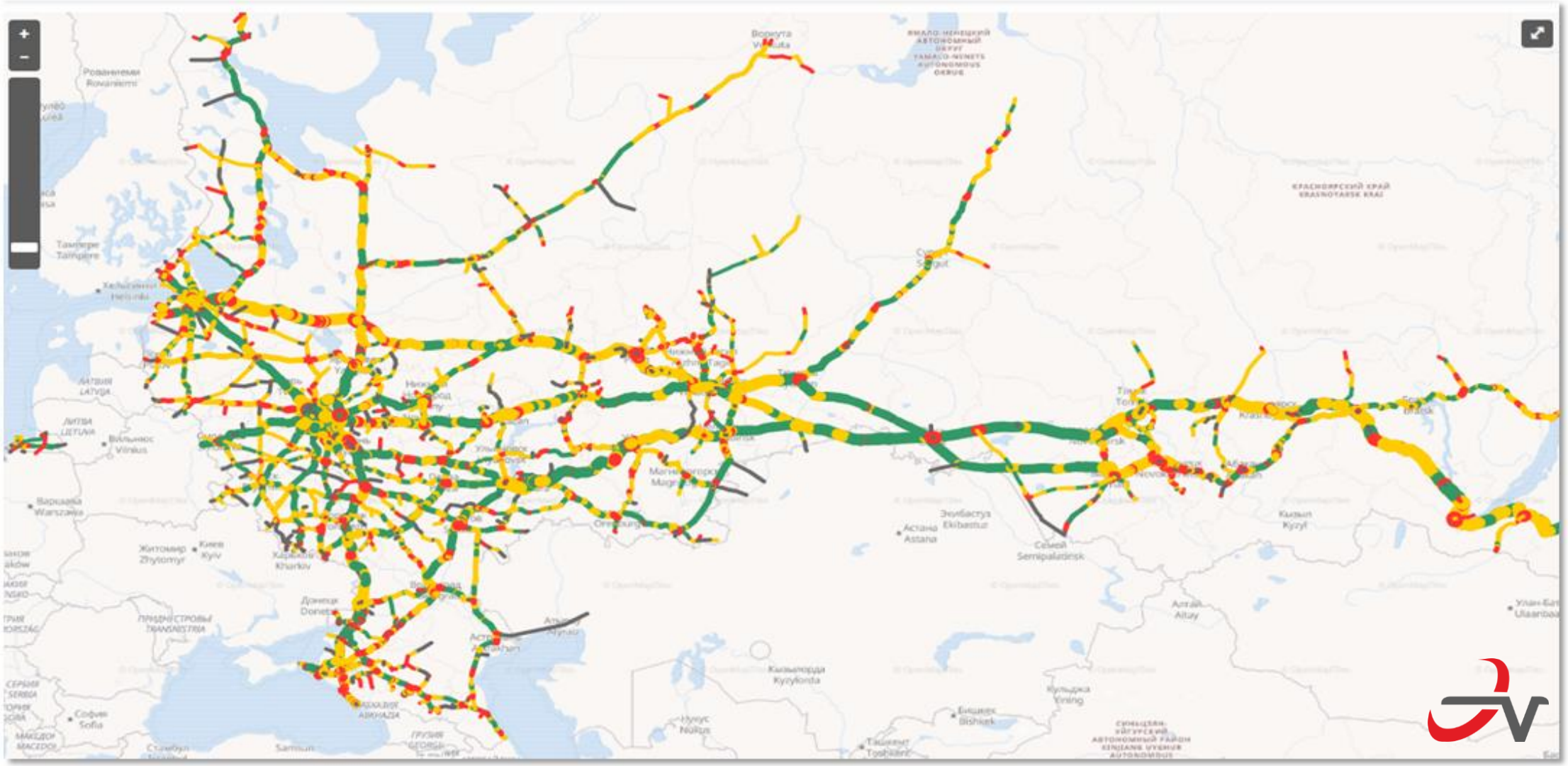
- ЭЛЬБРУС работает на всех железных дорогах России от Калининграда до Хабаровска

Первая премия UIC/МСЖД–2012 в области железнодорожных исследований и инноваций

Так выглядит расписание поездов снаружи



Так выглядит прогнозная аналитика



Так устроено расписание поездов изнутри

- 1 расписание = до 2 млн объектов 20 типов
- 1 расписание = до 500 Мбайт для 1 железной дороги
- 16 железных дорог России += ежедневно
  10-30 расписаний разных типов
- 1 дорога = 500 расписаний в оперативной базе
- Центральная архивная БД = история расписаний
  и прогнозная аналитика

Архитектура ЭЛЬБРУС: трехзвенная

- Распределенность: 16 узлов на железных дорогах и сервера центрального уровня
- Эксплуатация: 24/7; регулярные обновления серверных приложений 3-6 раз в год, включая приложения БД
- Толстый клиент: Windows, C++, очереди (ActiveMQ)
- Тонкий клиент: GWT, Angular
- Сервер приложений: Java, Tomcat, очереди (ActiveMQ)
- Сервер БД: ванильный Postgres 11/13

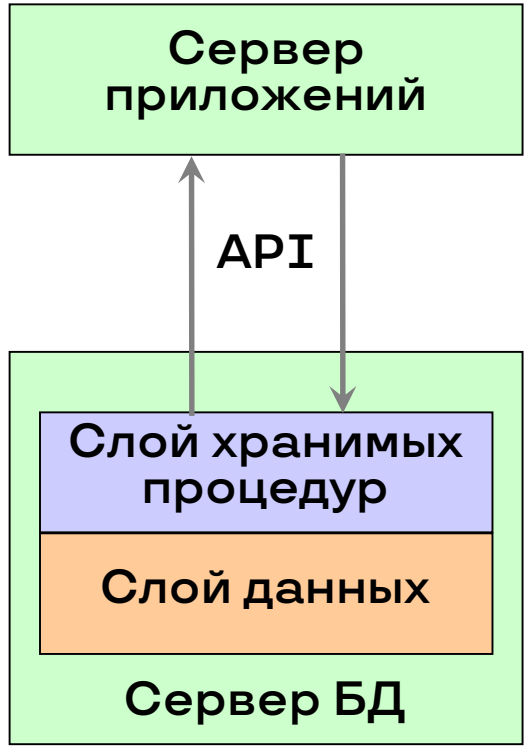
Используемые расширения PostgreSQL

Расширение	Назначение в проекте	Штатное?
dblink	Эмуляция автономных транзакций	да
pg_variables	Эмуляция автономных транзакций	нет (да в Pro)
pgcrypto	Генерация GUID	да
plpgsql_check	Проверка хранимых процедур (необязательна)	нет
plpythonu	Работа с файлами в ФС сервера	да
postgres_fdw	Связь с архивными серверами	да
oracle_fdw	Интеграция со смежными системами	нет (да в Pro)
pg_stat_statements	Мониторинг производительности БД	да
pgstattuple	Проверка «распухания» таблиц и индексов	да
pg_repack	Лечение «распухания» таблиц и индексов	да
pg_hint_plan	Настоящие хинты для планировщика запросов	нет

Нештатные расширения затрудняют обновление СУБД!

