IT's TINKOFF

Обеспечение качественных ETL на Vertica

Технологические и организационные аспекты

Крашенинников Александр

Александр Крашенинников

Руководитель отдела платформы загрузки данных

a.krasheninnikov

Disclaimer

Описанный пользовательский сценарий реализовывался
не в рамках компании Tinkoff

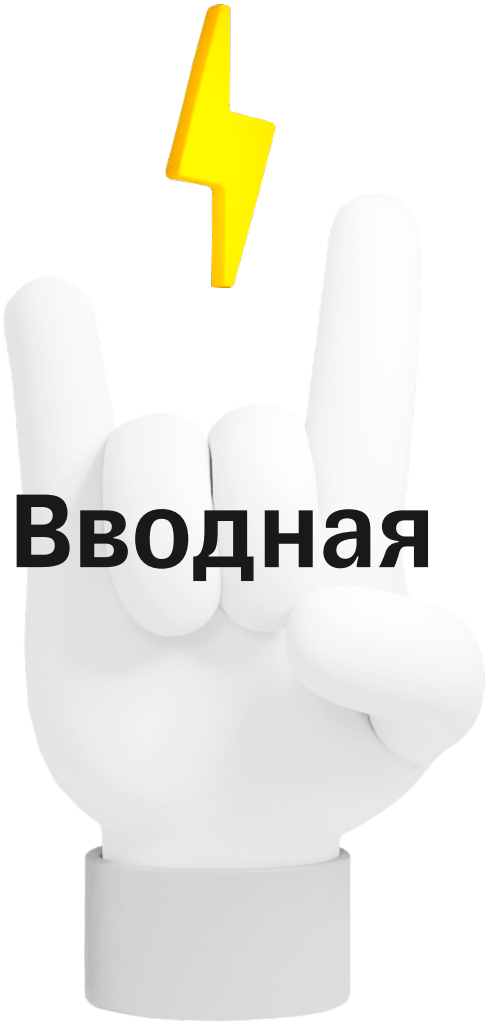
Компания, в которой происходили события из доклада, останется неназванной из соображений NDA

О чем доклад

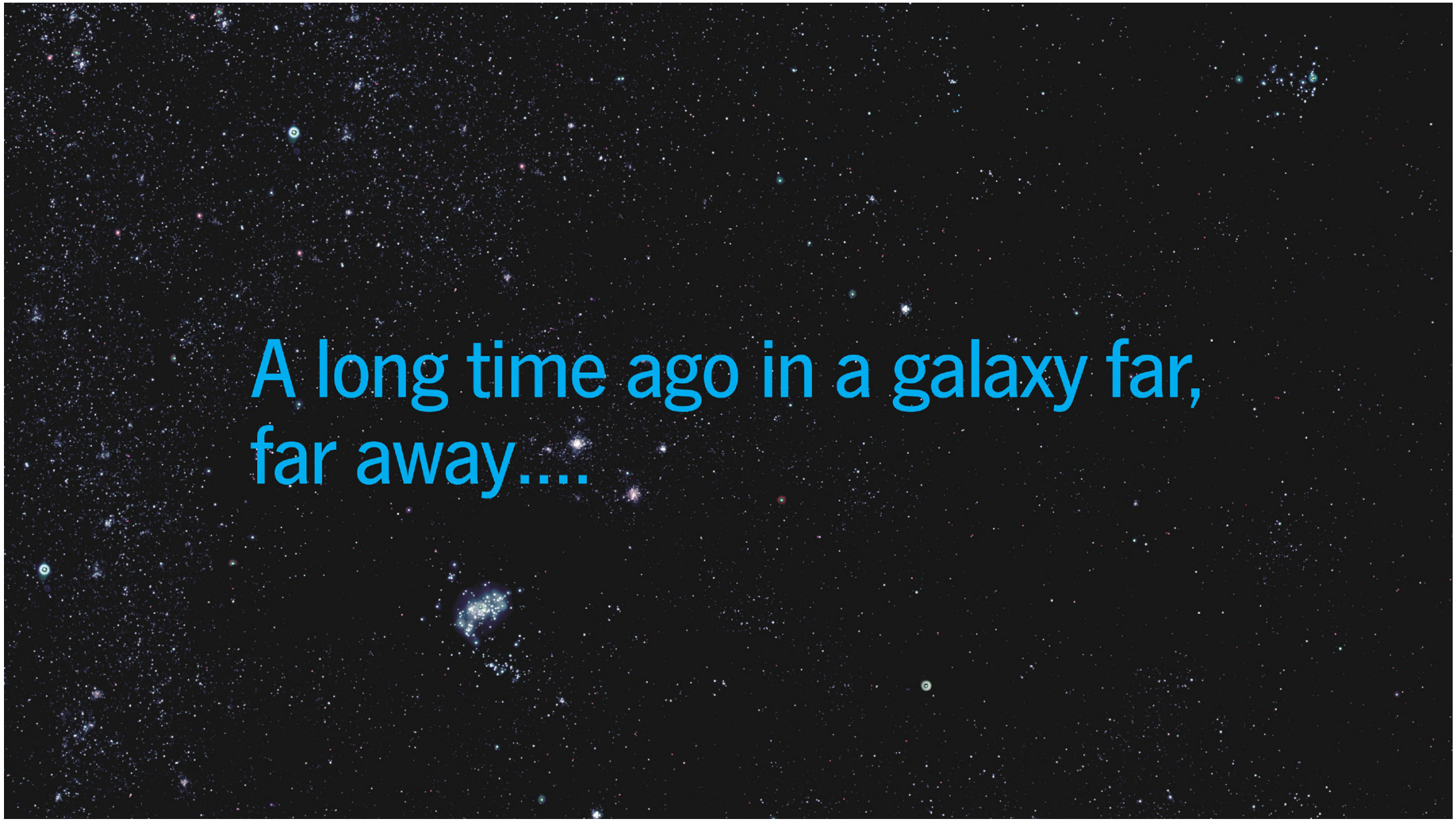
«Условия задачи»

Эволюция подходов к клиентскому счастью

Выводы



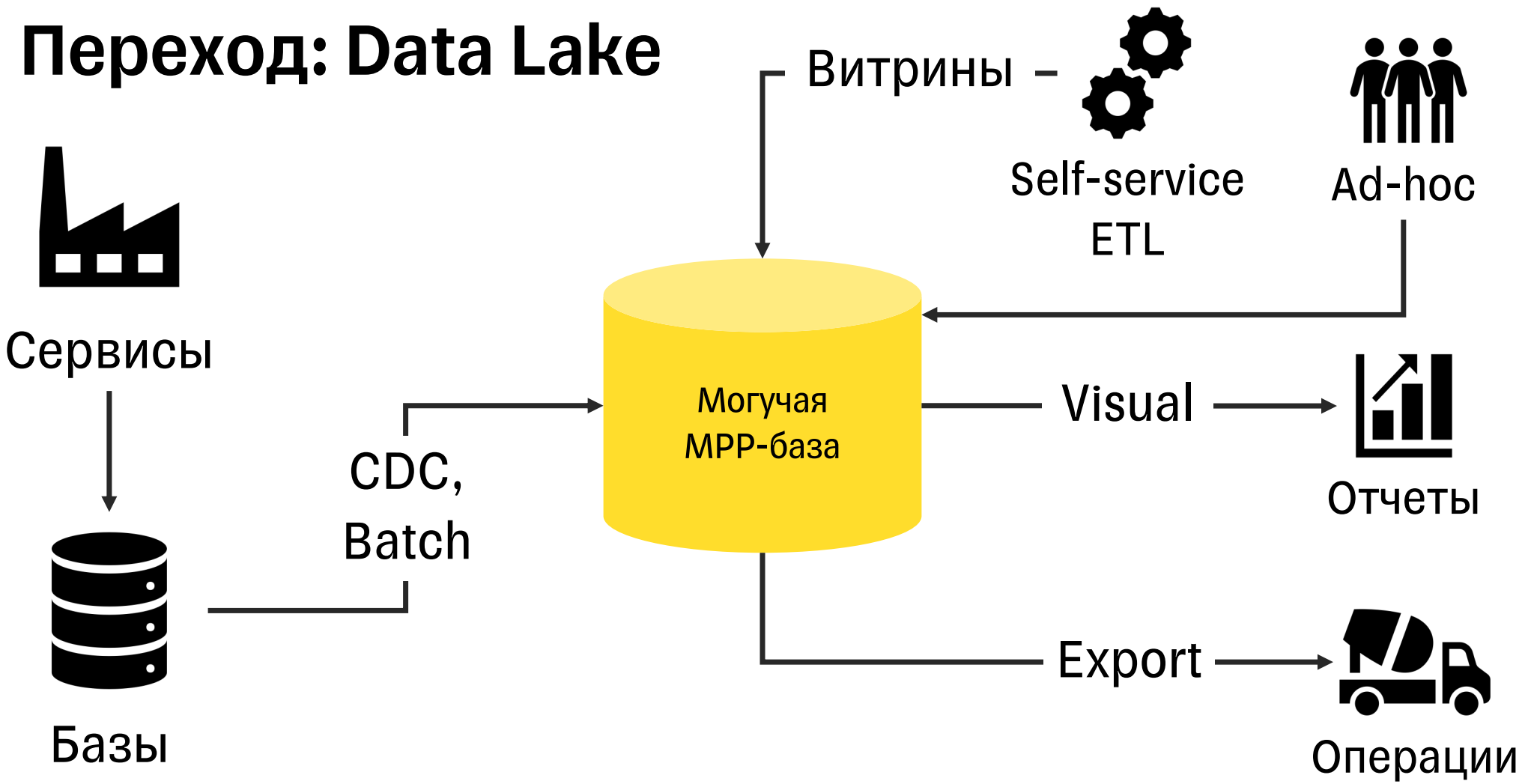
Вводная


A long time ago in a galaxy far,
far away....

