Погружение в СУБД. Сезон 2017

Нереляционные возможности

Дмитрий Барашев

Computer Science Center

Санкт-Петербург 2017

Сверстано в Папирии

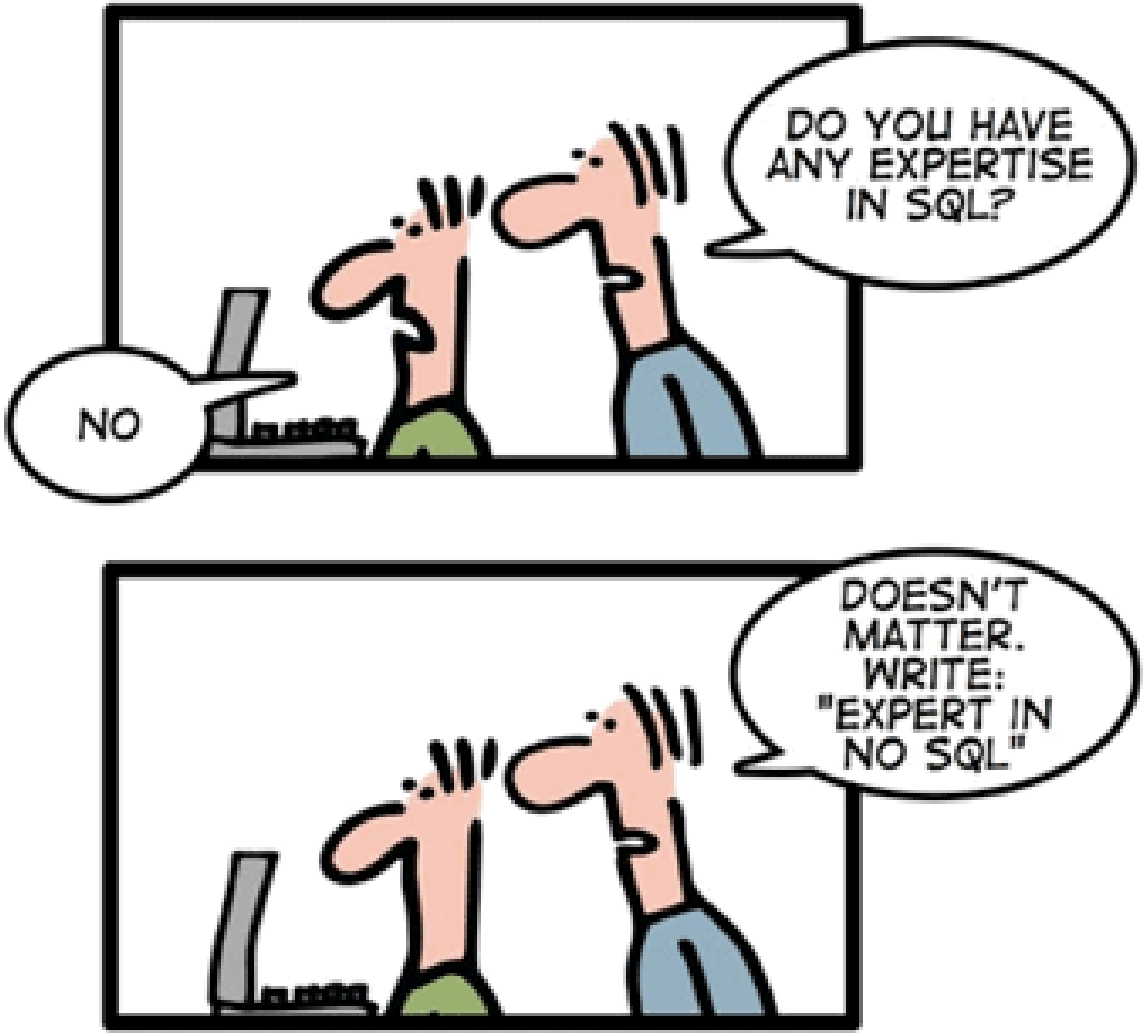
papeeria.com

онлайн редактор для LaTeX и Markdown• совместное редактирование в реальном времени • интеграция с Git репозиториями • графики

