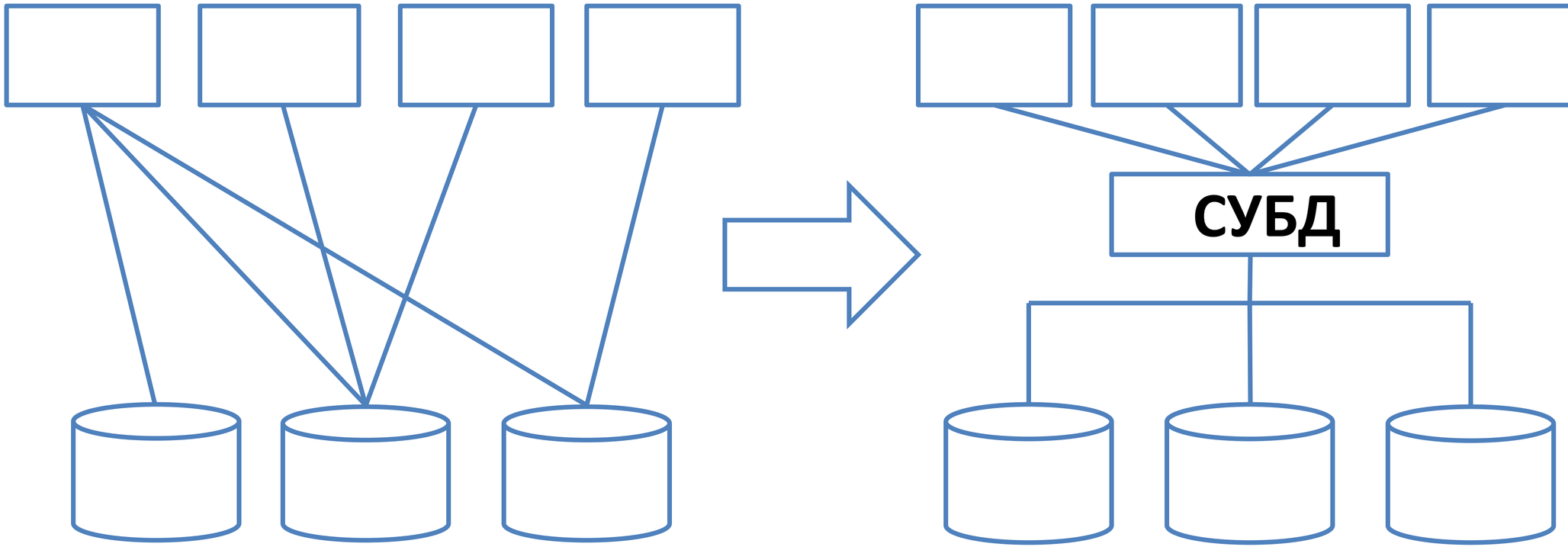
Основы технологий баз данных

Б. А. Новиков
Санкт-Петербургский университет, JetBrains Research

Предыдущие серии

краткое содержание

Централизация управления общими данными нескольких приложений



Модели данных

- Основные свойства
  - Методы идентификации объектов внутри и вне модели
  - Поиск и связывание объектов
  - Массовая или штучная обработка объектов данных
- Реляционная модель данных
  - Домены, атрибуты, отношения
  - Реляционные языки запросов
  - Функциональные зависимости, ключи, Нормальные формы
- Другие модели данных

Язык запросов SQL

- Декларативный язык
- Структуры данных и операторы описания данных: CREATE, DROP, ALTER
- Ограничения целостности: NOT NULL, CHECK, UNIQUE, PRIMARY KEY, FOREIGN KEY
- Операторы: SELECT, INSERT, UPDATE, DELETE
- Реализация операций реляционной алгебры в SQL и другие возможности SQL