Ситуация

1. Есть организация
2. Интенсивный рост оборотов и бизнеса в целом
3. Большой спрос на аналитику (500+ аналитиков)
4. Технологии BI не успели за ростом
5. В качестве «подорожника» – воткнули MPP-базу, и отправили всех с ней работать

Переход: Data Lake



Mighty MPP: Vertica

1. Классическая MPP (multi-node, column store)
2. ACID-совместимость
3. Нет SPOF («узел-координатор»)
4. Быстрая
5. Щедрая на системную информацию
6. Проприетарная

Особенности эксплуатации

1. Инженеры данных привозят «сырьё»
2. Пользователи сами пишут ETL и выводят его в prod
3. Всё работает

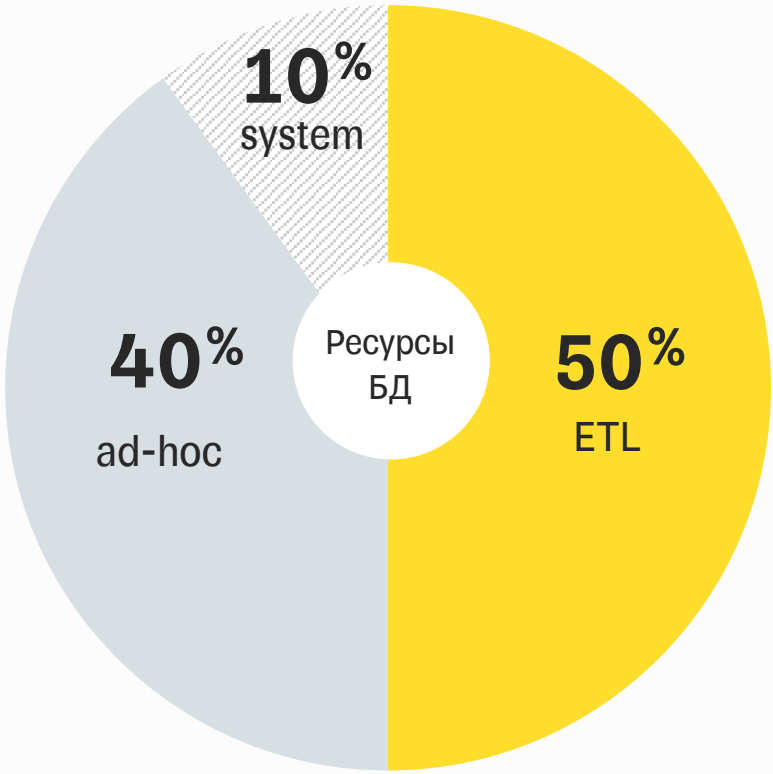
Эволюция подходов: Watchdog

Ситуация: «база тормозит»

1. Запущен «жирный» запрос
2. Потреблены все ресурсы БД
3. У потребителей вырастает время ответа
4. Пользовательский опыт хромает
5. Бизнес-процессы замедляются

Решение: квотирование

1. Самый дешевый способ
2. Решает 70-80% сценариев
3. Ограничивает
   - Общее время выполнения
   - CPU
   - RAM
   - Concurrency
   - Объем временных данных

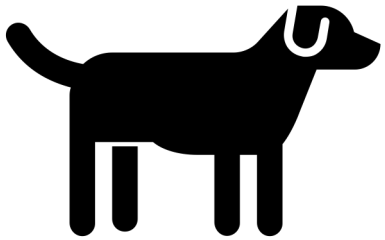


Что не решает квотирование

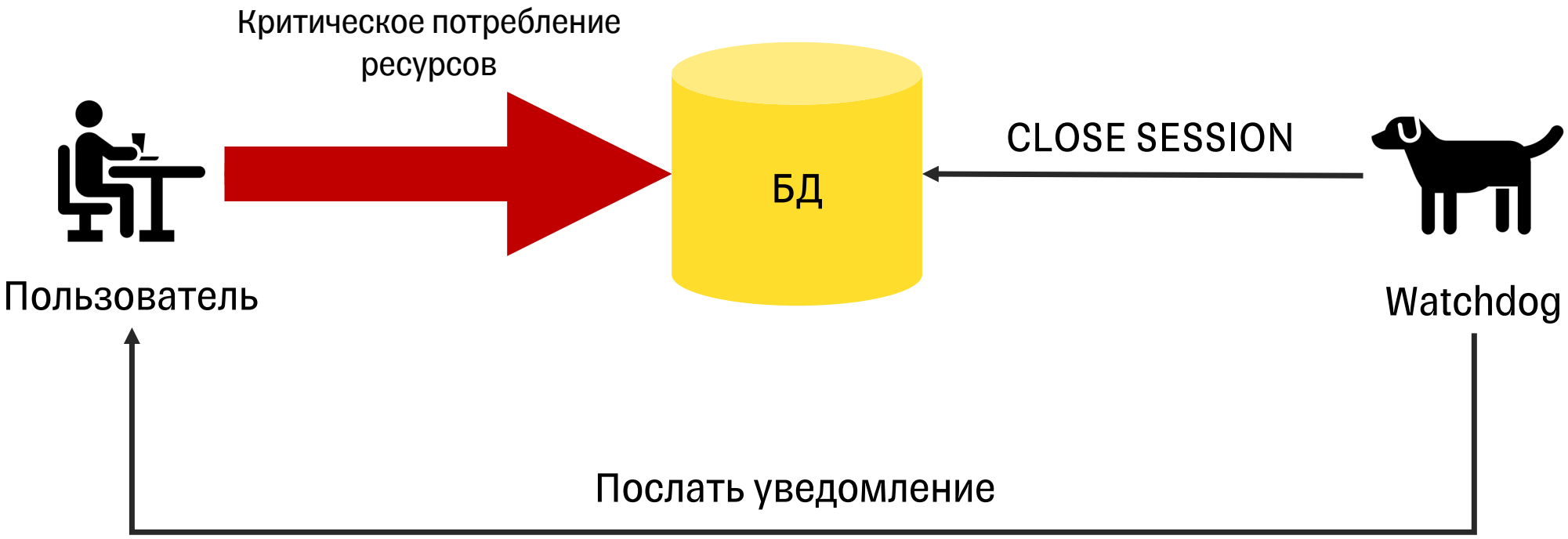
1. Объем чтения с диска
2. Объем передачи по сети
3. Перекос нагрузки
4. «Нецелевое» использование БД

Решение: query watchdog

1. Сторонний сервис
2. Набор threshold + эвристики
3. Запуск мониторинговых запросов
4. Отстрел пользовательских запросов



Watchdog


Критическое потребление ресурсов
БД
CLOSE SESSION
Пользователь
Watchdog
Послать уведомление

Уведомление об отстреле

Close session v_vertica_node0015-XXXXXX (временные данные)
Пользователь: (some_system_user) <@Василий Петров, @Федор Никитин>

Transaction+statement:	1234567-987654321	
Длительность:	10m 54s (recom: 10m,	лимит: 15 m)
Память:	167 GB (recom: 128 GB,	лимит: 256 GB)
Сеть:	12 GB (recom: 10GB,	лимит: 100 GB)
Временные данные:	**1.2 TB** (recom: 500GB,	лимит: 1.0 TB)


Скачать текст запроса

Что случилось?