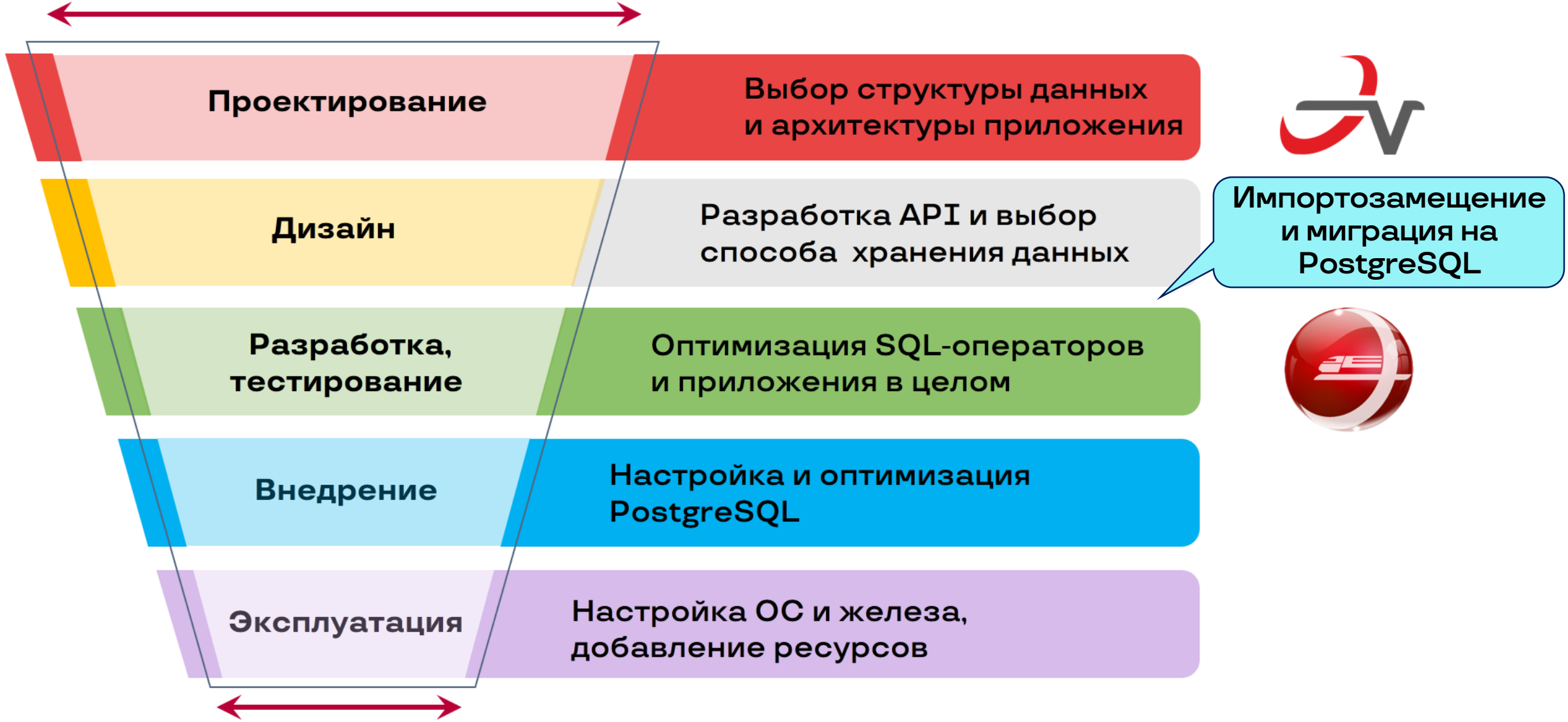
Почему только хранимые процедуры?

- Взаимодействие в предметных категориях приложения.
- Логика хранения с помощью API ХП отделена от бизнес-логики приложения;
- Можно вносить изменения в структуры данных и организацию БД без изменения сервера приложений (в пределах API).
- Возможно гладкое поэтапное обновление распределенного ПО за счет версионности API.
- Встроенный механизм диагностики, логирования, отладки и профилирования приложения БД без остановки сервиса, включаемый и управляемый параметрически.



Управление производительностью БД

Возможность влиять на производительность



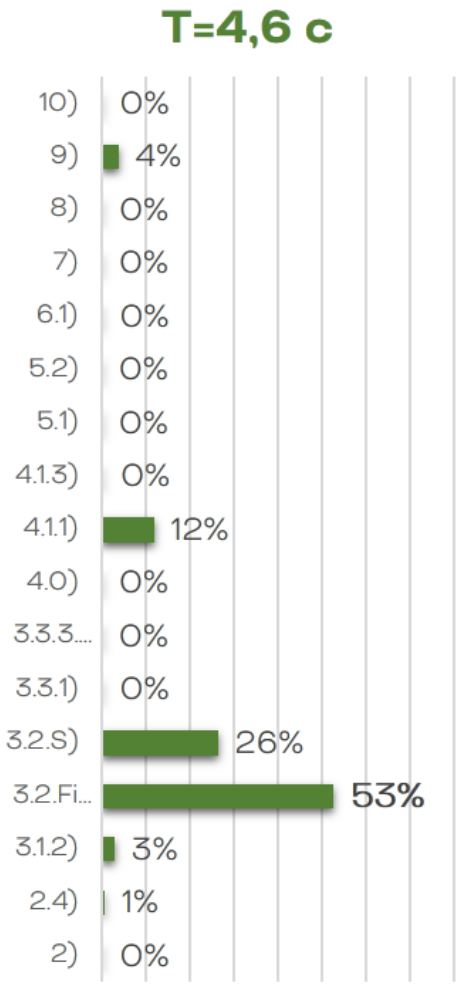
Где измерять производительность ХП?