подсветка синтаксиса • автодополнение • проверка орфографии • предпросмотр математических формул • галерея шаблонов

Нереляционные возможности

Нереляционные возможности



Нереляционные возможности

- Что не так с реляционными СУБД

Нереляционные возможности

- Хранение и обработка JSON

Нереляционные возможности

- Модель ключ-значение

Нереляционные возможности

- Шардирование данных

Почему выбирают NoSQL

- Продуктивность разработчика
- Другая модель данных
- Горизонтальное масштабирование

Продуктивность

- Нужен софт, много софта
- Нужен сейчас, нет времени объяснять

Продуктивность

- Нужен софт, много софта
- Нужен сейчас, нет времени объяснять
- Времени вообще нет

Модели данных

- JSON и другие древовидные модели
- Ключ-значение
- Разреженные таблицы с неограниченным числом колонок и временны́м измерением
- Графы
- Почти неструктурированный текст

Модели данных

- JSON и другие древовидные модели

  MongoDB
- Ключ-значение
- Разреженные таблицы с неограниченным числом колонок и временны́м измерением
- Графы
- Почти неструктурированный текст

Модели данных

- JSON и другие древовидные модели

  MongoDB
- Ключ-значение

  Redis
- Разреженные таблицы с неограниченным числом колонок и временны́м измерением
- Графы
- Почти неструктурированный текст

Модели данных

- JSON и другие древовидные модели

  MongoDB
- Ключ-значение

  Redis
- Разреженные таблицы с неограниченным числом колонок и временны́м измерением

  Cassandra
- Графы
- Почти неструктурированный текст

Модели данных

- JSON и другие древовидные модели

  MongoDB
- Ключ-значение

  Redis
- Разреженные таблицы с неограниченным числом колонок и временны́м измерением

  Cassandra
- Графы

  Neo4J
- Почти неструктурированный текст

Модели данных

- JSON и другие древовидные модели

  MongoDB
- Ключ-значение

  Redis
- Разреженные таблицы с неограниченным числом колонок и временны́м измерением