Close session v_vertica_node0015-XXXXXX (временные данные)

Пользователь: (some_system_user) <@Василий Петров, @Федор Никитин>

Transaction+statement: 1234567-987654321
Длительность: 10m 54s (recom: 10m, лимит: 15 m)
Память: 167 GB (recom: 128 GB, лимит: 256 GB)
Сеть: 12 GB (recom: 10GB, лимит: 100 GB)
Временные данные: **1.2 TB** (recom: 500GB, лимит: 1.0 TB)

Скачать текст запроса

Кому это важно?

Close session v_vertica_node0015-XXXXXX (временные данные)
Пользователь: (some_system_user) <@Василий Петров, @Федор Никитин>

Transaction+statement: 1234567-987654321
Длительность: 10m 54s (recom: 10m, лимит: 15 m)
Память: 167 GB (recom: 128 GB, лимит: 256 GB)
Сеть: 12 GB (recom: 10GB, лимит: 100 GB)
Временные данные: 1.2 TB (recom: 500GB, лимит: 1.0 TB)

Скачать текст запроса

Where to go from here?

Close session v_vertica_node0015-XXXXXX (временные данные)
Пользователь: (some_system_user) <@Василий Петров, @Федор Никитин>

Transaction+statement: 1234567-987654321
Длительность: 10m 54s (recom: 10m, лимит: 15 m)
Память: 167 GB (recom: 128 GB, лимит: 256 GB)
Сеть: 12 GB (recom: 10GB, лимит: 100 GB)
Временные данные: 1.2 TB (recom: 500GB, лимит: 1.0 TB)

Скачать текст запроса

Как предотвратить?

Close session v_vertica_node0015-XXXXXX (временные данные)
Пользователь: (some_system_user) <@Василий Петров, @Федор Никитин>

Transaction+statement: 1234567-987654321
Длительность: 10m 54s (recom: 10m, лимит: 15 m)
Память: 167 GB (recom: 128 GB, лимит: 256 GB)
Сеть: 12 GB (recom: 10GB, лимит: 100 GB)
Временные данные: **1.2 TB** (recom: 500GB, лимит: 1.0 TB)

Скачать текст запроса

Неоптимальное использование БД

1. OLTP-style нагрузка

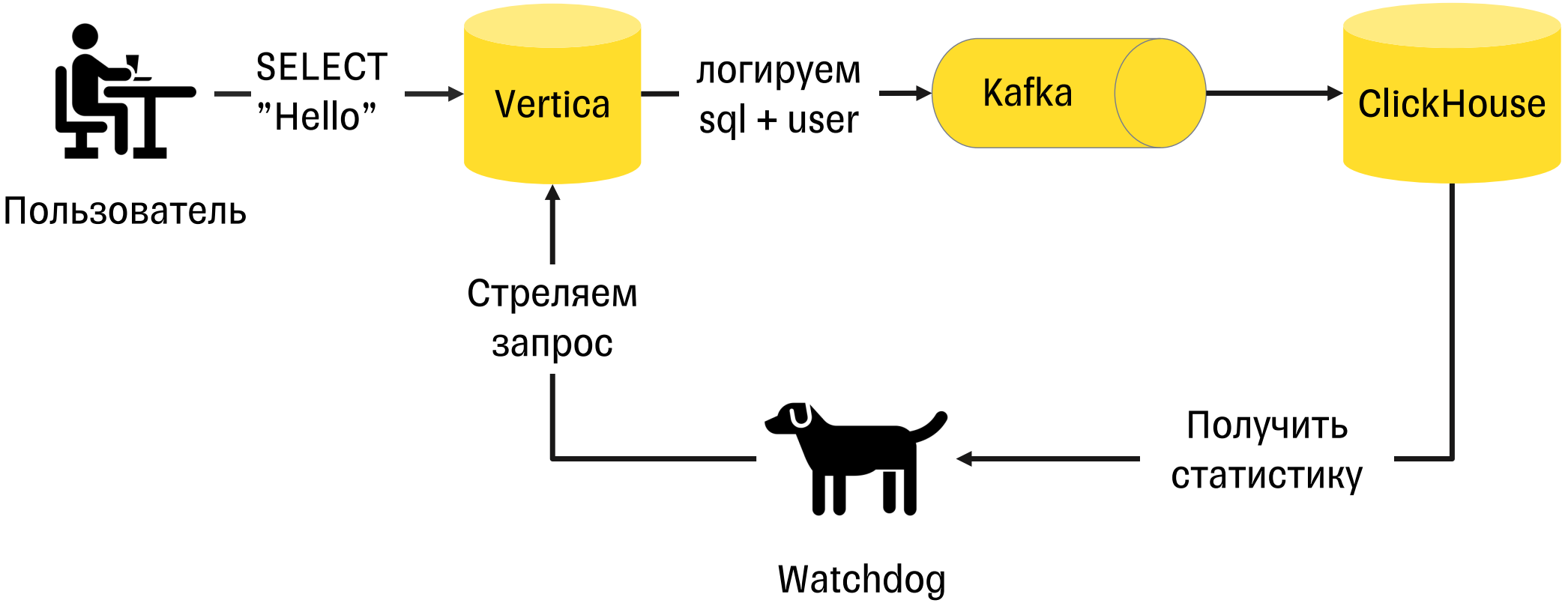
```
SELECT * FROM table WHERE id IN (/* тысячи их*/)
```

2. Однопоточная интеграция данных

OLTP-style нагрузка

1. Профиль Vertica – «нечастые запросы на больших массивах данных»
2. Плохо – «частые запросы на больших массивах данных»
3. Забивание очереди запросов
4. Деградация «целевых» запросов
5. Надо найти «повторяемые»
6. Предотвратить выполнение

Выявление дублей запросов



Почему Kafka + ClickHouse?

1. Штатная интеграция Vertica+Kafka для системной информации
2. ClickHouse: long-term хранение статистики
3. ClickHouse: low-latency доступ к истории
4. ClickHouse: выделение fingerprint запроса

Логи запросов в ClickHouse

sql	user	normarlizeQueryHash(sql)	time
SELECT 123	igor	12345	...
SELECT 456	igor	12345	...

```
SELECT
    user,
    normalizeQueryHash(sql) AS hash,
    count(1) AS cnt
FROM vertica_queries
WHERE time > now() – INTERVAL ‘15’ minute
GROUP BY user, hash
```