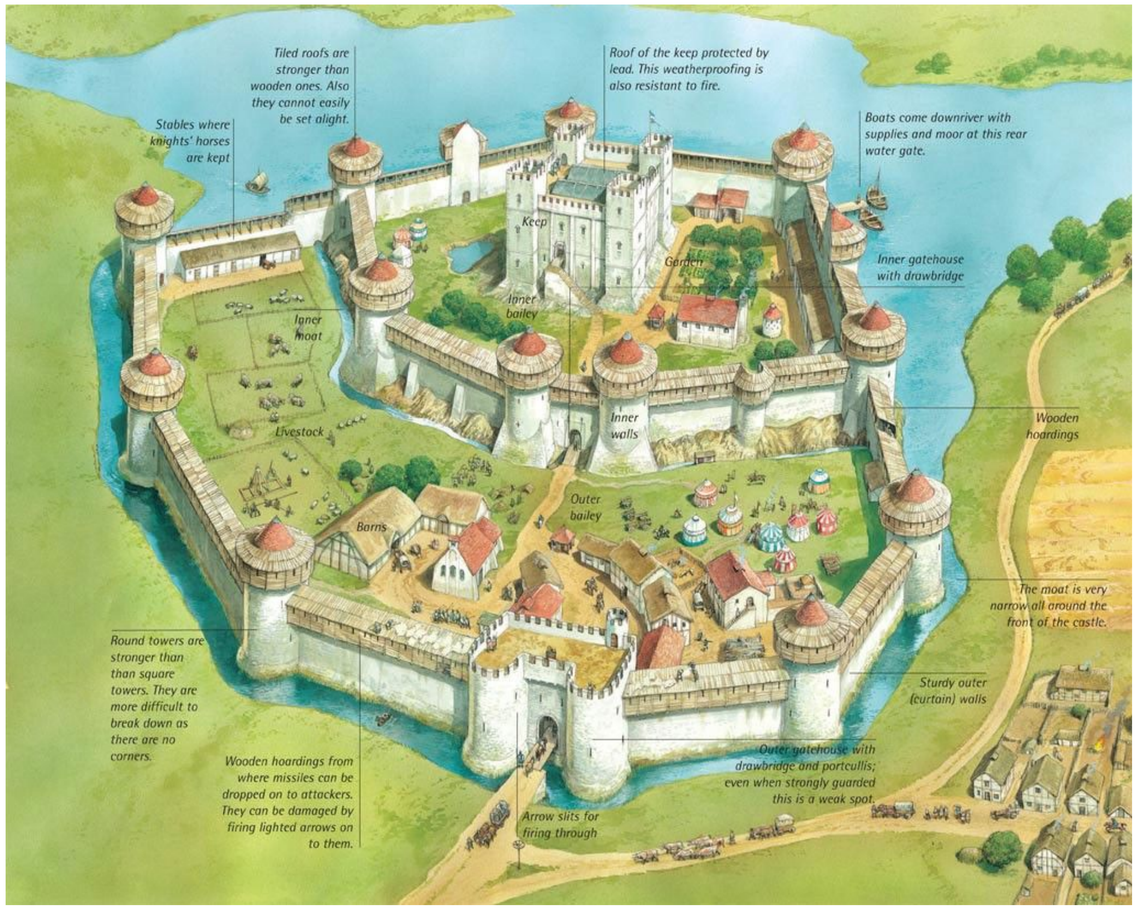
Postgres PROFESSIONAL

Внешние ключи и другие мифы

- FOREIGN KEY — ограничение целостности
- Никак не влияет на семантику запросов на чтение данных
- Соединение (JOIN) никак не связано с понятием внешнего ключа, хотя часто ключом соединения является внешний ключ
- Ограничения целостности никак не влияет на эффективность операции соединения
- Миф о влиянии возник потому, что для внешних ключей часто создаются индексы

Защита должна быть похожа на крепость

- Физическая защита оборудования
- Защита в вычислительной сети
- Операционная система
- Файловая система
- Сервер приложений и приложения
- База данных



Согласованность и надежность

- Согласованность необходима для сохранения корректности данных
- Простые протоколы ограничивают пропускную способность системы
- Многоверсионные протоколы SI существенно превосходят протоколы двухфазового блокирования
- Средства защиты от отказов позволяют обеспечить любую степень надежности
- Создание и эксплуатация высоконадежных систем оказываются дорогостоящими

Разработка приложений

Альтернативные подходы к разработке приложений

- Тщательное проектирование базы данных и приложения (11%)

  \+ Высокое качество результата разработки

  – Высокая стоимость

- Применение средств быстрой разработки

  \+ Относительно низкая стоимость и быстрое получение результата

  – О качестве говорить не приходится

- Компромиссные варианты

Модели прикладной системы

- Функциональная модель
- Модель потоков данных
- Модель бизнес-процесов
- Модель базы данных (обычно «сущность-связь»)
- Объектная модель приложения

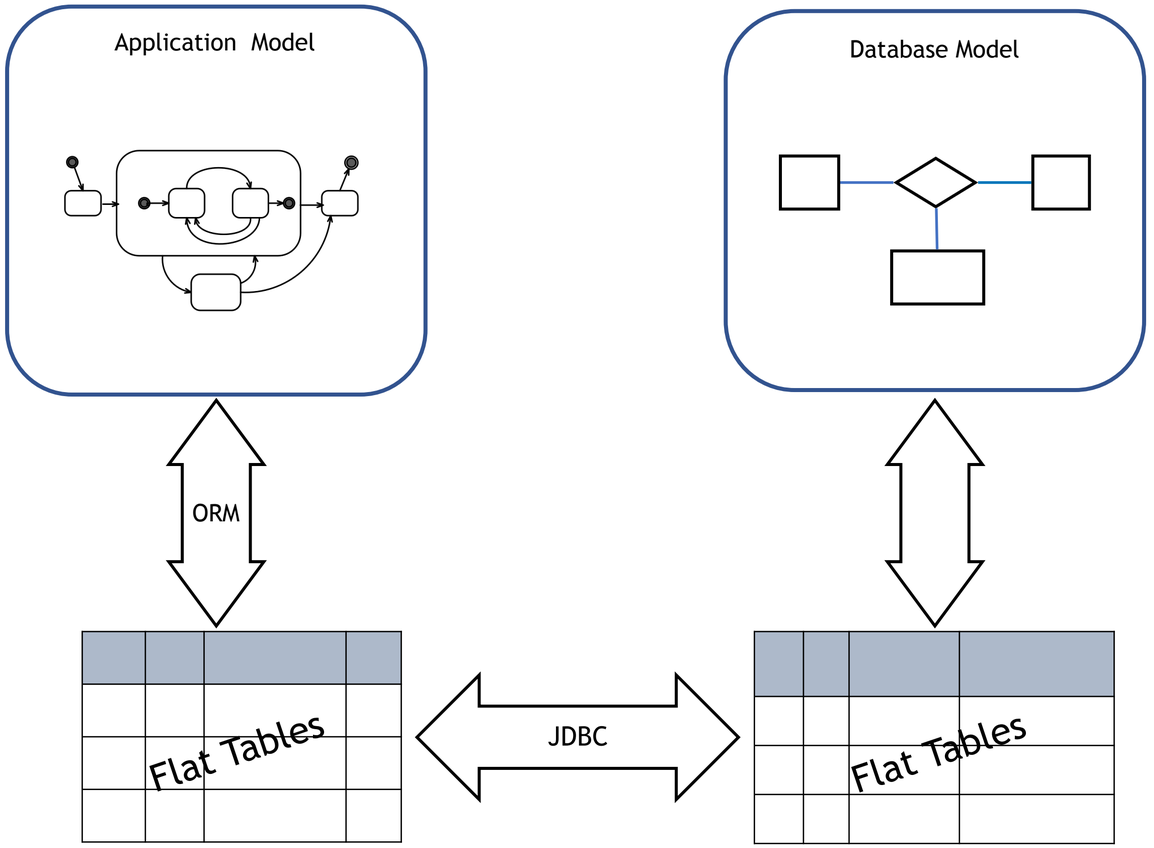
Объектно-реляционная потеря соответствия (Impedance Mismatch)