  Cassandra
- Графы

  Neo4J
- Почти неструктурированный текст

  Elastic Search

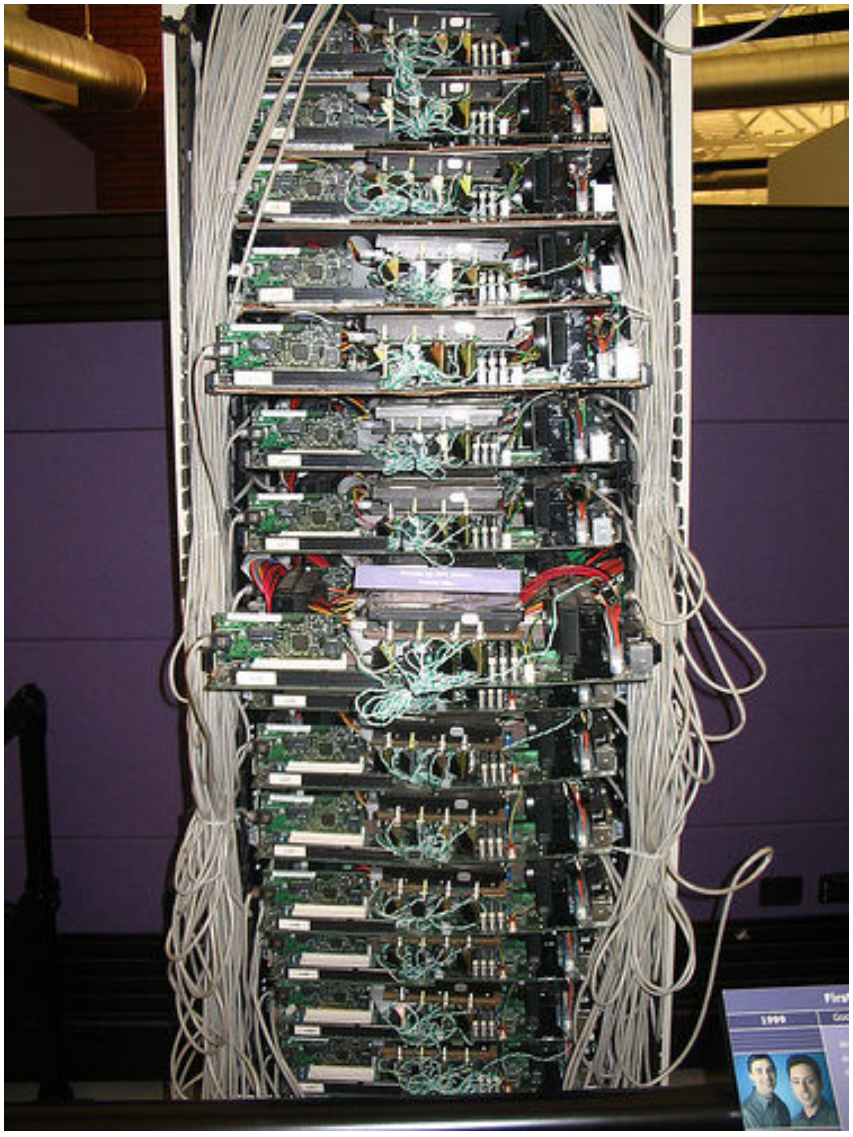
Масштабирование

Вертикальное масштабирование

IBM System Z10

Горизонтальное масштабирование

Первая боевая серверная стойка компании Google



Горизонтальное масштабирование

- Реляционным СУБД очень сложно дать гарантии ACID-транзакций
- От NoSQL систем часто ACID-транзакций и не ждут, а запаса кода 20-летней выдержки у них нет.

Почему не использовать NoSQL всегда?

Теряете нормализацию

```
[
  { id: 1,
    spacecraft: {name: 'Пегас', capacity: 3},
    commander: {name: 'Зелёный', rating: 'Competent'},
    planet: {name: 'Lave', government: 'Демократия'}
  },
  { id: 2,
    spacecraft: {name: 'Пегас', capacity: 3},
    commander: {name: 'Зелёный', rating: 'Competent'},
    planet: {name: 'Diso', government: 'Демократия'}
  },
  { id: 3,
    spacecraft: {name: 'Синяя Чайка', capacity: 2},
    commander: {name: 'Буран', rating: 'Elite'},
    planet: {name: 'Lave', government: 'Демократия'}
  }
]
```

Почему не использовать NoSQL всегда?

Так тоже можно. Но вы не захотите.

```
// Корабли
[
  {id: 1, name: 'Пегас', capacity: 3},
  {id: 2, name: 'Синяя Чайка', capacity: 2}
]
// Капитаны
[
  {id: 1, name: 'Зелёный', rating: 'Competent'},
  {id: 2, name: 'Буран', rating: 'Elite'}
]
// Планеты
[
  {id: 1, name: 'Lave', government: 'Демократия'},
  {id: 2, name: 'Diso', government: 'Демократия'}
]
// Полёты
[
  {spacecraft_id: 1, commander_id: 1, planet_id: 1},
  {spacecraft_id: 1, commander_id: 1, planet_id: 2},
  {spacecraft_id: 2, commander_id: 2, planet_id: 1}
]
```

Почему не использовать NoSQL всегда?

Теряете ACID транзакции: возможные грабли

- Результат подтверждённой транзакции станет виден другим транзакциям не сразу
- Не гарантируется повторяемость чтения
- Даже чтение только что лично тобой записанных данных не гарантируется

Почему не использовать NoSQL всегда?