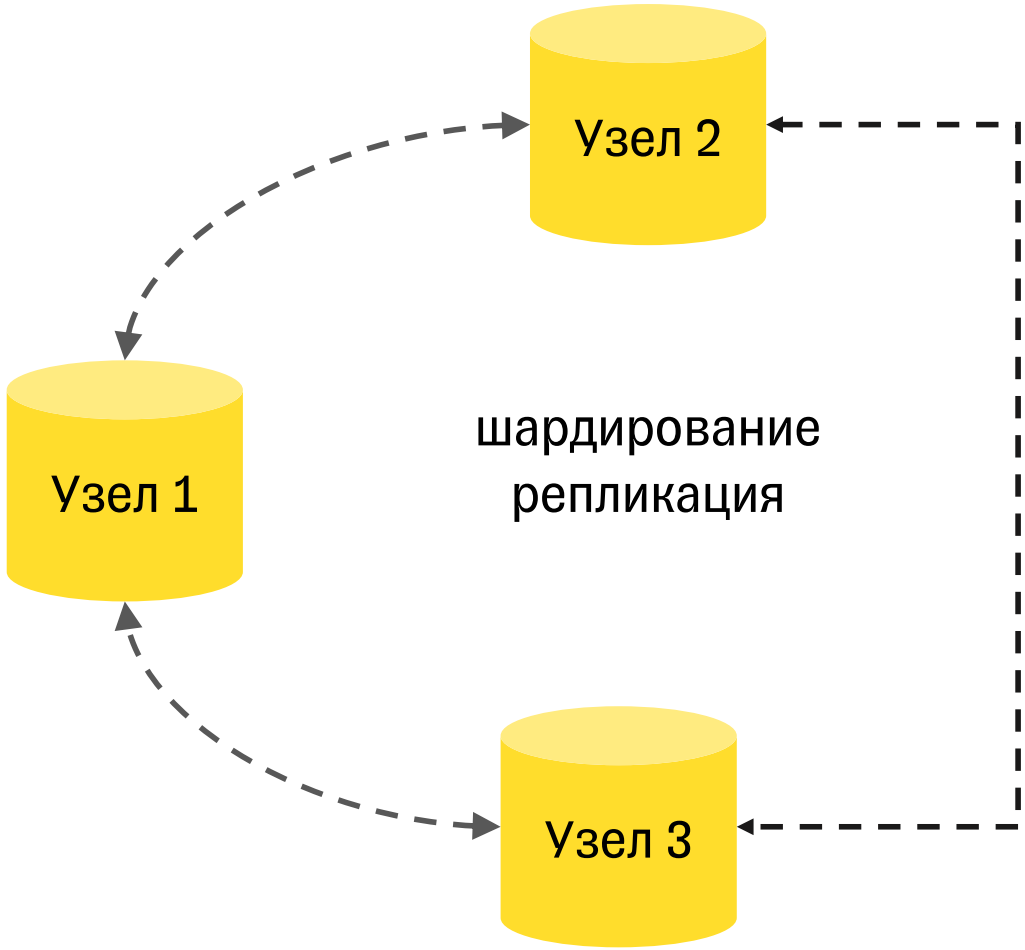
Неоптимальное использование БД

1. OLTP-style нагрузка

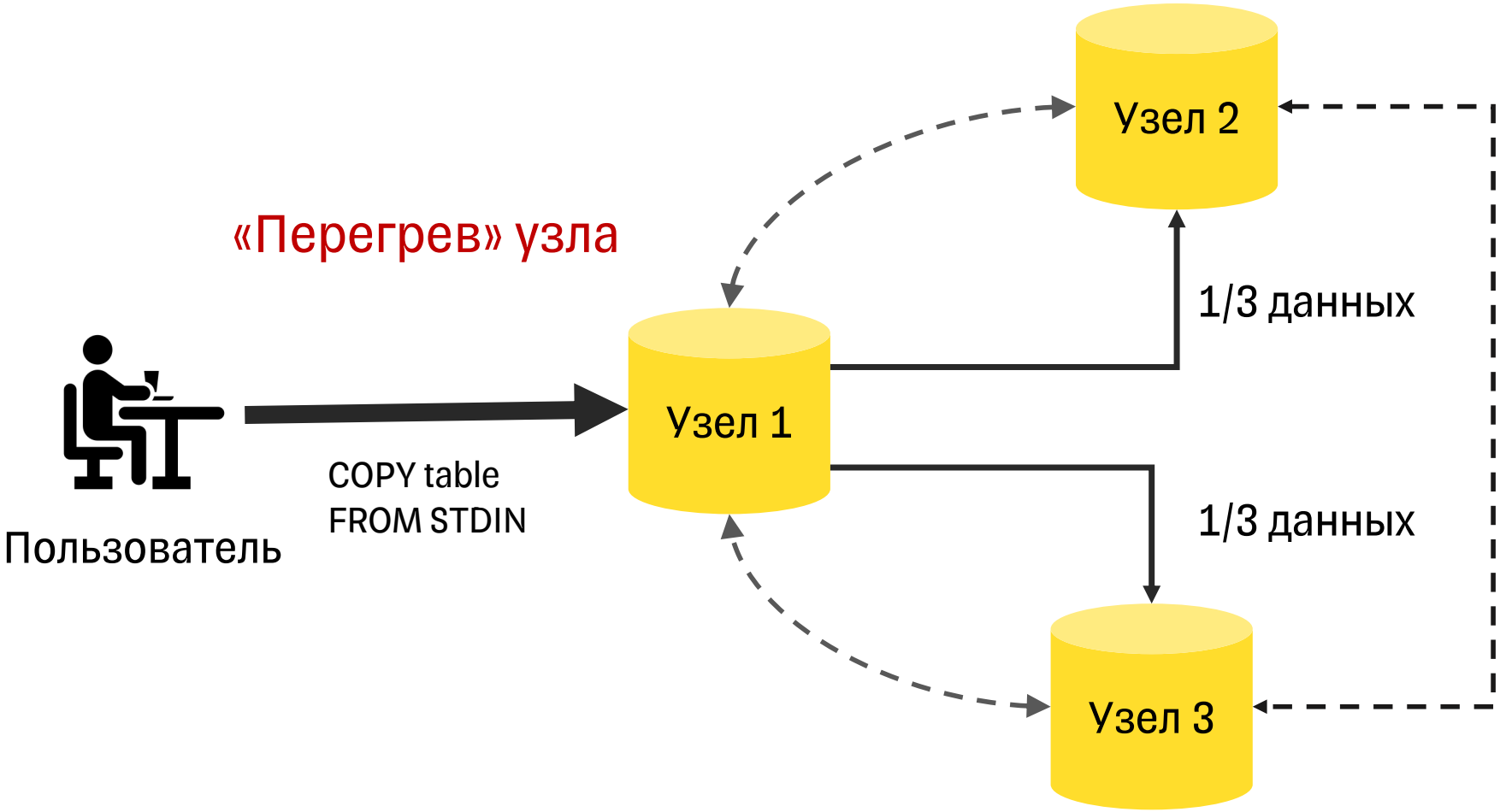
```
SELECT * FROM table WHERE id IN (/* тысячи их*/)
```

2. Однопоточная интеграция данных

Организация данных в Vertica



Однопоточная интеграция данных



Перегрев узла

1. Парсинг всей входящей нагрузки
2. Повышенное потребление ресурсов
3. Деградация всех остальных запросов
4. Где встречается: Vertica, GreenPlum, ClickHouse

Уведомления

Обнаружена частая/долгая вставка в БД от пользователя

Частота:	1 запрос / 5 m	
Длительность:	10m 45s	(recom: 5 m, limit: 15 m)
Объём данных:	**21 GB**	(recom: 2GB, limit: 20GB)

Обнаружена долгая выгрузка данных из БД

Длительность:	5m 45s	(recom: 5 m, limit: 15 m)
Строк передано:	112,456	(**325** строк в секунду)

Обнаружен повторяющийся запрос

Частота:	30 запросов / 5 минут

Приобретенный опыт

1. Квоты-first подход
2. Требуются дополнительные защитные средства
3. Надо регламентировать границы дозволенного у БД (сценарии использования)

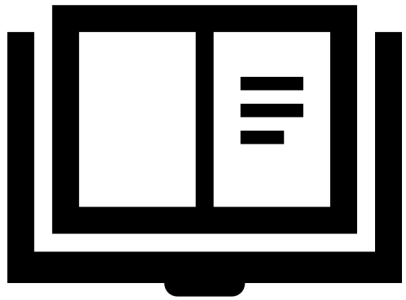
Эволюция подходов: обучение

Ситуация

1. Проведены воркшопы по MPP
2. Написана документация, best practices, правила пользования
3. New joiners не читают документацию
4. Качество производимого кода падает
5. БД деградирует
6. Потребители негодуют

Решение: обучающий курс

1. Структурированная подача информации
2. Тесты для самопроверки
3. Аттестация (итоговый курс)
4. Блокировка личной УЗ при непрохождении



Что включить в курс

1. Вводная про MPP
2. Ключевые отличия от single-node баз
3. Для чего используется в организации
4. Как подключиться
5. Базовый query language
6. Хранение: шардирование, data locality, сортировка
7. Оптимизации: типы JOIN, GROUP BY
8. Оптимизации: неточный distinct, argmax

Пример подачи

```
SELECT a.id, count(b.id)
FROM
  schema.a
INNER JOIN schema.b USING (id)
GROUP BY a.id
```

Какой тип JOIN будет использоваться в данном запросе?

- HASH
- MERGE

Пример подачи

Как удалить 80% данных из таблицы 1B строк и 500GB объёмом?

- ☐ Партицировать таблицу + сделать DROP PARTITION по диапазону
- ☐ Выполнить DELETE FROM table
- ☐ Создать временную таблицу, вставить 20% записей и затем произвести SWAP

Санкционные меры

Привет!
Мы обнаружили что у тебя есть роль «Доступ к Vertica» - она требует пройти курс «Основы работы с БД».
Если не пройти курс до **18 октября 2023**, твоя учетная запись будет заблокирована.

Все вопросы можно задать в канале ~vertica-query-performance

Nice to have

1. Множественные варианты ответов
2. Пояснения «почему этот ответ неправильный»
3. Обратная связь по курсу («что можно улучшить»)
4. Актуализация курса новым материалом
5. «Песочница» для запуска на рабочей машине
6. Площадка для обсуждения («канал в чате для community»)

Приобретенный опыт

1. Нужен процесс единообразного онбординга пользователей
2. «Добрым словом и пистолетом...» - обязательная аттестация
3. Наличие community повышает уровень инженерной грамотности

Эволюция подходов: анализ профиля нагрузки

Ситуация

1. Пользователи обучены
2. Запросы стреляются
3. Пользовательский опыт хромает
4. Нужен deep dive в то, что происходит

Открытые вопросы

1. Что является нормальным?
2. Справедливо ли распределение ресурсов?
3. Может пора залить проблему железом?
4. А при добавлении железа все ли запросы отмасштабируются?
5. Может нас спасёт чатгпт?