	О-О языки программирования	Реляционные БД
Идентификация	суррогаты	естественные ключи
Обработка	отдельные объекты	множества
Поиск	навигация	поиск по значениям

- Различие в базовых типах данных
- Различие в методах представления сложных объектов
- Невозможность передачи сложных объектов через интерфейсы доступа к БД

Объектно-реляционные отображения

- Описание данных генерируется из модели приложения
- Упрощенная модель БД
  - игнорируются связи
  - не используются возможности языка запросов
  - дублирование функций БД в приложении



Базы данных работают медленно. Почему?

- Слишком много слишком мелких запросов
- Синхронные обмены данными приводят к ожиданиям даже на сетях с высокой пропускной способностью
- Как приложение, так и СУБД находятся в ожидании более 95% времени
- Распределенные системы не могут улучшить время отклика

Генерация HTML-форм, ~100 полей

год	язык	кол-во запросов	время
2003	Perl	16 000	5 мин
2013	Ruby	25 000 – 30 000	50–70 сек

Мифы программной инженерии

- Performance is not an issue
- Увеличение мощности оборудования решит все проблемы
- Масштабируемость может компенсировать недостаточную производительность
- Стоимость разработки является доминирующим фактором

Важно ли время отклика?

- Если время отклика превышает несколько секунд, значительная доля покупателей уходит к конкурентам

Компромиссы

- Поддержка логической модели приложения, на основе которой создаются объектная модель приложения и модель базы данных
- Начальная (полученная автоматически) версия модели данных подвергается ручной доработке
- Схема базы данных создается и модифицируется с помощью SQL-скриптов

Наследование
Отображение в одну таблицу

persons

class	name	title	degree	stud_id
person	Елена	секретарь		
person	Анастасия	лаборант		
professor	Аристарх	профессор	2001	
professor	Ксенофон	доцент	2012	
student	Анна	студент		1451
student	Виктор	студент		1432
student	Нина	студент		1556

Наследование
Горизонтальное разбиение

persons

name	title
Елена	секретарь
Анастасия	лаборант

professors

name	title	degree
Аристарх	профессор	2001
Ксенофон	доцент	2012

students

name	title	stud_id
Анна	студент	1451
Виктор	студент	1432
Нина	студент	1556

Наследование
Вертикальное разбиение

persons

name	title
Елена	секретарь
Анастасия	лаборант
Аристарх	профессор
Ксенофон	доцент
Анна	студент
Виктор	студент
Нина	студент

professors

name	degree
Аристарх	2001
Ксенофон	2012

students

name	stud_id
Анна	1451
Виктор	1432
Нина	1556

Программирование запросов: Thinking Sets

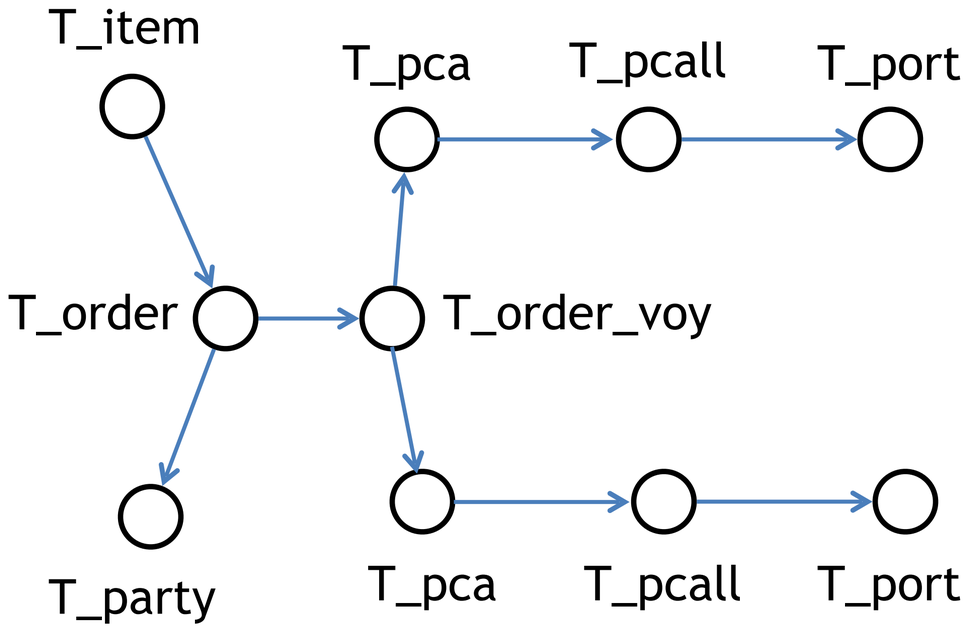
- Определить (содержательно), какие элементы составляют множество строк результата
- Построить предложения FROM и, возможно, GROUP BY (получить ключ результата)
- Предложение WHERE (выбрать нужные строки)
- Выражения, определяющие колонки результата
- Возможно, применять эти шаги рекурсивно для построения подзапросов