Теряете оптимизатор

- Ты сам должен написать сложный запрос оптимальным способом.

Почему не использовать NoSQL всегда?

Теряете оптимизатор

- Ты сам должен написать сложный запрос оптимальным способом.
- Произошли качественные или количественные изменения в данных, запрос стал тормозить?

Почему не использовать NoSQL всегда?

Теряете оптимизатор

- Ты сам должен написать сложный запрос оптимальным способом.
- Произошли качественные или количественные изменения в данных, запрос стал тормозить?
- Перепиши его.

PostgreSQL и JSON

Типы данных

- JSON – документ в текстовом виде с сохранением пробелов и форматирования
- JSONB – бинарный JSON, более эффективный и поддерживающий более богатое множество операций

Добавляем JSON в таблицу

```
CREATE TABLE Flight(id SERIAL, flight_log JSONB);
INSERT INTO Flight(flight_log) VALUES ('
{
  "start": "2084-06-12 12:00 MSK",
  "launchpad": 1,
  "last_update": "2084-06-22",
  "log": [
    {"author": "Зелёный",
      "timestamp": "2084-06-12 12:00 MSK",
     "message": "Поехали!"},
    {"author": "Алиса",
      "timestamp": "2084-06-12 12:30 MSK",
      "mood": "отличное",
      "message": "Всем привет с Луны!"},
    {"author": "Профессор Селезнёв",
     "timestamp": "2084-06-22 17:30 MSK",
     "message": "Селфи с Шелезякой",
     "photo": "shelezyaka.jpg"}
  ]
}
')
```

Простейшие запросы

Найти все полёты, сделанные со стартовой площадки номер 1

```
SELECT id, flight_log
FROM Flight
WHERE (flight_log->>'launchpad')::INT = 1
```

Простейшие запросы

Найти все полёты, у которых была предстартовая проверка

```
SELECT id, flight_log
FROM Flight
WHERE flight_log ? 'launch_check'
```

Простейшие запросы

Найти все полёты, у которых была тщательная предстартовая проверка

```
SELECT id, flight_log
FROM Flight
WHERE flight_log -> 'launch_check' ->> 'type' = 'thorough'
---
SELECT id, flight_log
FROM Flight
WHERE flight_log #>> '{launch_check, type}' = 'thorough'
----
SELECT id, flight_log
FROM Flight
WHERE flight_log #>> ARRAY['launch_check', 'type']
    = 'thorough'
```

Простейшие запросы

Найти все полёты, в журнале которых есть сообщение ”Поехали!”

```
-- Не работает
SELECT id, flight_log
FROM Flight
WHERE flight_log->'log'->>'message' = 'Поехали!'
```

Поиск среди элементов массива

```
-- Работает
SELECT id, flight_log
FROM Flight
WHERE flight_log->'log' @> '[{"message": "Поехали!"}]'
----
-- Поиск конкретного
WITH good_log_records AS (
  SELECT
    id AS flight_id,
    jsonb_array_elements(flight_log->'log') AS record
  FROM Flight
  WHERE flight_log->'log' @> '[{"message": "Поехали!"}]'
)
SELECT flight_id, record->>'author' AS author
FROM good_log_records
WHERE record->>'message' = 'Поехали!'
```

Изменение JSON объектов

- Функции `jsonb_insert`, `jsonb_set` позволяют изменять JSON объекты
- Записывать значения в таблицу всё равно нужно традиционно:
  `UPDATE Flight SET flight_log = jsonb_set(flight_log, ....)`

Индексирование JSON

- Функциональные индексы на значения конкретных выражений
- GIN-индексы для операций проверки вложенности