Аналитика артефактов аналитики

1. Собрать телеметрию запросов
2. Сгруппировать по пользователям («учетная запись», «ресурсная группа»)
3. Посмотреть выбросы
4. Проанализировать корреляции
5. Кластеризация?
6. Anomaly detection?
7. Начнем с простого – OLAP-кубик ☺

Потребление ресурсов | Статические дефекты | Статические дефекты (детали)

Настройки

Дата

2023-05-18

Разбивка по

Пользователю

Ось X

Время суток (час)

логарифм

Ось Y

(Длительность) sum, часов

логарифм

Ось Z

(CPU) sum, часов

логарифм

Размер

(Диск, чтение) sum, Gb

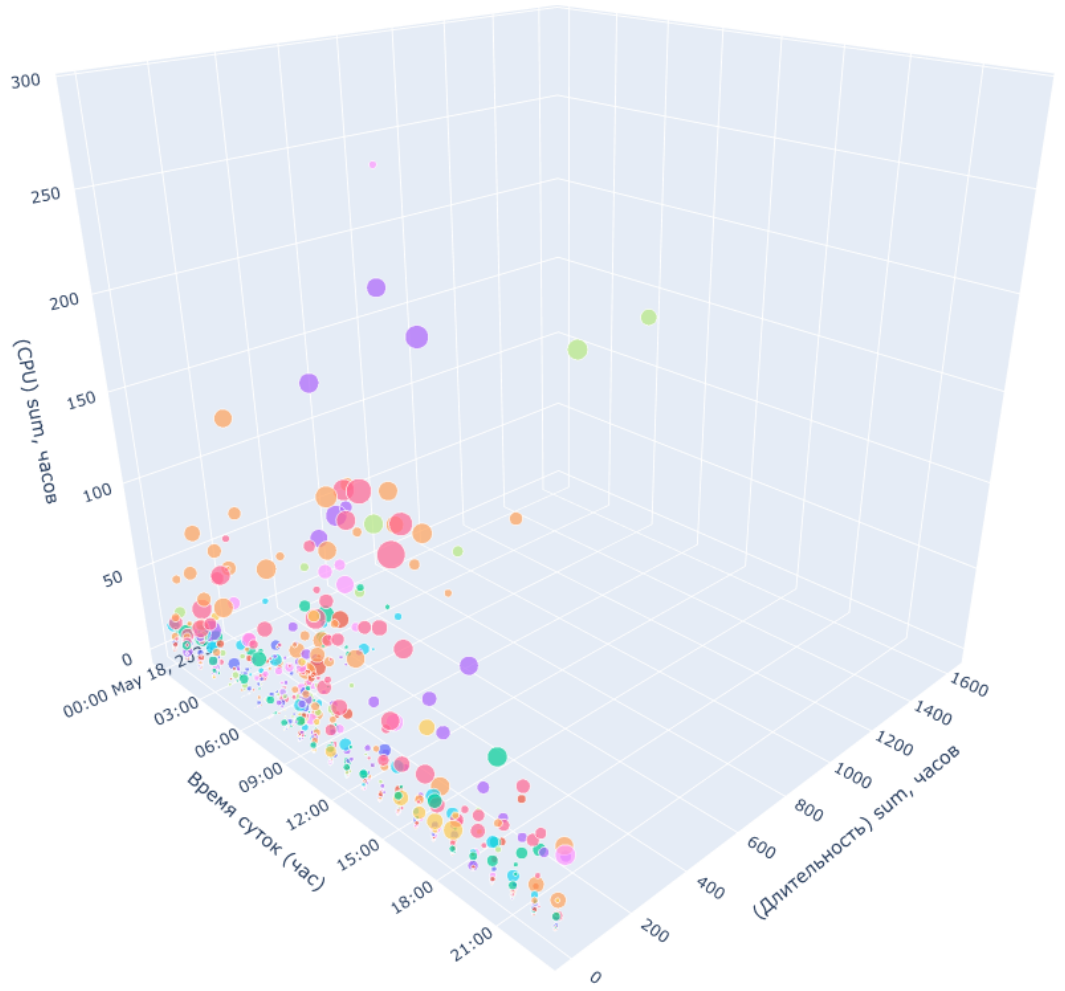
Фильтры

User in

User not in

services

Pool not in



Потребление ресурсов | Статические дефекты | Статические дефекты (детали)

Настройки

Дата

2023-05-18

Разбивка по

Пользователю

Ось X

Время суток (час)

логарифм

Ось Y

(Длительность) sum, часов

логарифм

Ось Z

(CPU) sum, часов

логарифм

Размер

(Диск, чтение) sum, Gb

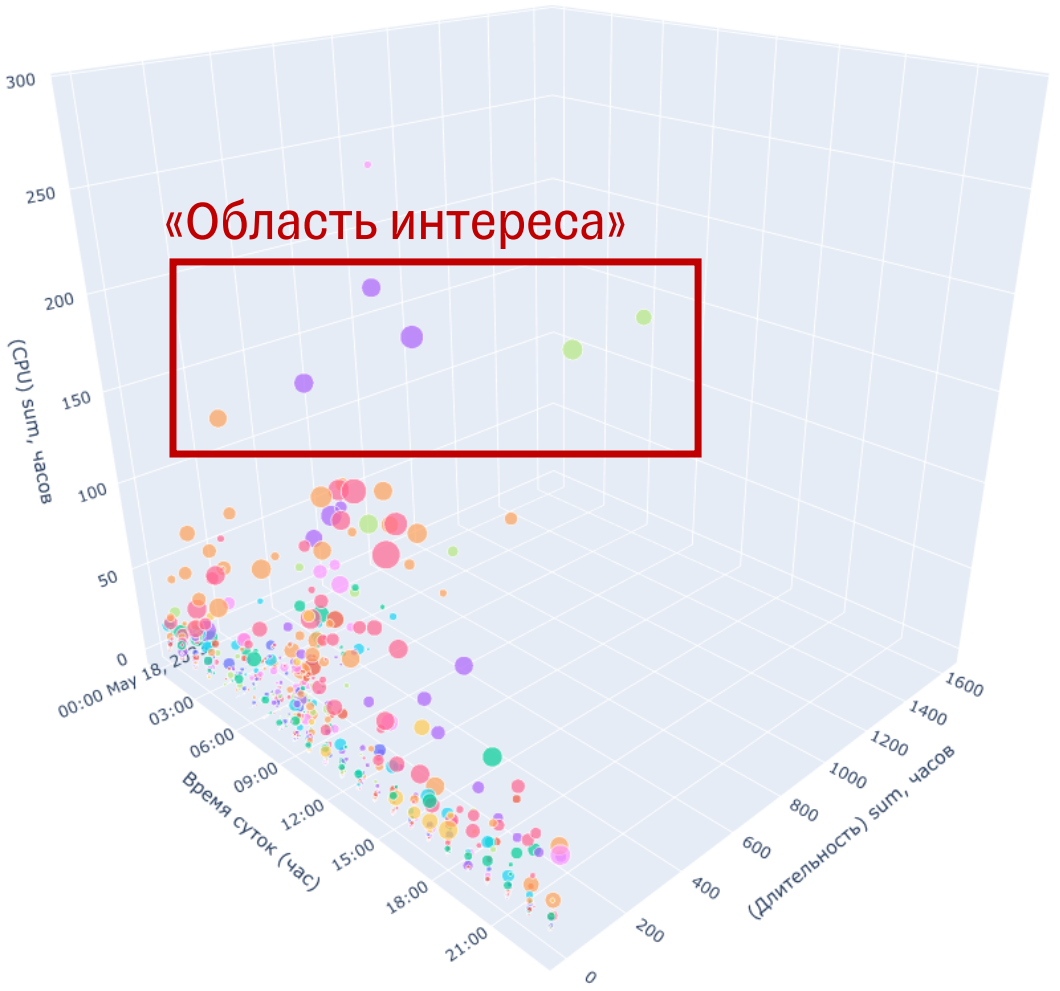
Фильтры

User in

User not in

services

Pool not in



Компоненты куба

1. Измерения: время, УЗ, ресурсная группа, узел кластера
2. Показатели: sum/skew/avg/q50/q75/q90/q95/q99 по метрикам запросов
3. Время: CPU, общее время
4. RAM
5. Объемы данных: чтение/запись по диску, чтение/запись по сети
6. Число запросов
7. COST