Место измерения	+	−
На стороне сервера приложений	Низкие накладные расходы	Нет подробностей: что, где, почему и как это исправить
Изнутри хранимой процедуры	Максимальная точность и информативность. Данные о скорости работы ХП внутри БД	Накладные расходы. Необходимость модификации кода
Со стороны БД	Не надо модифицировать приложение. Есть pg_profile (Андрей Зубков)	Недостаточная информативность. Накладные расходы

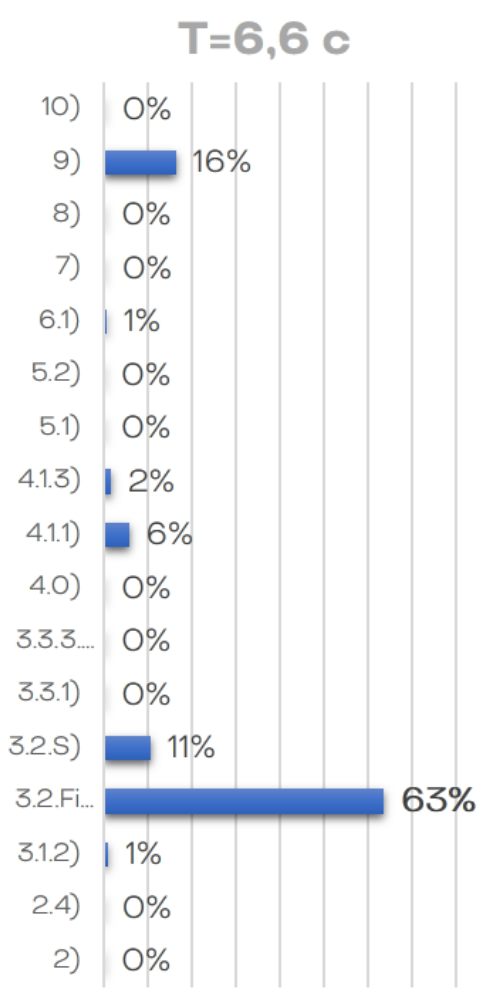
Профилирование хранимых процедур

- Возможность профилировать хранимые процедуры должна быть заложена при проектировании
- Процедура разбита на этапы (один или несколько сходных по типу SQL-операторов); времена их выполнения сохраняются в логи производительности
- Профилирование должно включаться и управляться параметрически с возможностью отключения; времена выполнения должны попадать в мониторинг
- Необходимо отслеживать планы выполнения проблемных запросов изнутри хранимой процедуры (было в ХП для Oracle, но ~~это невозможно~~ теперь возможно – ура, мы научились!)

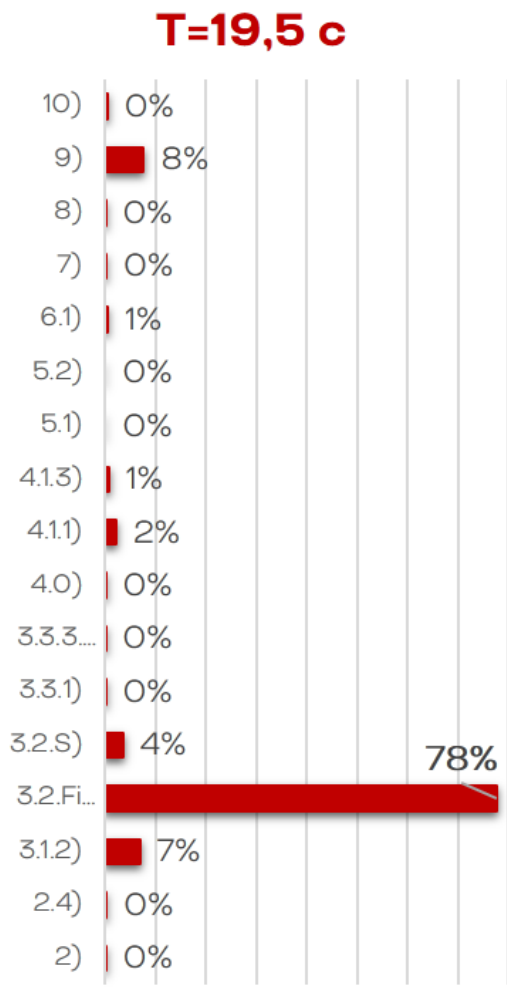
Гистограммы: THE GOOD, THE BAD AND THE UGLY