Расширение JsQuery

```
-- Так!
SELECT * FROM Flight
WHERE flight_log @@ 'log.#.message = "Поехали!"';
-- Или так!
SELECT * FROM Flight
WHERE flight_log @@ '*.message = "Поехали!"';
-- Или даже так!
SELECT * FROM Flight WHERE flight_log
    @@ '*.message = "Поехали!"
        OR station = "Baikonur"';
```

- Разработано командой Postgres Professional
- Входит в дополнительные пакеты postgrespro

Полнотекстовый поиск в JSONB

```
CREATE INDEX idx_log_fulltext ON Flight
USING GIN ((to_tsvector('russian', flight_log)));

SELECT * from Flight
WHERE to_tsvector('russian', flight_log)
    @@ to_tsquery('russian', 'Селфи & Алиса');

SELECT * from Flight
WHERE to_tsvector('russian', flight_log)
    @@ plainto_tsquery('russian',
        'отличное селфи, Алиса');
```

- Разработано командой Postgres Professional
- Работает в стандартном PostgreSQL 10

PostgreSQL и ключ-значение

Задача

Для каждой записи в таблице хотим иметь нефиксированный набор свойств – пар ключ-значение

- Столбец на каждое свойство
- (Анти)-шаблон Entity-Attribute-Value
- Тип HSTORE

Entity-Attribute-Value

```
CREATE TABLE Flight(id SERIAL, ...);
CREATE TABLE FlightProps(
    flight_id INT REFERENCES Flight,
    name TEXT,
    value TEXT
);
```

Опции в PostgreSQL

- JSON/JSONB
- Тип HSTORE

Расширение HSTORE

Расширение HSTORE

Он вам не JSON

Индексирование HSTORE

- Поддерживаются GIN и GiST индексы
- Производительность примерно такая же, как у JSONB