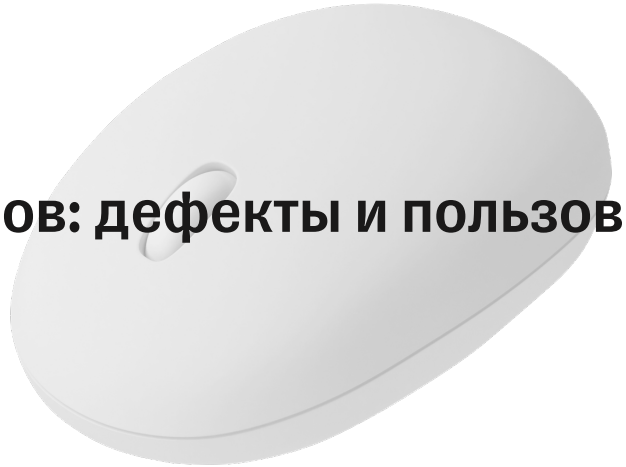
Процесс

1. Эксперт отсматривает куб/сводник по телеметрии
2. Находит outlier’ов
3. Коммуницирует с владельцами кода
4. Выставляет задачу-блокер на исправление (20 дней на исправление)
5. Проверяет факт исправления
6. Производит блокировку УЗ при отсутствии реакции

Приобретенный опыт

1. Лимиты на отдельный запрос не защищают от «неудобной» нагрузки на БД в целом
2. Нужно создавать прозрачность по профилю запросов – отчеты, Grafana
3. Важно таргетировано доносить до пользователей информацию о проблемах в их процессах

Эволюция подходов: дефекты и пользовательский рейтинг



Ситуация

1. Эксперты не успевают выставлять задачи на исправление
2. В ходе работы выявили схожие паттерны ошибок
3. Систематизируем и учимся выявлять автоматически

Дефекты

Ошибка проектирования структур данных или процессов их обработки

Статические

Когда эти данные будут обрабатывать – станет больно

- Таблицы
- Колонки

Динамические

Это уже сейчас работает неоптимальным образом

- Запросы
- Последовательности запросов

Статические дефекты 🏢

1. Неоптимальные типы колонок (text/decimal)

Работа со строками в Vertica

```
CREATE TABLE sample (name VARCHAR(2048))
```

1. Пессимистичный оптимизатор – завышаем COST
2. В user-defined функциях (java) происходит аллокация 2Kb на каждую строку
3. Подход к «правильному» выставлению длины поля:

```
length = max(octet_length(value)) x safety_factor
```

Ах этот numeric....

1. Случайно/по ошибке создаются таблицы с numeric разной длины
2. Больше точность (precision) => больше байт на диске и в RAM
3. Несовпадение точности => приведение типов в runtime, CPU/RAM потребление
4. numeric(18,4) – 90% сценариев работы с деньгами
5. numeric(18,X) эквивалентен в стоимости integer (int64)
6. «Чемпион» - numeric(128, 34)

Дефекты: типы данных

1. Размер VARCHAR/VARBINARY более XXX символов

   Эта БД не совсем про хранение и обработку текстов

2. Встречается NUMERIC с precision более 64

   В компании отсутствует потребность в таких вычислениях

3. В колонках–идентификаторах используется numeric(XXX,0) вместо INT

   Потенциально теряем возможность MERGE JOIN

Статические дефекты 🏢

1. Неоптимальные типы колонок (text/decimal)

2. Сортировка данных

Сортировка данных

- Указываем поля, по которым сортировать данные при вставке
- Вся вставка должна быть пересортирована по указанным полям перед записью на диск

```
CREATE TABLE sample (
    id INT,
    name VARCHAR(256),
    surname VARCHAR(256)
)
ORDER BY name, surname
```

Сортируем по 512 байтам
вместо 8 байт для INTEGER

Дефекты: сортировка

1. Более 4 полей в ключе сортировке

   Ошибка в DDL или не понимание для чего нужна сортировка

2. Суммарная длина ключа более 2KB

   Ошибочно выбранные поля

3. Rule of thumb – «выбирать минимальные типы данных»

4. Можно встретить в: Vertica, ClickHouse

Статические дефекты 🏢

1. Неоптимальные типы колонок (text/decimal)
2. Сортировка данных
3. Распределение данных

Распределение данных

- Указываем поля, на основании которых будет распределение при вставке
- node = ключ дистрибуции % число узлов (упрощенно)

```
CREATE TABLE sample (
    id INT,
    name VARCHAR(256),
    surname VARCHAR(256)
)
SEGMENTED BY hash(name, surname)
ALL NODES
```

Хеш от 512 байт вместо 8 байт для INTEGER

Несегментированные таблицы

- Таблицы, имеющие копию на каждом узле
- Предназначены для «небольших» справочных значений в JOIN

```
CREATE TABLE sample (
    id INT,
    name VARCHAR(256),
    surname VARCHAR(256)
)
UNSEGMENTED ALL NODES
```

Дефекты: распределение данных

1. Более 4 полей в ключе сегментации

   Ошибка в DDL или не понимание для чего нужна сегментация

2. Суммарная длина ключа сегментации более 2KB

   Ошибочно выбранные поля

3. В сегментации используются поля FLOAT

   Есть риск что это около-случайное распределение

4. В таблице менее 1M строк и она сегментирована

   Во время SELECT будет интенсивный обмен данными по сети

Дефекты: распределение данных

1. Несегментированная таблица большого размера (XXX строк, YYYGB)

   Накладные расходы на обслуживание таблицы дороже бенефитов SELECT

Статические дефекты 🏢

1. Неоптимальные типы колонок (text/decimal)
2. Сортировка данных
3. Распределение данных
4. Партицирование данных

Партицирование данных

- Разбиение таблицы на меньшие кусочки
- Уменьшение размера обслуживаемых и выбираемых данных

```
CREATE TABLE sample (
    id INT,
    name VARCHAR(256),
    birthdate DATE
)
PARTITION BY birthdate
```

Партицирование

1. Оптимизирует RANGE/IN предикат

   ```
   partition_field IN (…), partition_field BETWEEN XXXX
   AND YYYY
   ```

2. Число партиций желательно иметь в пределах нескольких тысяч

Дефекты: партицирование

1. Таблица содержит более 1500 партиций

   Высокая стоимость обслуживания, много места в системном каталоге

2. Большая непартицированная таблица (> XXX строк или > YYY GB)

   Большие расходы на обслуживание и сканирование таблицы

3. Партицированная таблица малого размера (< XXX строк или < YYY GB)

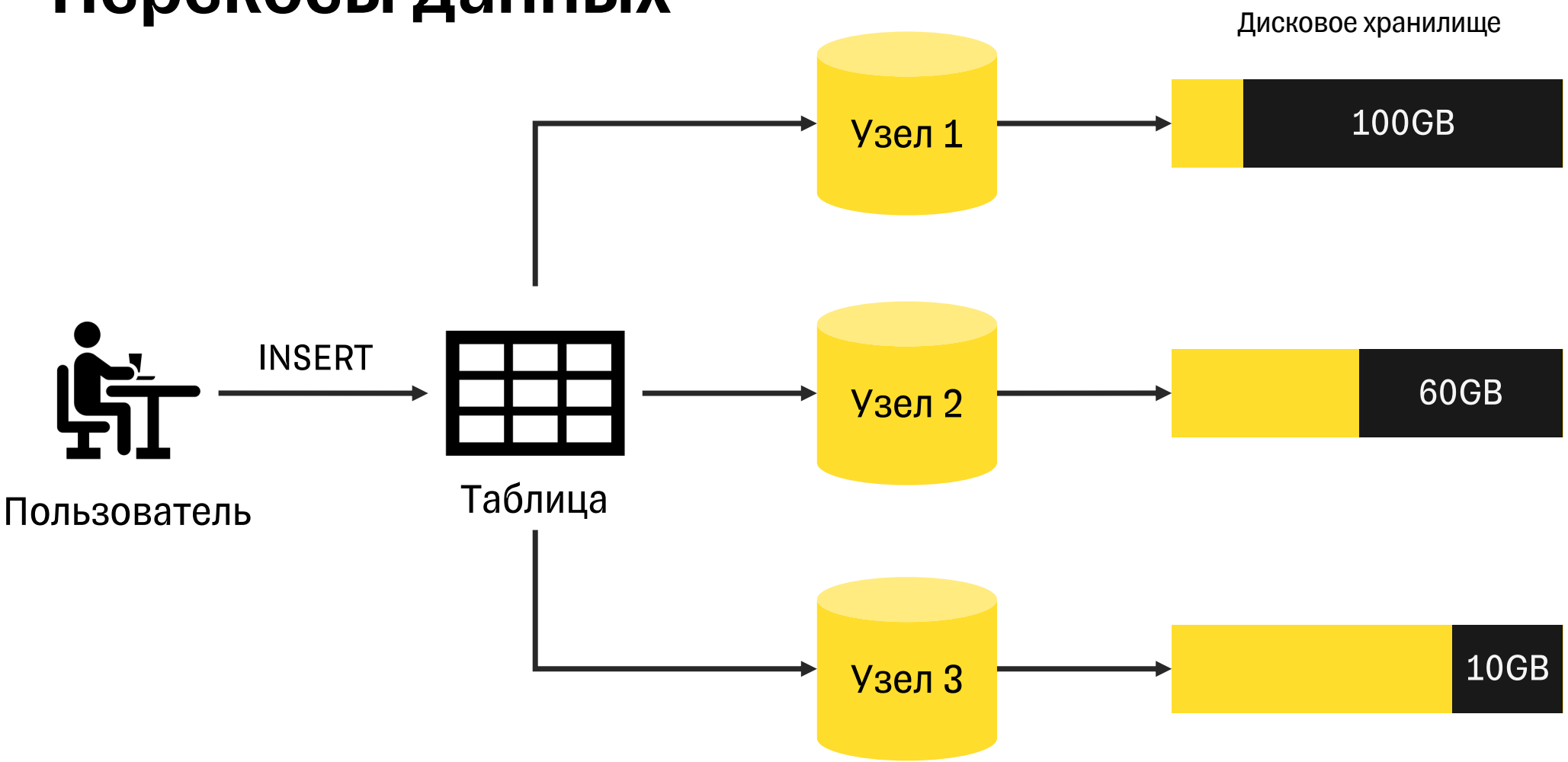
   Дешевле сделать FULLSCAN чем обслуживать партиции