Гранулярность запросов

- Количество запросов не должно быть пропорционально количеству данных, получаемых из БД
- Если выбирается какая-либо строка таблицы, то все нужные атрибуты должны быть выбраны в том же запросе
- Условия селекции должны по возможности точно выделять необходимые для обработки данные

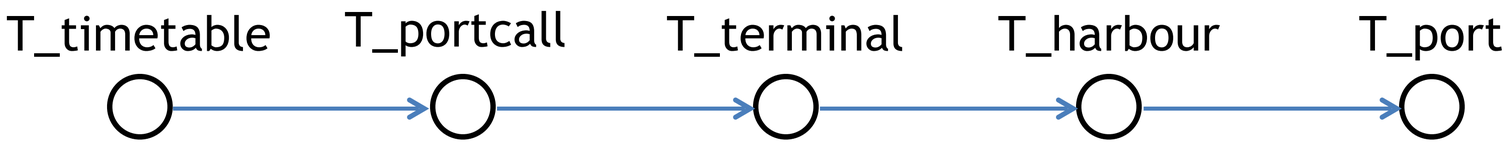
Диаграмма запроса

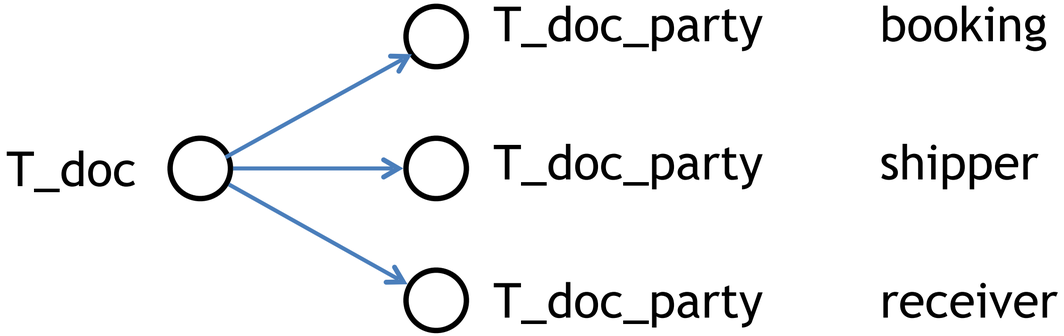
```
SELECT
   needed columns
FROM t_item
JOIN t_order ON ...
JOIN t_order_voy ON ...
JOIN t_pcadep ON ...
JOIN t_pcaarr ON ...
JOIN t_pc depc ON ...
JOIN t_pcall arrc ON ...
JOIN t_port depp ON ...
JOIN t_port arrp ON ...
JOIN t_party book ON ...
WHERE
   some conditions
```



Построение диаграмм запросов

- Цепочки ссылок (первичный-внешний ключи)
- Связи первичный-внешний с дополнительными условиями
- Содержательные предикаты соединения (не обязательно первичный-внешний ключ)

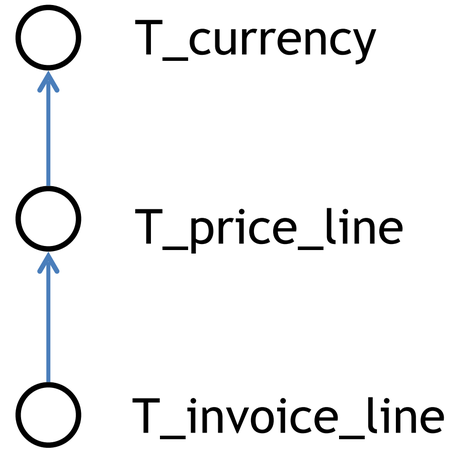




Подзапросы и диаграммы

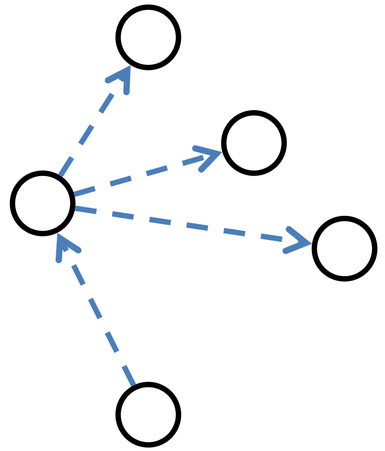
- Для подзапросов создаются вершины на диаграмме

```
SELECT ...,
 (SELECT currency_cod
  FROM t_currency
  WHERE currency_id = pl.currency_id) cur
FROM t_price_line pl
WHERE pl.item_id = 12345
AND NOT EXISTS(
  SELECT 1
  FROM t_invoice_line
  WHERE priceline_id = pl.priceline_id)
```



Влияние функций, содержащих подзапросы

```
SELECT
  get_name(item_id),
  get_currency_code(item_id),
  get_length(item_id),
  get_status(item_id)
FROM t_item
WHERE item_wait_bol = ‘Y’
```



Проектирование транзакций в приложении

- Атомарность: вывод результатов только после фиксации транзакции
- Блокировки не должны удерживаться в течение длительного времени
- Недопустимо взаимодействие с пользователем, пока транзакция не завершена
- Ослабленные критерии согласованности при вычислении агрегированных значений

Optimistic locking

- Управление согласованностью на уровне приложений
- Предотвращает потери обновления в интернет-приложениях
- Каждый объект данных (например, строка таблицы) дополняется атрибутом, хранящим время последнего обновления
- Данные читаются в режиме Read Committed и отображаются пользователю
- Перед обновлением проверяется время обновления объекта данных.
  Если оно отличается от прочитанного ранее, операция обновления отвергается