Шардирование

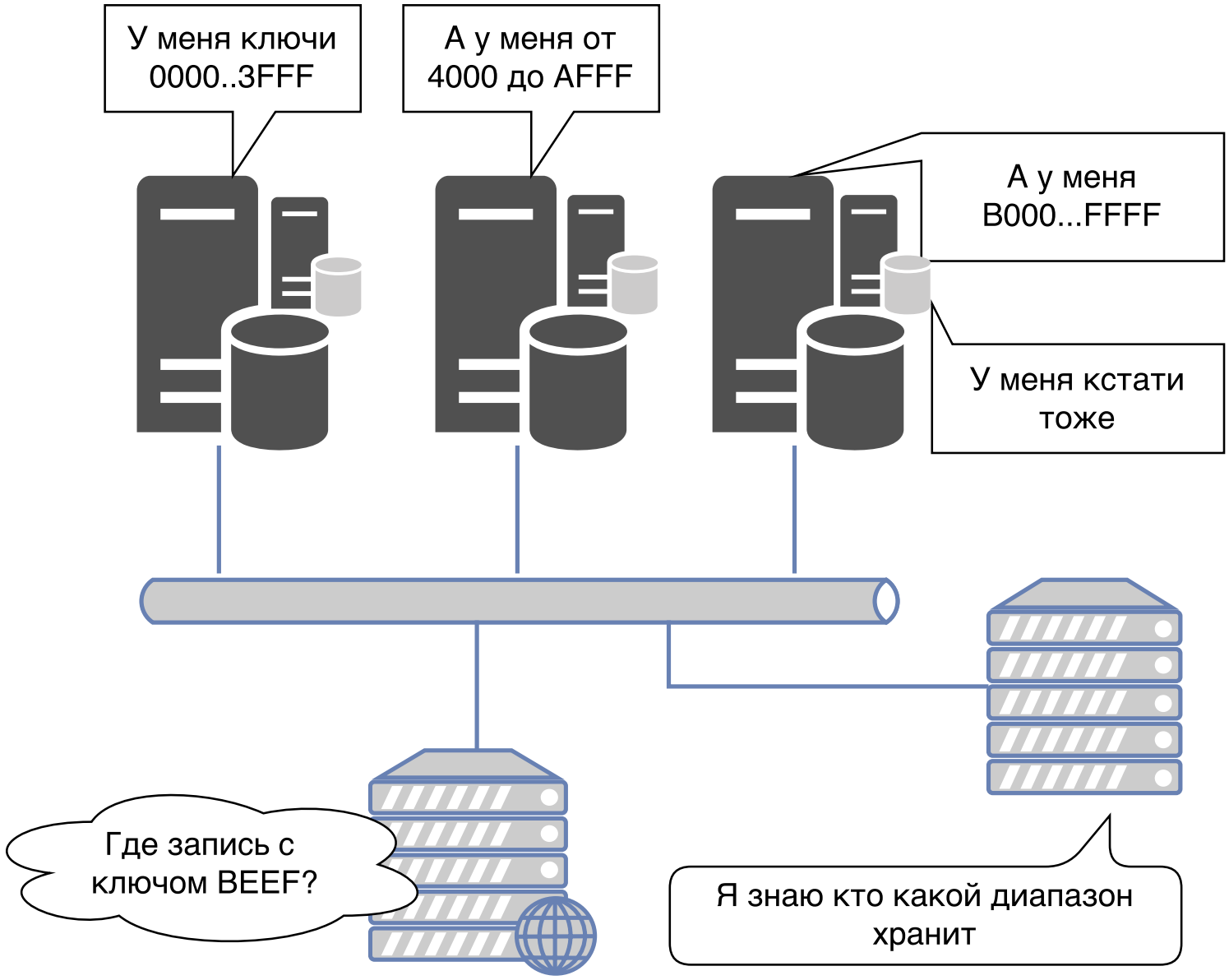
Key-Value хранилище

С точки зрения клиента

```
SELECT value FROM Store
WHERE key=’user:d1b665d528f3821c:profile’;
```