4. Также встречается в: Vertica, GreenPlum, ClickHouse

Статические дефекты 

1. Неоптимальные типы колонок (text/decimal)
2. Сортировка данных
3. Распределение данных
4. Партицирование данных
5. Перекосы данных

Перекосы данных



Перекосы данных

1. Некоторые узлы «заканчиваются» раньше
2. Время запроса == время выполнения на самом заполненном узле

Дефекты: перекос данных

1. Данные перекошены по числу строк

   Перекошенные узлы тормозят весь запрос

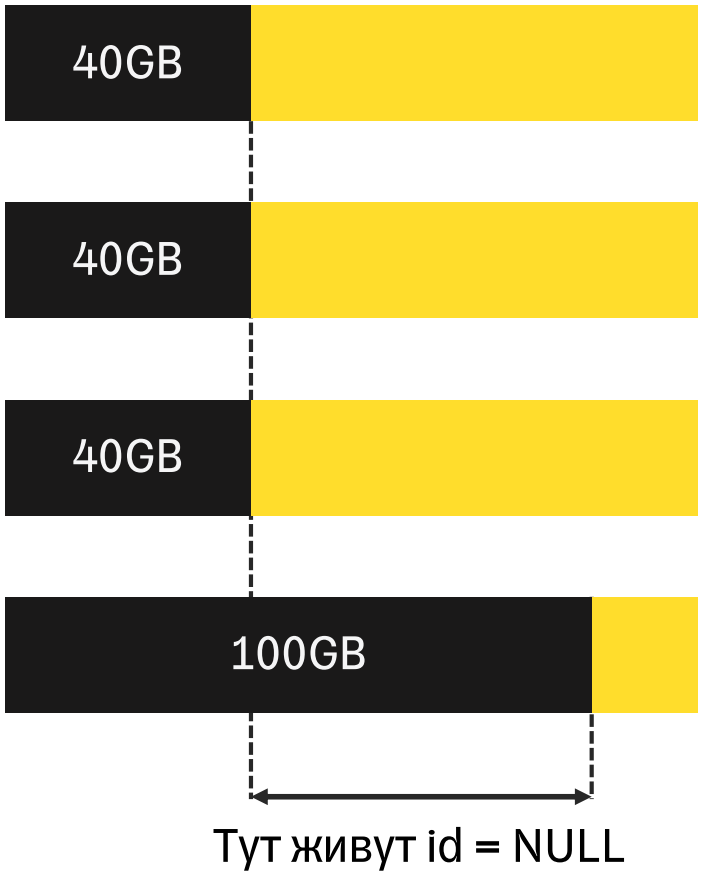
2. median – медиана числа строк в таблице в разрезе на узлы

3. max – максимальное число строк в таблице на узле

4. max/median < пороговое значение

5. Также встречается в: Vertica, ClickHouse, GreenPlum, Exasol

Advanced трюк



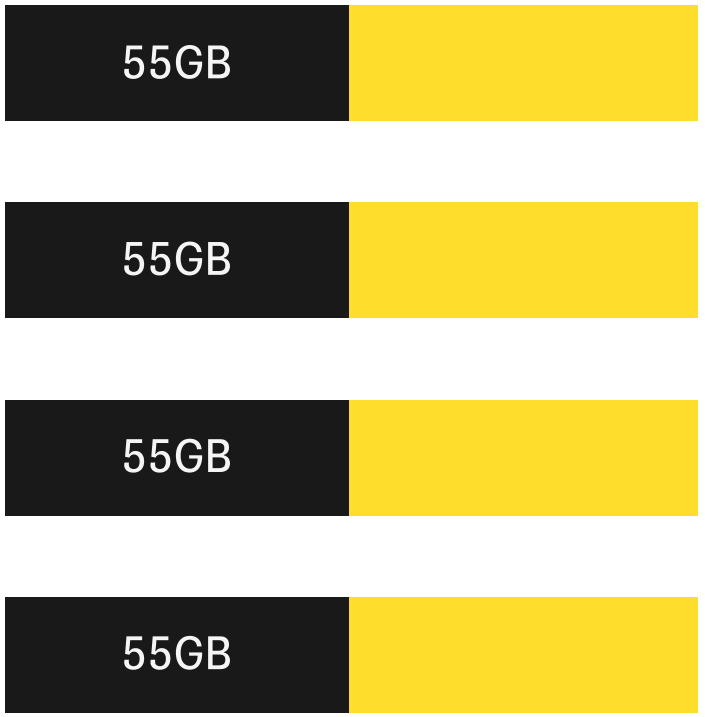

```
CREATE TABLE sample
(
    id INT
)
SEGMENTED BY hash(id) ALL NODES
```

Advanced трюк

```
INSERT INTO sample
SELECT
coalesce(id, -RANDOMINT(1000000))
FROM another_table
```

Advanced трюк

Было



Стало

Статические дефекты

1. Неоптимальные типы колонок (text/decimal)
2. Сортировка данных
3. Распределение данных
4. Партицирование данных
5. Перекосы данных
6. Прочее

Дефекты: прочее

1. Все колонки в таблице являются NULLABLE
2. GROUP BY по null-полям всегда дороже (Vertica - ~20%, ClickHouse - ~50%)
3. Физика мира такова, что очень редки «по-настоящему всё Nullable»
4. «Идеальный» подход – сделать профилирование данных (ресурсоёмко)

Дефекты

Ошибка проектирования структур данных или процессов их обработки

Статические

Когда эти данные будут обрабатывать – станет больно

- Таблицы
- Колонки

Динамические

Это уже сейчас работает неоптимальным образом

- Запросы
- Последовательности запросов

Динамические дефекты 

1. COST запроса
2. Soft-лимиты до watchdog

Дефекты: неоптимальные процессы

1. «Если это не починить – начнутся отстрелы запросов»
2. Суммарный объём временных данных превысил XXX байт
3. Размер временных данных на одном узле превысил XXX байт
4. Суммарный объём передачи по сети превысил XXX байт

Динамические дефекты 

1. COST запроса
2. Soft-лимиты до watchdog
3. Неоптимальные процессы

Дефекты: неоптимальные процессы

1. Общий объём обновленных данных более XXX байт

   Необходимо перейти на инкрементальное обновление

2. Общий объём прочитанных данных превышает XXX байт

   Пересмотреть процесс ETL – инкрементальное обновление

Динамические дефекты 

1. COST запроса
2. Soft-лимиты до watchdog
3. Неоптимальные процессы
4. Перекосы

Дефекты: перекосы

1. Время выполнения на узле больше медианного времени выполнения

   в 1.5 раза

   Перекос хранения, однопоточная выгрузка, bottleneck

2. Число строк, обработанных в пределах узла больше медианного

   в 1.5 раза

   Аналогично п.1

3. Объём временных данных на узле сильно выбивается на фоне остальных

Динамические дефекты 

1. COST запроса
2. Soft-лимиты до watchdog
3. Неоптимальные процессы
4. Перекосы
5. Прочее

Дефекты: прочее

1. Явный COMMIT после DDL-операции

COMMIT после DDL

1. Фиксация транзакции == блокировка глобального каталога (bottleneck)
2. Меньше блокировок => быстрее работа
3. DDL-операция вызывает неявный COMMIT
4. Пользователи ошибаются и расставляют COMMIT явно

```
TRUNCATE TABLE …. ;
COMMIT;

CREATE TABLE …. ;
COMMIT;

ALTER TABLE … ;
COMMIT;
```