Postgres PROFESSIONAL

Optimistic locking: уточнения

- При проверке корректности необходимо считывать метку всех данных, которые были использованы или показаны пользователю
- Проверка только меток обновляемых строк таблиц недостаточна
- Вместо меток можно проверять значения всех атрибутов

Разработка приложений: итоги

- Объектно-ориентированные методологии и средства разработки не могут учесть особенности моделей данных
- Необходим компромисс между качеством результата и стоимостью разработки
- Автоматически построенные схемы базы данных обычно требуют доработки
- Не следует использовать слишком мелкие запросы
- Должна быть обеспечена согласованность

Расширения SQL в PostgreSQL

Типы данных

- Домены
- Составные типы данных
- Массивы
- Базовые типы данных

Массивы

```
demo=# SELECT array_agg(airport_code)
FROM (SELECT * FROM airports LIMIT 4) air;
     array_agg
-------------------
 {YKS,MJZ,KHV,PKC}
(1 row)
```

```
demo=# SELECT *
FROM unnest(ARRAY['AAQ','ABA','AER','ARH']);
 unnest
--------
 AAQ
 ABA
 AER
 ARH
(4 rows)
```

Составные типы данных

```
demo=# CREATE TYPE departure AS (
  scheduled_departure timestamp with time zone,
  departure_airport_code text);
CREATE TYPE
```

```
demo=# CREATE TYPE arrival AS (
  scheduled_arrival timestamp with time zone,
  arrival_airport_code text);
CREATE TYPE
```

```
demo=# CREATE TYPE booking_leg AS (
  depart  departure,
  arrive arrival);
CREATE TYPE
```

```
demo=# CREATE TYPE booking_info AS (
  booking_ref text,
  legs booking_leg[] );
CREATE TYPE
```

Заполнение массива из базы данных

```
demo=# SELECT (b.book_ref,
 (SELECT array_agg( (
     (f.scheduled_departure, f.departure_airport)::departure,
     (f.scheduled_arrival, f.arrival_airport)::arrival
   )::booking_leg )
  FROM tickets t
    JOIN ticket_flights ft ON t.ticket_no = ft.ticket_no
    JOIN flights f ON ft.flight_id = f.flight_id
  WHERE t.book_ref = b.book_ref)
 )::booking_info
FROM bookings b LIMIT 3;
```

Заполнение массива из базы данных

```
row
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
 (000004,"{""(\\""(\\""\\""2015-10-28 10:05:00+03\\""\\"",DME)\\"",\\""(\\""\\""2015-10-28 13:30:00+03\
\""\\"",OVB)\\"")"","""(\\""(\\""\\""2015-11-06 10:50:00+03\\""\\"",OVB)\\"",\\""(\\""\\""2015-11-06
14:15:00+03\\""\\"",DME)\\"")""}")
 (00000F,"{""(\\""(\\""\\""2016-09-13 16:15:00+03\\""\\"",SVO)\\"",\\""(\\""\\""2016-09-14 01:00:00+03\
\""\\"",UUS)\\"")"","""(\\""(\\""\\""2016-09-21 09:45:00+03\\""\\"",UUS)\\"",\\""(\\""\\""2016-09-21
18:30:00+03\\""\\"",SVO)\\"")""}")
 (000010,"{""(\\""(\\""\\""2016-03-22 10:15:00+03\\""\\"",DME)\\"",\\""(\\""\\""2016-03-22 12:25:00+03\
\""\\"",OVS)\\"")"","""(\\""(\\""\\""2016-03-22 15:15:00+03\\""\\"",OVS)\\"",\\""(\\""\\""2016-03-22
16:30:00+03\\""\\"",UFA)\\"")"","""(\\""(\\""\\""2016-03-22 10:15:00+03\\""\\"",DME)\\"",\\""(\\""\
\""2016-03-22 12:25:00+03\\""\\"",OVS)\\"")"","""(\\""(\\""\\""2016-03-22 15:15:00+03\\""\\"",OVS)\\"",\
\""(\\""\\""2016-03-22 16:30:00+03\\""\\"",UFA)\\"")""}")
(3 rows)
```

Слабоструктурированные типы данных

- XML — инструмент для передачи данных
  - предшественники: SGML, HTML
- JSON, JSONB
- Значение (единственного) атрибута в одной строке
- Различные значения в нескольких строках и/или атрибутах
- Развитый набор функций и операторов для преобразований между моделями данных в другие типы и т. п.

Конструирование объектов JSON

```
demo=# SELECT
  json_build_object('code', airport_code, 'name', airport_name)
FROM airports
LIMIT 3;
                json_build_object
-----------------------------------------------
 {"code" : "YKS", "name" : "Якутск"}
 {"code" : "MJZ", "name" : "Мирный"}
 {"code" : "KHV", "name" : "Хабаровск-Новый"}
(3 rows)
```

Выделение пар ключ-значение из JSON

```
demo=# SELECT *
FROM json_each(
  '{"code" : "KVK", "name" : «Апатиты–Кировск"}'
);
 key  |        value
------+---------------------
 code | "KVK"
 name | "Апатиты–Кировск"
(2 rows)
```

Выделение значений составного типа из JSON

```
demo=# SELECT *
FROM json_populate_record( NULL::airports,
  '{"airport_code" : "KVK",
    "name" : "Апатиты-Кировск",
    "city" : "Кировск"}'
);
 airport_code | airport_name |  city   | coordinates | timezone
--------------+--------------+---------+-------------+----------
 KVK          |              | Кировск |             |
(1 row)
```