отлично

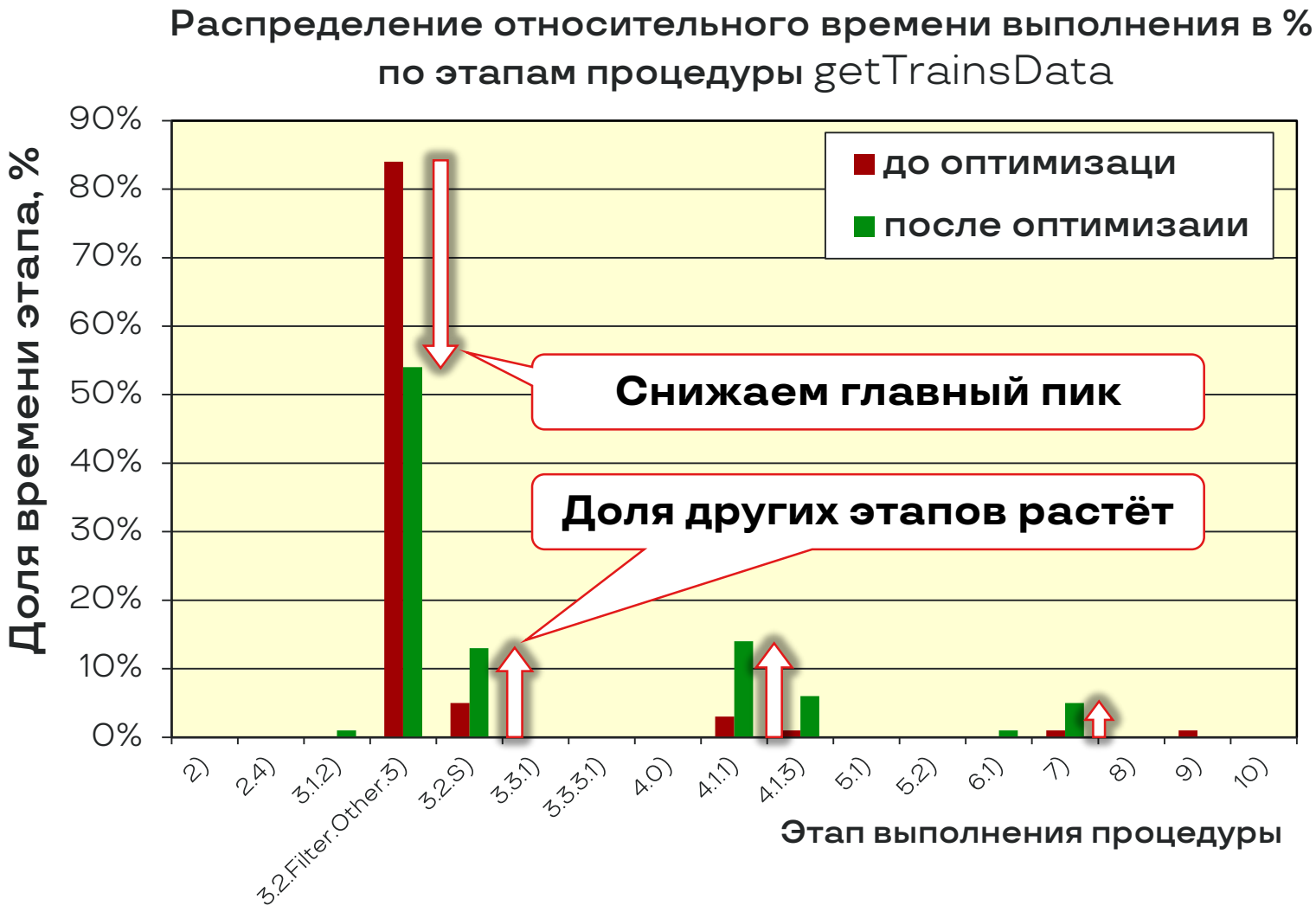


приемлемо



плохо

Гистограмма до и после оптимизации



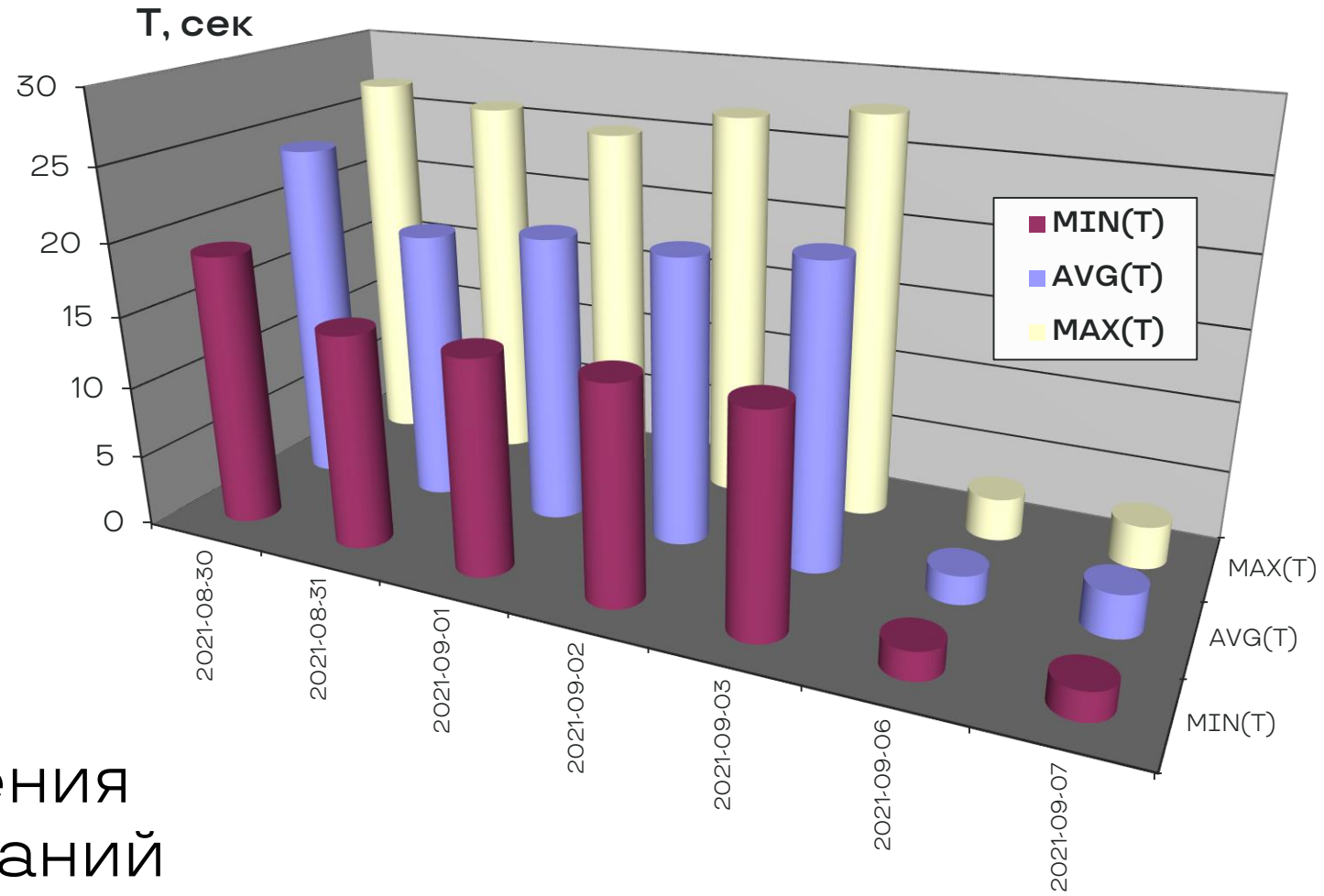
- Гистограмма времени выполнения этапов процедуры позволяет выявить «узкие места» — этапы, на которые ушло более чем 80% времени выполнения процедуры
- Цель — по возможности более равномерная гистограмма!