Дефекты: прочее

1. Явный COMMIT после DDL-операции
2. Множественные SQL-инструкции в одном вызове

Множественные SQL-инструкции

1. Можно в одном вызове к БД можно послать много SQL-clause

2. Во время обработки на одном из них может быть ошибка

3. В python надо это «правильно обработать» или будет false-positive

```
query = '''
INSERT INTO table SELECT * FROM another_table;
SELECT 1 / 0; /* Вызовет ошибку */
COMMIT;
'''

connection.execute(query) /* не вылетел Exception */
```

Вовлечение пользователей

1. В разрезе на учетную запись собираем дефекты
2. Складываем статистику в БД
3. Рисуем нескучный UI

Потребление ресурсов | Статические дефекты | Статические дефекты (детали)

Фильтр

Дата

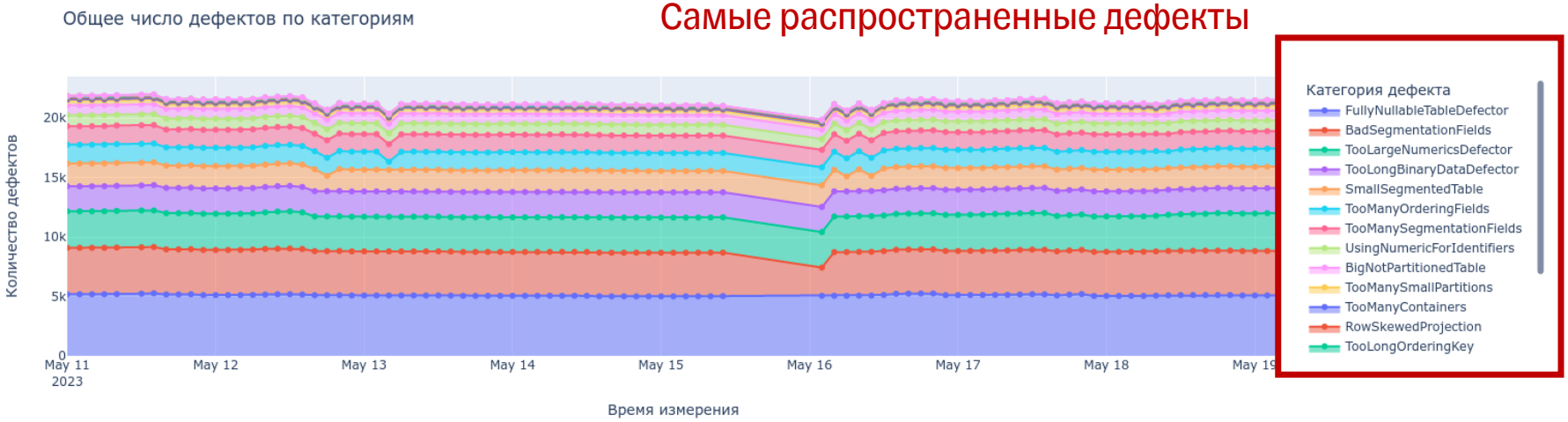
2023-05-11 → 2023-05-19

Содержимое: Все

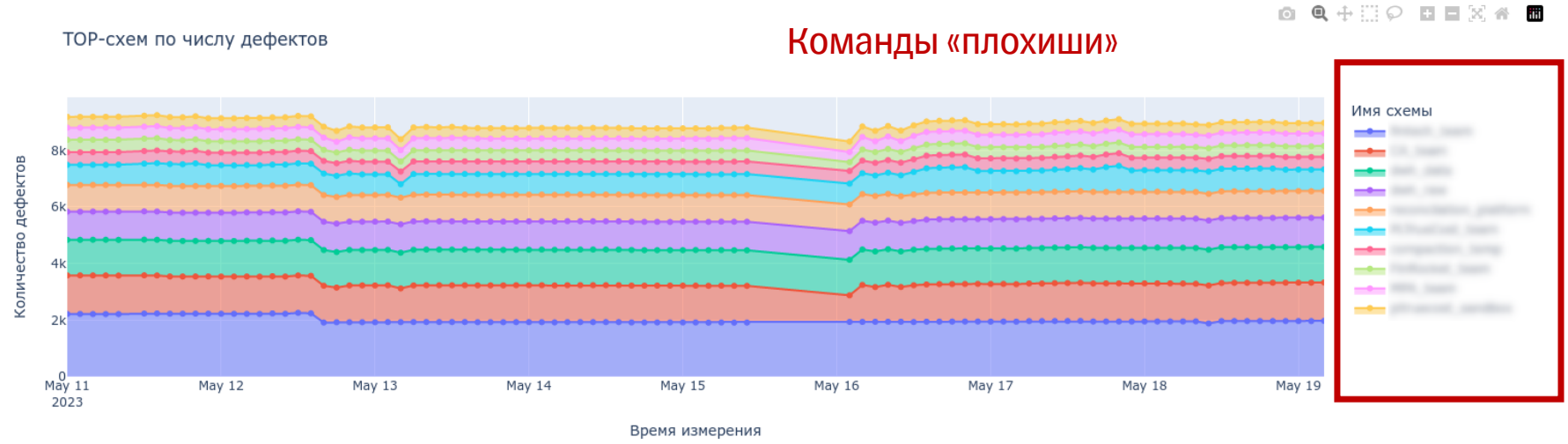
Схема: Все

Категория дефекта: Все

Самые распространенные дефекты



Команды «плохиши»



Вовлечение пользователей

1. В разрезе на учетную запись собираем дефекты
2. Складываем статистику в БД
3. Рисуем нескучный UI
4. Организуем еженедельную встречу с лидами команд

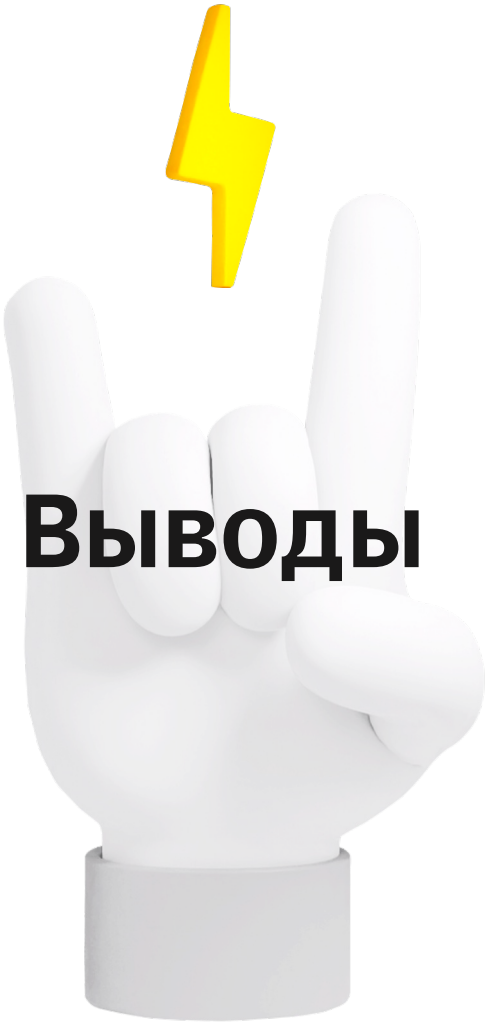
Ударники оптимизации и аутсайдеры

ТОП по динамике

Схема	Было дефектов	Стало дефектов	Дельта
filter data...			
[illegible]	2216	1972	-244
[illegible]	285	193	-92
[illegible]	351	288	-63
[illegible]	62	0	-62
[illegible]	438	379	-59
[illegible]	71	16	-55
[illegible]	104	68	-36
[illegible]	121	87	-34
[illegible]	170	143	-27
[illegible]	206	181	-25
[illegible]	75	107	32
[illegible]	182	214	32
[illegible]	40	74	34
[illegible]	91	130	39
[illegible]	118	157	39
[illegible]	998	1038	40
[illegible]	0	41	41
[illegible]	710	761	51
[illegible]	231	290	59
[illegible]	24	187	163

Вовлечение пользователей

1. В разрезе на учетную запись собираем дефекты
2. Складываем статистику в БД
3. Рисуем нескучный UI
4. Организуем еженедельную встречу с лидами команд
5. Пользователи челлеленджат друг друга



Выводы

Выводы

1. Квотируйте ресурсы на самом старте
2. Обозначайте правила игры с БД и границы дозволенного
3. Обучайте пользователей – документация, воркшопы, открытые площадки
4. Собирайте аналитику по деятельности процессов
5. Не бойтесь «отстреливать» процессы, убивающие массовое обслуживание
6. Создавайте прозрачность пользователям про показатели их процессов

That’s it, folks!