Создание сложного объекта JSON

```
demo=# SELECT to_json( (b.book_ref,
 (SELECT array_agg( (
     (f.scheduled_departure, f.departure_airport)::departure,
     (f.scheduled_arrival, f.arrival_airport)::arrival
   )::booking_leg )
  FROM tickets t
    JOIN ticket_flights ft ON t.ticket_no = ft.ticket_no
    JOIN flights f ON ft.flight_id = f.flight_id
  WHERE t.book_ref = b.book_ref)
 )::booking_info ) booking
FROM bookings b LIMIT 3;
```

JSON: результат выборки


```
 {"booking_ref":"000004","legs":[{"depart":
{"scheduled_departure":"2015-10-28T10:05:00+03:00","departure_airport_code":"DME"},"arrive":
{"scheduled_arrival":"2015-10-28T13:30:00+03:00","arrival_airport_code":"OVB"}},{"depart":
{"scheduled_departure":"2015-11-06T10:50:00+03:00","departure_airport_code":"OVB"},"arrive":
{"scheduled_arrival":"2015-11-06T14:15:00+03:00","arrival_airport_code":"DME"}}]}
 {"booking_ref":"00000F","legs":[{"depart":
{"scheduled_departure":"2016-09-13T16:15:00+03:00","departure_airport_code":"SVO"},"arrive":
{"scheduled_arrival":"2016-09-14T01:00:00+03:00","arrival_airport_code":"UUS"}},{"depart":
{"scheduled_departure":"2016-09-21T09:45:00+03:00","departure_airport_code":"UUS"},"arrive":
{"scheduled_arrival":"2016-09-21T18:30:00+03:00","arrival_airport_code":"SVO"}}]}
 {"booking_ref":"000010","legs":[{"depart":
{"scheduled_departure":"2016-03-22T10:15:00+03:00","departure_airport_code":"DME"},"arrive":
{"scheduled_arrival":"2016-03-22T12:25:00+03:00","arrival_airport_code":"OVS"}},{"depart":
{"scheduled_departure":"2016-03-22T15:15:00+03:00","departure_airport_code":"OVS"},"arrive":
{"scheduled_arrival":"2016-03-22T16:30:00+03:00","arrival_airport_code":"UFA"}},{"depart":
{"scheduled_departure":"2016-03-22T10:15:00+03:00","departure_airport_code":"DME"},"arrive":
{"scheduled_arrival":"2016-03-22T12:25:00+03:00","arrival_airport_code":"OVS"}},{"depart":
{"scheduled_departure":"2016-03-22T15:15:00+03:00","d
```

Выборка XML из таблиц

```
demo=# SELECT xmlelement(
    name flight,
    xmlforest(  dep.airport_code AS dep, arr.airport_code AS arr) )
FROM ticket_flights tf
  JOIN flights f ON tf.flight_id = f.flight_id
  JOIN airports dep ON f.departure_airport = dep.airport_code
  JOIN airports arr ON f.arrival_airport = arr.airport_code
WHERE tf.ticket_no = '0005432369015'
ORDER BY f.scheduled_departure;
                  xmlelement
-----------------------------------------------
 <flight><dep>VKO</dep><arr>BZK</arr></flight>
 <flight><dep>BZK</dep><arr>EGO</arr></flight>
 <flight><dep>EGO</dep><arr>AAQ</arr></flight>
 <flight><dep>AAQ</dep><arr>EGO</arr></flight>
 <flight><dep>EGO</dep><arr>BZK</arr></flight>
 <flight><dep>BZK</dep><arr>VKO</arr></flight>
(6 rows)
```

Извлечение таблицы из XML

```
demo=# SELECT *
FROM xmltable( '//flights/flight'
  PASSING ( SELECT xmlconcat( '<ticket>
       <number>0005432369015</number>
       <flights><flight><from>VKO</from><to>BZK</to></flight>
                <flight><from>BZK</from><to>EGO</to></flight>
                <flight><from>EGO</from><to>AAQ</to></flight>
       </flights>
    </ticket>' ) )
  COLUMNS
    departs_from text PATH 'from',
    arrives_to   text PATH 'to');
 departs_from | arrives_to
--------------+------------
 VKO          | BZK
 BZK          | EGO
 EGO          | AAQ
(3 rows)
```

Другие типы в PostgreSQL

- Денежные типы
- Перечислительные типы
- Типы для сетевых адресов
- Геометрические типы
- Типы для текстового поиска

Расширения SQL: итоги

- Реализованы логические структуры объектно-реляционной модели данных
- Гибкие возможности для хранения слабоструктурированных типов данных в форматах JSON и XML
- Преобразование данных из одной модели в другую
- Система типов для отдельных классов прилоежний (геометрические, текстовый поиск)
- Разнообразные возможности расширения

Многообразие

Структуры хранения: строки или колонки?

1	Анна	БД	5
2	Анна	МО	5
3	Виктор	БД	4
4	Виктор	МО	5
5	Елена	БД	5
6	Елена	МО	4
7	Игорь	БД	3