Как это было достигнуто? Выявлением и оптимизацией «плохих» запросов

Диагностируем и лечим хронические заболевания БД

Проявления хронических заболеваний

Первое проявление в виде проблемы с индексами: при объединении двух основных по размеру таблиц возникал Sec Scan вместо обращения по индексу.



После ~~исправления~~ симптоматического лечения скорость чтения расписаний возросла в среднем в 15 раз (временно).

Проблема с Sec Scan по главной таблице

GID_Events
Train_EID
…

142 млн.строк

TMP_Trains
Train_EID
…

3682 строки

GID_Events имеет индекс по Train_EID

```
SELECT E.*
FROM   GID_Events E INNER JOIN
       TMP_Trains T
       ON(T.Train_EID=E.Train_EID);
```

```
QUERY PLAN
Hash Semi Join  (cost=170.84..4792890.47 rows=11787245 width=286) (actual time=13254.652..135489.742
rows=816779 loops=1)
  Hash Cond: (e.train_eid = t.train_eid)
  ->  Seq Scan on gid_events e  (cost=0.00..4287162.24 rows=142637824 width=286) (actual
time=0.217..117560.619 rows=136429750 loops=1)
  ->  Hash  (cost=124.82..124.82 rows=3682 width=8) (actual time=2.882..2.892 rows=3682 loops=1)
        Buckets: 4096  Batches: 1  Memory Usage: 176kB
        ->  Seq Scan on tmp_trains t  (cost=0.00..124.82 rows=3682 width=8) (actual time=0.036..2.109
rows=3682 loops=1)
Planning Time: 5.428 ms
Execution Time: 135523.087 ms
```

Временное решение проблемы с Sec Scan

Увеличить глубину сбора статистики по колонке с ПК:

```
ALTER TABLE GID_Events ALTER COLUMN Train_EID SET STATISTICS 10000;
ANALYZE GID_Events;
```

```
QUERY PLAN
Nested Loop  (cost=134.59..774248.40 rows=193491 width=286) (actual time=30.496..6586.509 rows=816779
loops=1)
  ->  HashAggregate  (cost=134.03..170.84 rows=3682 width=8) (actual time=3.238..7.687 rows=3682 loops=1)
        Group Key: t.train_eid
        ->  Seq Scan on tmp_trains t (cost=0.00..124.82 rows=3682 width=8) (actual time=0.034..1.865
rows=3682 loops=1)
  ->  Index Scan using gid_events_pk on gid_events e  (cost=0.57..209.70 rows=53 width=286) (actual
time=0.762..1.674 rows=222 loops=3682)
        Index Cond: (train_eid = t.train_eid)
Planning Time: 6.667 ms
Execution Time: 6619.910 ms
```

Ускорение в 20 раз!

Хроническое заболевание прогрессирует

Со временем скорость работы снизилась до неприемлемых значений, при объединении двух основных по размеру таблиц снова возникал Sec Scan вместо обращения по индексу.

Прежний метод не помогал, перестроение индексов тоже.

Идеи по поводу простых решений также закончились...

А давайте попробуем хинты (которых «нет»)

- Хинтов в стиле Oracle в ванильном PostgreSQL нет, но давать советы планировщику все же можно.
- Менять нужно только настройки текущей сессии и не забыть восстанавливать исходное состояние настроечного параметра.

```
SET LOCAL enable_seqscan = OFF; -- Порицаем использование Sec Scan
… -- Выполняем запрос
SET LOCAL enable_seqscan TO DEFAULT; -- Возвращаем, как было
```

Ненадолго помогает, но так как причина не выявлена и не устранена, вскоре снова становится плохо…

Спойлер: до pg_hint_plan мы додумаемся чуть позднее…

Причина Sec Scan: не работал autovacuum

Настройки **autovacuum** по умолчанию вызывают такой эффект в случае нашей БД.

Потребовалось изучить вопрос с **autovacuum** и внести изменения в настройки по умолчанию!

Спасибо Егору Рогову и Алексею Лесовскому!
Их статьи по теме очень помогли.

1) https://habr.com/ru/company/postgrespro/blog/452762/
2) https://dataegret.com/category/autovacuum/

1) 2)

Симптом проблем с autovacuum

- Простой SELECT из интерфейсной UNLOGGED-таблицы внезапно стал занимать 95% времени работы процедуры независимо от числа записей в этой таблице (плоть до 0).
- По состоянию UNLOGGED-таблиц мы видим, что `autovacuum` к ним не применяется!

table_name	n_live_tuples	n_dead_tuples
tmp_trains	0	521 457
tmp_timetablepoints	0	15 048 410
tmp_traincalendars	0	8 544 236