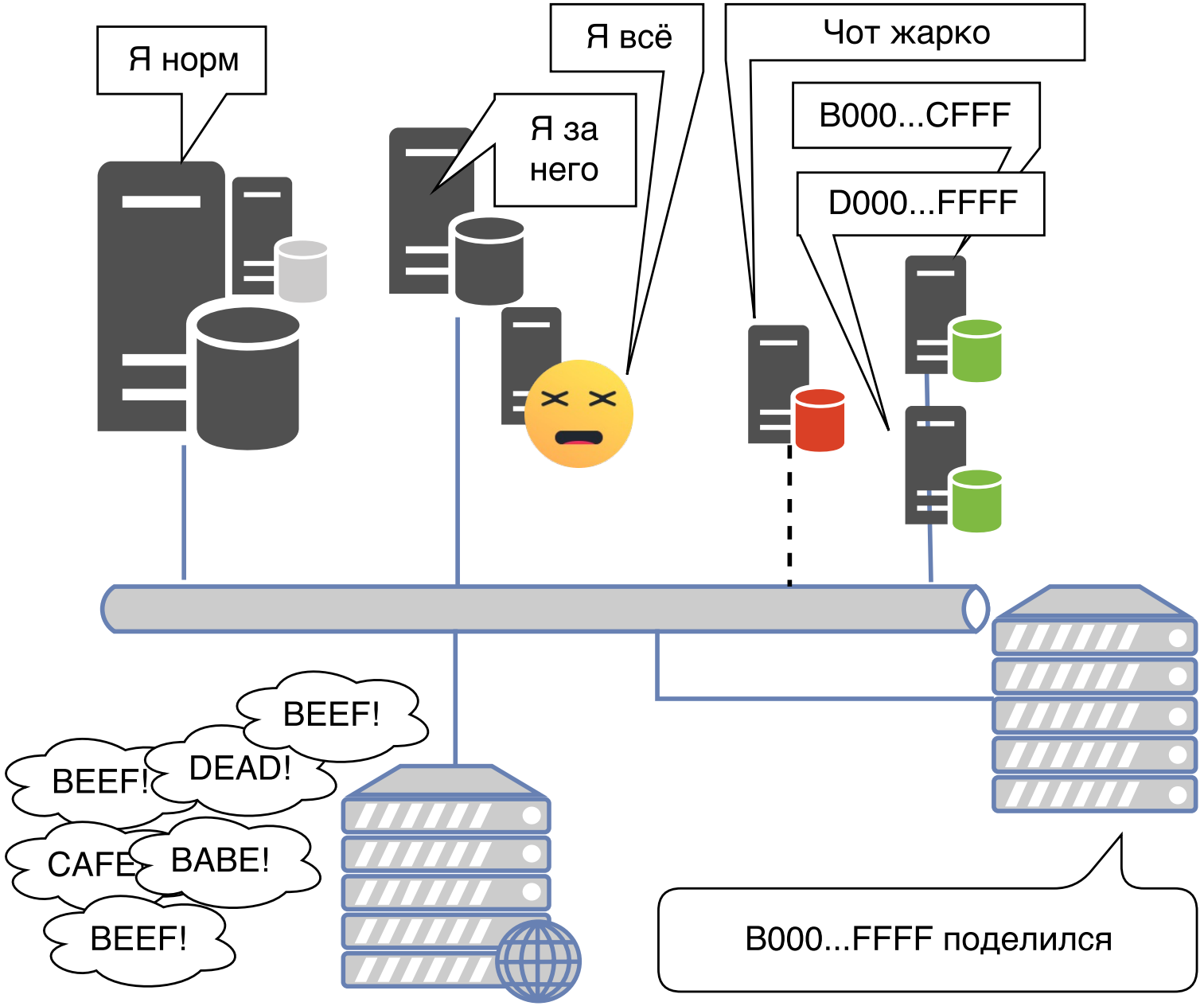
Key-Value хранилище

а под капотом. . .



Key-Value хранилище

. . . идёт жизнь



Эксперименты на коленке

из расширений и триггеров

- Расширение `postgres_fdw`, позволяющее обращаться к ”удалённым” таблицам как к своим собственным
- Представление, изменяемое при помощи триггеров

Промышленные решения

- Postgres Professional, расширение `multimaster`. Кластер из нескольких экземпляров Postgres с идентичными данными и чтением и записью на любой экземпляр

Промышленные решения

- Citus Data, расширение `citus`. Шардирование таблиц.

Решения компании Citus Data

- Бесплатная сборка: инфраструктура для масштабирования
- Коммерческая сборка: инфраструктура + автоматизация
- Облачный сервис

Расширение CITUS

- Узел-координатор
- Рабочие узлы

Расширение CITUS

Координатор

- Принимает все запросы
- Хранит разбиение пространства ключей на логические шарды
- Хранит привязку логического шарда в рабочие узлы
- Строит план выполнения запроса
- Следит за состоянием рабочих узлов
- Сам никакие шарды не обслуживает

Расширение CITUS

Рабочий узел

- Получает запрос от координатора и выполняет его

Расширение CITUS

Варианты шардирования

- Простое шардирование одной таблицы по хешу ключа
- Согласованное шардирование связанных таблиц
- Наращиваемое шардирование

Расширение CITUS

Логические шарды

- Логический шард – диапазон значений ключей
- Даже если рабочих узлов мало, логических шардов должно быть достаточно много
- Один узел будет обслуживать несколько шардов
- В случае роста нагрузки часть его шардов переедет на другие узлы

Что нужно запомнить

про NoSQL

- Прежде чем записывать JSON в БД, подумайте, нельзя ли этого не делать

Что нужно запомнить

про NoSQL

- Если всё же очень нужно, то JSONB работает прекрасно

Что нужно запомнить

про NoSQL

- HSTORE неплох для среднего размера отображений ключ-значение

Что нужно запомнить

про NoSQL

- Шардирование PostgreSQL только начинается и кое-что уже работает