1	Анна
2	Анна
3	Виктор
4	Виктор
5	Елена
6	Елена
7	Игорь

1	БД
3	БД
5	БД
7	БД
2	МО
4	МО
6	МО

7	3
3	4
6	4
1	5
2	5
4	5
5	5

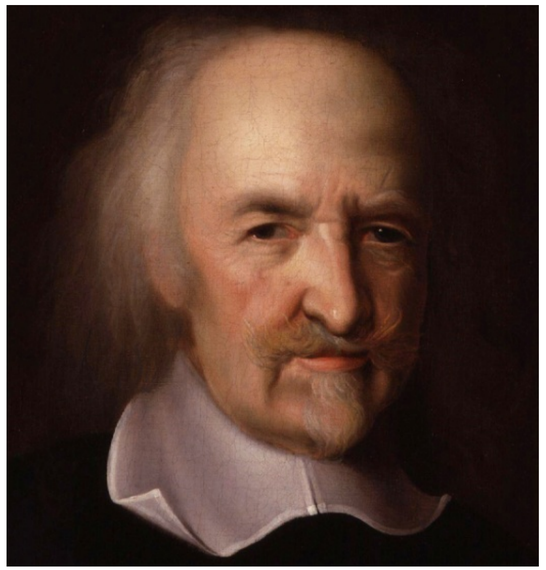
Движение noSQL

- Объектно-ориентированные СУБД
- Хранилища ключ-значение
- Документо-ориентированные системы
- Системы хранения колонок
- Распределенные масштабируемые системы

Задеты интересы крупных производителей СУБД

«Если бы геометрические аксиомы задевали интересы людей, они бы опровергались»

Томас Гоббс (англ. Thomas Hobbes)
1588–1679



Аксиомы noSQL

- Согласованность (ACID) недостижима в распределенных системах
- Основанные на SQL системы не поддерживают сложные структуры данных
- Реляционные системы не масштабируемы
- Системы SQL не обеспечивают достаточную производительность

Магия слов: «теорема CAP»

- Слово «теорема» предполагает наличие математического доказательства, однако:

«These are not (yet) provable principles, but merely ways to think about the issues that simplify design in practice»

Eric A. Brewer. 2000. Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing* (PODC '00). ACM, New York, NY, USA, 7-. DOI=10.1145/343477.343502 http://doi.acm.org/10.1145/343477.343502

Отказ от ACID в пользу производительности?

- Логическая цепочка:

  согласованность => транзакции => блокировки => низкая степень параллелизма

- Никак не связано с SQL

- СВОЕ: более 10 миллиардов обновляющих ACID-транзакций в сутки, СУБД Oracle

- Протокол управления транзакциями для распределенных систем, более 200 000 обновляющих ACID-транзакций в секунду (статья на конф. Data Engineering 2014)

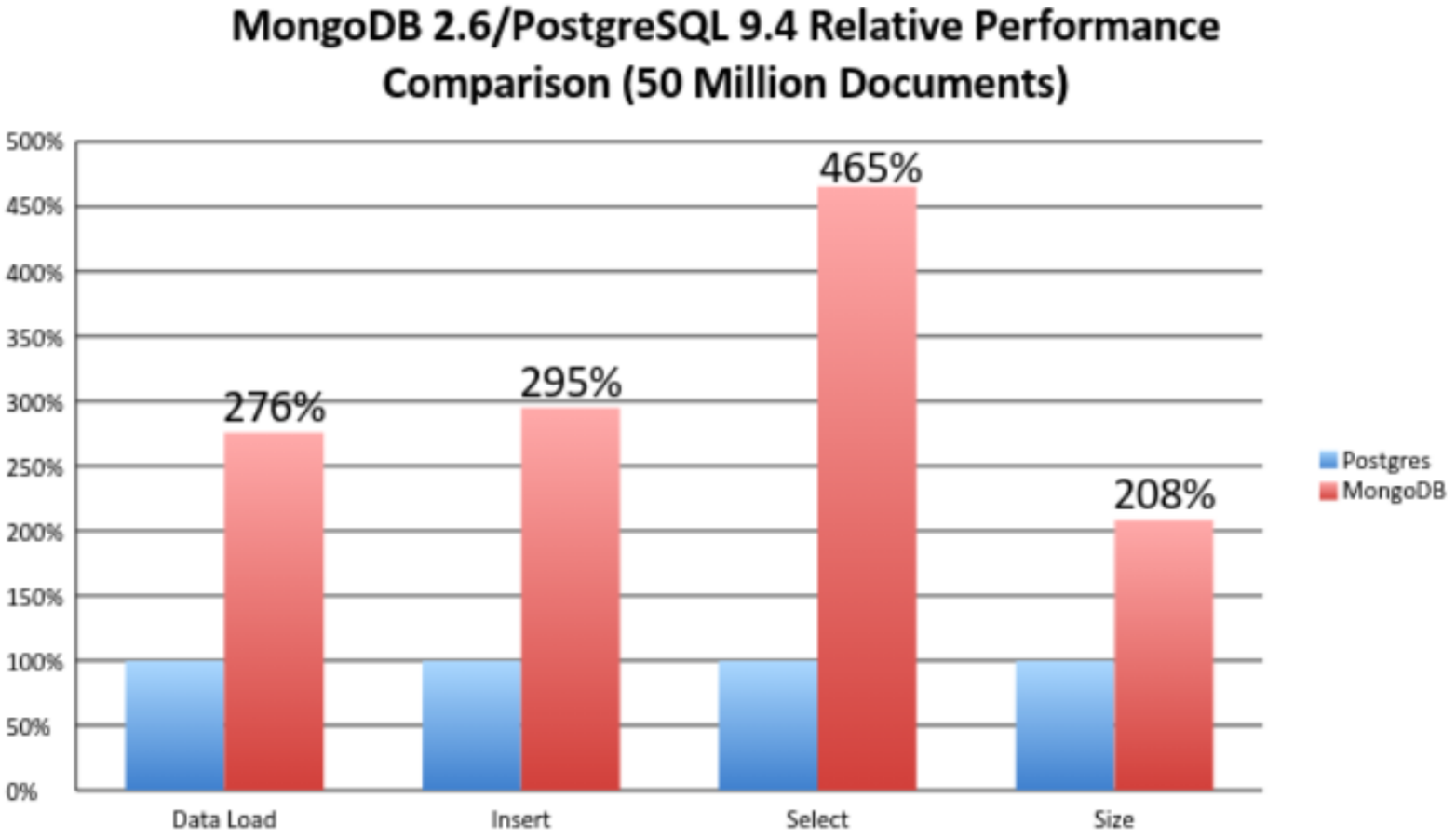
SQL ограничивает разнообразие типов данных, необходимых для интернет-приложений?