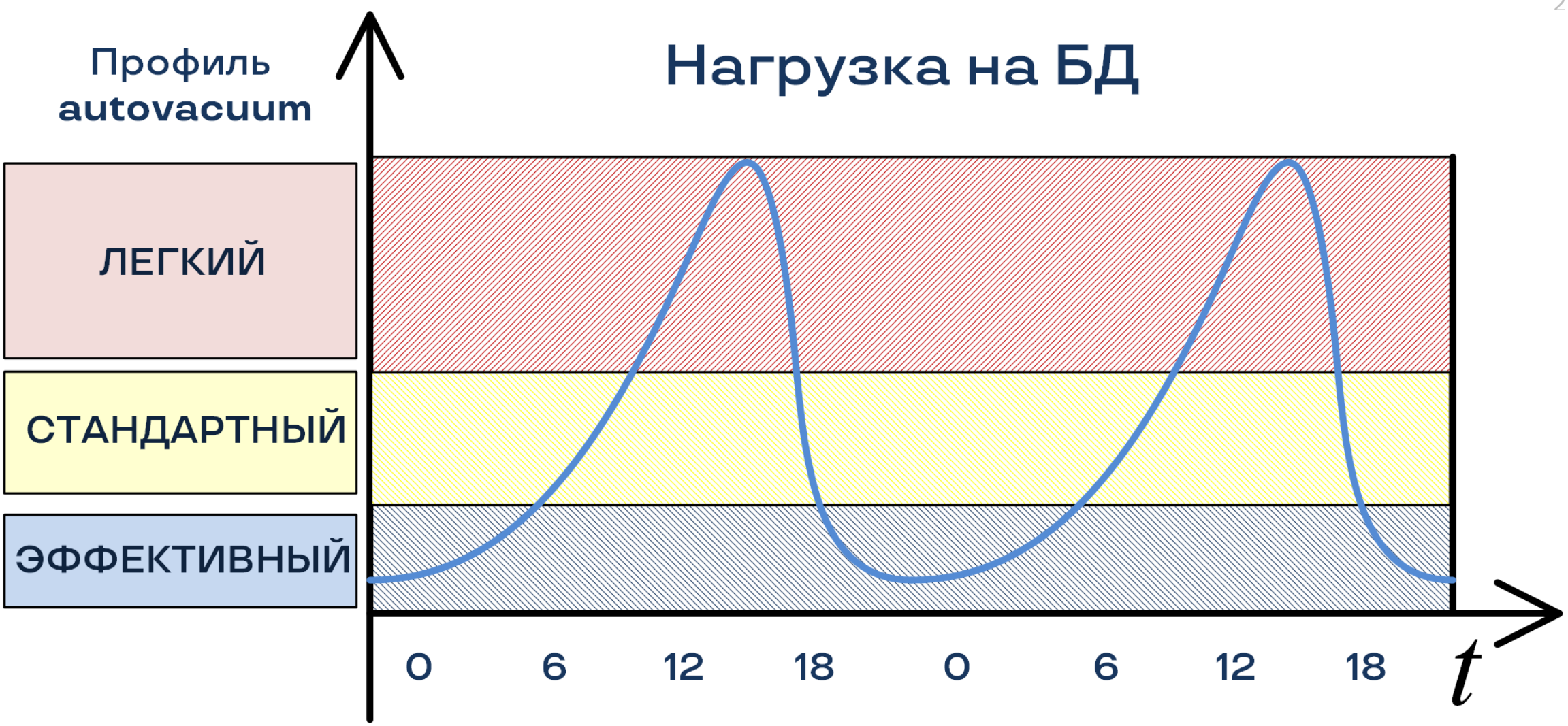
Укрощаем VACUUM

ВНИИЖТ
НЦ ЦМПЭ

Зачем настраивать autovacuum?

- До 60% содержимого основных таблиц в нашей БД обновляется за неделю.
- Настройки **autovacuum** по умолчанию таковы, что отработать интенсивно обновляемые таблицы он не успевает. Кстати, начиная с версии 12 настройки по умолчанию более активные.
- Статистика по таблицам и индексам собирается в ходе выполнения **autoanalyze**; в итоге статистика не собиралась и устаревала; поэтому в запросах переставали использоваться индексы и возникал Sec Scan.
- Параметры работы **autovacuum** и **autoanalyze** можно настроить индивидуально для отдельных таблиц.

Периодичность нагрузки на нашу БД



Пусть autovacuum ночью работает в полную силу, а днём не мешает!

Введем профили autovacuum для таблиц БД

- Таблицы БД можно разбить на 3 группы по отношению требуемой частоте автовакуума:
  1) Агрессивный VACUUM ('AGRESSIVE');
  2) Стандартный VACUUM ('STANDARD');
  3) Необременительный VACUUM ('LAZY').
- Профиль вакуумирования – распределение таблиц БД по группам потребности в autovacuum. Определен в соответствкющих таблицах.
- Можно иметь разные профили для разных уровней нагрузки БД и переключать их по расписанию или ситуативно вызовом хранимой процедуры.

Параметры профилей autovacuum

Профиль задает индивидуальные значения для входящих в него групп таблиц командой `ALTER TABLE … SET(…)`

Параметр настройки автовакуума	Значение для AGRESSIVE	Значение для STANDARD	Значение для LAZY
autovacuum_vacuum_scale_factor	0	0	0
autovacuum_vacuum_threshold	10	50	1000
autovacuum_vacuum_cost_delay (по умолчанию в 11 было 20)	1	2	5
autovacuum_vacuum_cost_limit (по умолчанию в 11 было 200)	1000	500	200

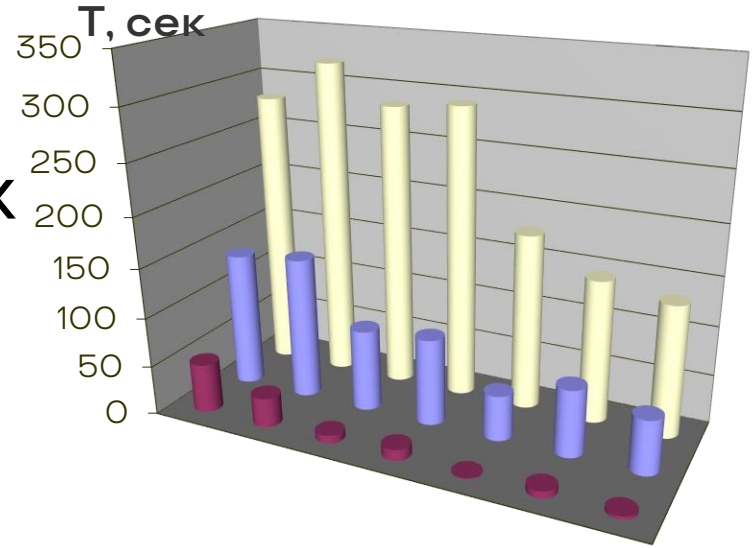
Профиль применяется автоматически с помощью job-а!

Подумав, для простоты оставили только AGGRESSIVE и LAZY

Наш рецепт борьбы с bloat:

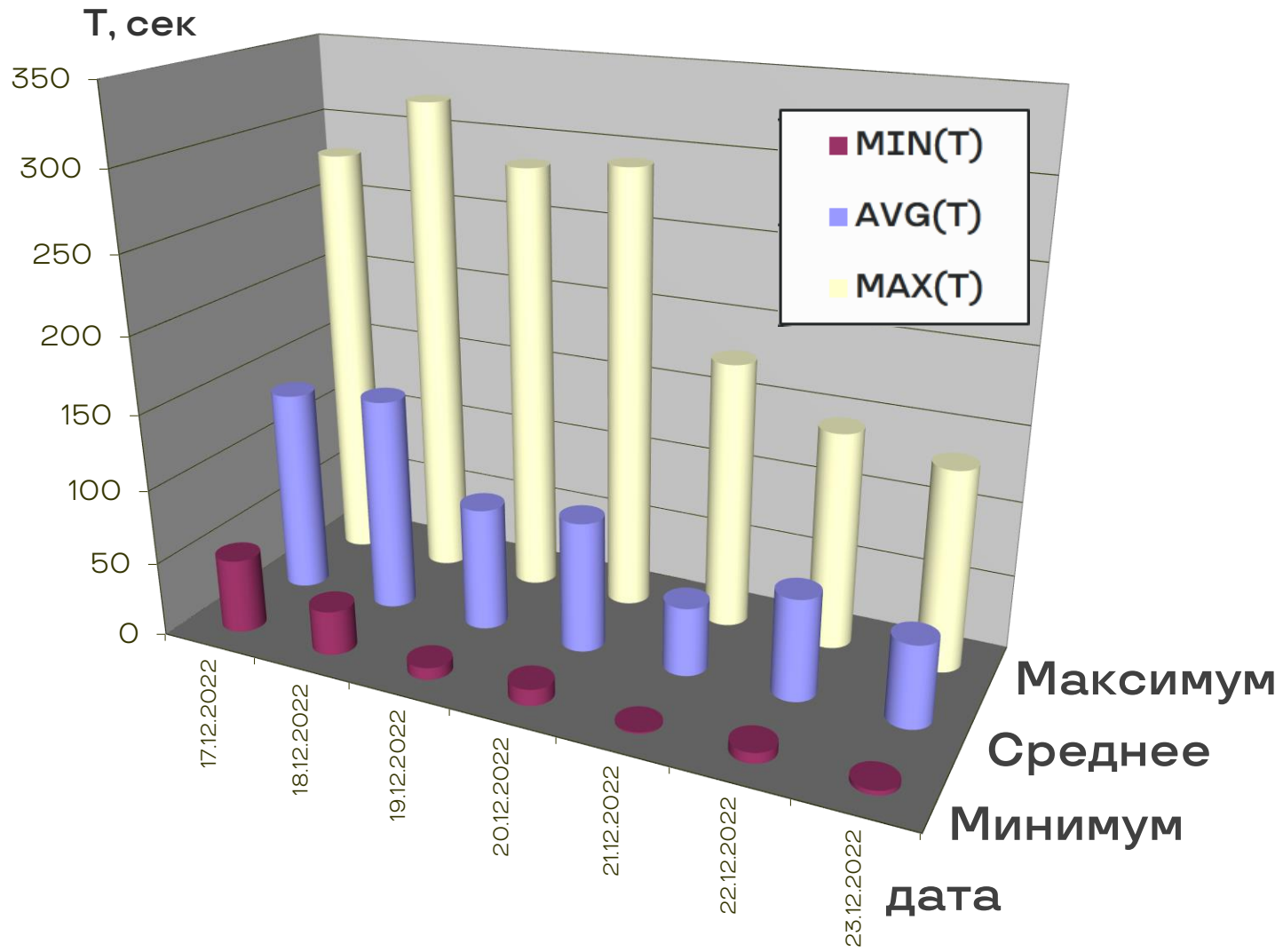
pg_repack, autovacuum и VACUUM

- Стартовая реорганизация основных таблицы и индексов БД при внедрении этого подхода – pg_repack
- 1 раз в 2 часа – VACUUM интерфейсных UNLOGGED-таблиц TMP_*
- Каждую ночь в периоды наименьшей нагрузки – VACUUM основных таблиц
- Индивидуально настроен autovacuum
- Результат: процедура saveTrainTimetable в среднем ускорилась в 2 раза!



Подробности далее

Результаты борьбы с bloat



Процедура saveTrainTimetable **ускорилась в 2 раза**

Проверяем результаты борьбы с bloat

```
-- Состояние вакуумирования основных таблиц приложения
SELECT *
FROM   VW_VACUUM_States
ORDER BY Table_Name;
```

Table	N_Live	N_Dead	% Bloat	A/vacuum	A/analyze	Vacuum	Analyze
gid_events	120617272	0	0	14:07:00	10:19:00	20:13:00	20:15:00
gid_gid_trains	1254149	87	0	23:39:00	23:54:00	20:18:00	20:18:00
gid_traindates	2932015	0	0	14:07:00	10:20:00	20:16:00	20:16:00
gid_trains	2932015	0	0	14:08:00	10:22:00	20:17:00	20:18:00
guid2id	28020097	0	0	14:08:00	11:00:00	20:16:00	20:17:00
keyvalues	815088	0	0	11:28:00	11:24:00	20:15:00	20:15:00
traincalendars	74085723	0	0	11:29:00	11:27:00	20:11:00	20:12:00

Каждое утро БД – как новенькая!

ВНЕЗАПНО

* И. Е. Репин, «Не ждали»

ВНИИЖТ
НЦ ЦМПЭ

Проблемы с Sec Scan. Опять

GID_Events
Train_EID
…

142 млн.точек

TMP_Trains
Train_EID
…

82000 поездов

GID_Events имеет индекс по Train_EID

```
SELECT E.*
FROM   GID_Events E INNER JOIN
       TMP_Trains T
       ON(T.Train_EID=E.Train_EID);
```

Запрос возвращает 816 779 строк

```
QUERY PLAN
Hash Semi Join  (cost=170.84..4792890.47 rows=11787245 width=286) (actual time=13254.652..135489.742
rows=816779 loops=1)
  Hash Cond: (e.train_eid = t.train_eid)
  ->  Seq Scan on gid_events e  (cost=0.00..4287162.24 rows=142637824 width=286) (actual
time=0.217..117560.619 rows=816779 loops=1)
  ->  Hash  (cost=124.82..124.82 rows=82427 width=8) (actual time=2.882..2.892 rows=82427 loops=1)
        Buckets: 4096  Batches: 1  Memory Usage: 176kB
        ->  Seq Scan on tmp_trains t  (cost=0.00..124.82 rows=82427 width=8) (actual time=0.036..2.109
rows=82427 loops=1)
Planning Time: 5.428 ms
Execution Time: 135523.087 ms
```

Запросы с меньшим числом поездов работают по индексу!

А вот теперь пришло время pg_hint_plan

- Устанавливается на ванильный PostgreSQL из исходников (на наш 11.17 установилось)
- Нужно добавить в `postgresql.conf` строку `shared_preload_libraries = 'pg_hint_plan'`
- В хранимых процедурах применим только в динамическом SQL `EXECUTE … USING …`
- Может быть установлен как EXTENSION; в этом случае позволяет влиять на запросы, которые нельзя переписать (например, в существующем приложении от стороннего разработчика)
- У нас почему-то заработало только для запросов в нижнем регистре

https://github.com/ossc-db/pg_hint_plan

Ура, заработало!

GID_Events
Train_EID
…

142 млн.точек

TMP_Trains
Train_EID
…

82000 поездов

ХИНТЫ

```
INSERT /*+
 MergeJoin(t e)
 SeqScan(t)
 IndexScan(e)
*/ INTO tmp_points
SELECT e.*
FROM tmp_trains t LEFT JOIN
      gid_events e
ON(t.train_eid=e.train_eid)
WHERE (e.dt<=$1)AND(e.dt>=$2)
```

```
SELECT E.*
FROM   GID_Events E INNER JOIN
       TMP_Trains T
       ON(T.Train_EID=E.Train_EID);
```

```
QUERY PLAN
Insert on tmp_points (cost=10000423817.69..10000681142.18 rows=4079701 width=286)
                     (actual time=1607.286..1607.288 rows=0 loops=1)
  ->  Merge Join  (cost=10000423817.69..10000681142.18 rows=4079701 width=286)
                  (actual time=579.042..1141.264 rows=816779 loops=1)
        Merge Cond: (e.train_eid = t.train_eid)
        ->  Index Scan using gid_events_pk on gid_events e (cost=0.56..4247967.02 rows=4187153 width=286)
                           (actual time=0.022..663.640 rows=3172841 loops=1)
              Index Cond: ((dt <= '2023-01-24 00:22') AND (dt >= '2022-12-07 20:00'))
        ->  Sort  (cost=10000002606.54 rows=82427 width=8) (actual time=6.679..46.061 rows=816779 loops=1)
              Sort Key: t.train_eid
              Sort Method: quicksort  Memory: 1918kB
              ->  Seq Scan on tmp_trains t (cost=10000000000.00..10000000818.27 rows=82427 width=8)
                                           (actual time=0.005..2.487 rows=82427 loops=1)
Planning Time: 0.349 ms
Execution Time: 1607.316 ms
```

Стало быстрее в десятки раз!

Новая надежда!

- Пётр Петров, PostgresPro, указал путь к решению проблемы с Sec Scan !
- В проблемном запросе планировщик очень сильно ошибается в оценке стоимости, поэтому отказывается от индекса в пользу Sec Scan
- Возможно, причина кроется в особенностях алгоритма выборки для оценки (функция `acquire_sample_rows`, файл `analyze.c`, алгоритм J. S. Vitter, Random Sampling with a Reservoir)
- Это проявляется на больших таблицах (у нас – 150 млн строк)
- Путей решения два: или разбираться с алгоритмом, или партицировать таблицу!

Будет что рассказать в следующий раз!

* «Новая надежда» по мнению нейросети

Бонус: сохраняем план из-под PL/pgSQL


```
FOR v_Plan_Line IN
   EXPLAIN ANALYZE
      INSERT INTO TMP_Points
      SELECT E.*
      FROM   TMP_Trains T LEFT OUTER JOIN GID_Events E
                          ON(T.Train_EID=E.Train_EID)
      WHERE (E.DT <= v_LastDate )
         AND(E.DT >= v_FirstDate)
LOOP
     v_Plan := v_Plan || v_Plan_Line || CHR(10);
END LOOP;
```

- Пока не ясно, как синтаксически оформить это в динамический SQL с хинтами внутри
- С интересом смотрим в сторону pg_store_plans от тех же авторов, что и pg_hint_plan

Выводы:

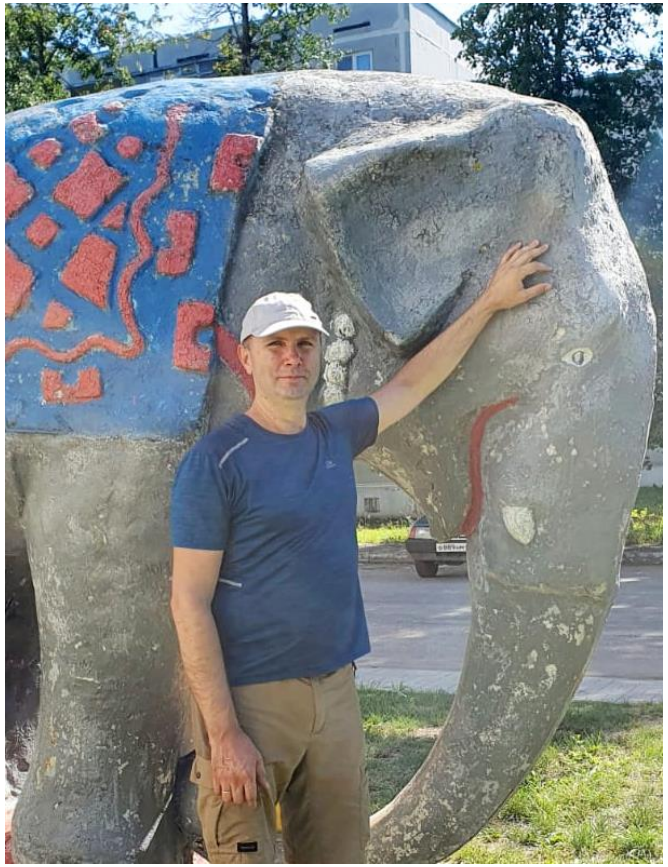
«Многие вещи нам непонятны не потому, что наши понятия слабы; но потому, что сии вещи не входят в круг наших понятий»

Козьма Прутков

«Это невозможно понять, это можно только запомнить»

Народ

О докладчике



anfinogenov.anatoly@vniizht.ru
+7 (499) 262-45-06

АО «ВНИИЖТ» (АО «Научно-исследовательский институт железнодорожного транспорта»)

Анфиногенов Анатолий Юрьевич

Заместитель директора научного центра – начальник отдела разработки ПО, кандидат физико-математических наук

Работаю уже два десятка лет во ВНИИЖТ над задачами имитационного моделирования и оптимизации железнодорожных перевозок.

Проектировал, разрабатывал и сопровождал БД и серверное ПО для этих задач (PostgreSQL, Oracle, C++, Python), чем и продолжаю заниматься.

Понравился доклад– голосуйте по ссылке:

https://pgconf.ru/2023/346811