- Отказ от схемы никак не увеличивает разнообразие типов
- Базовые типы:
  - числа
  - текст
  - даты и время
  - ссылки (URL)
  - позиции (координаты и т. п.)
  - изображения
  - звук
  - видео
  - есть ли что-то еще?
- User-defined object types (Oracle, IBM DB/2, MS SQL, PostgreSQL)

SQL-системы не обеспечивают гибкость структур данных?

- Отсутствие схемы не создает гибкость
- Плохие решения: CLOB, BLOB не отличаются от решений в системах ключ-значение
- XML
  - PostgreSQL, Oracle, MS SQL, IBM DB2
- JSON

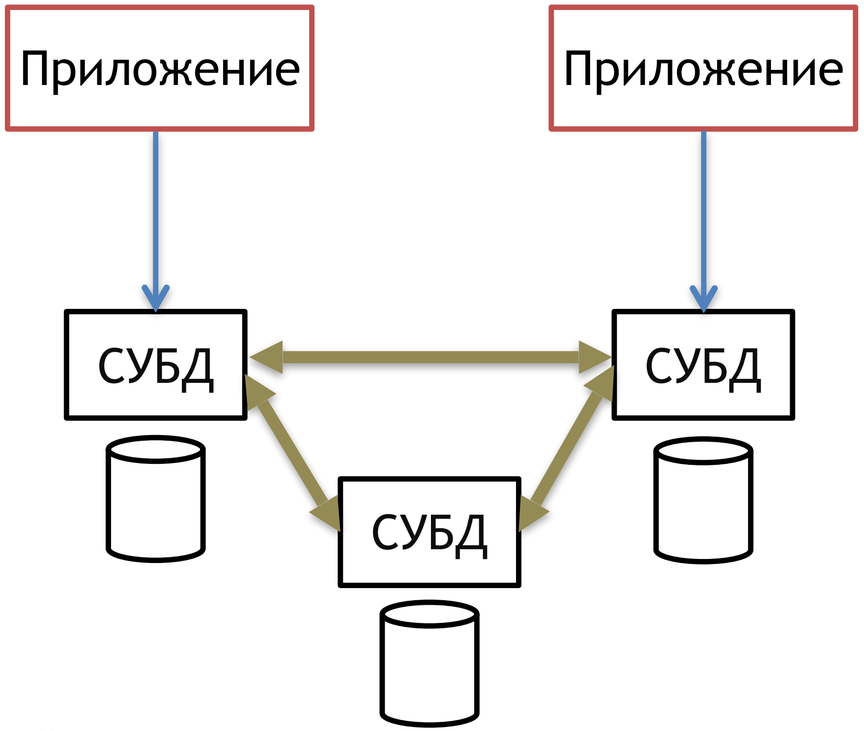
SQL-системы не могут обеспечить высокую производительность?



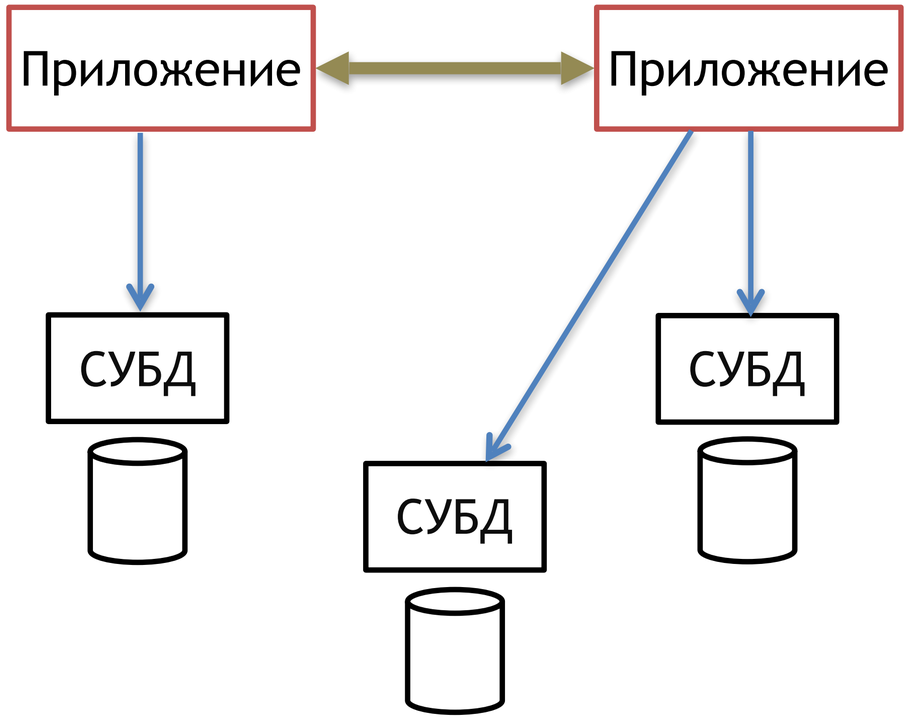
https://www.enterprisedb.com/node/3441

Распределенные системы

БД



Приложение



Распределенные системы

noSQL	SQL
Распределение данных определяется приложением	Распределенность прозрачна для приложения (partitioned tables)
Конфигурационный файл, обычно XML	Необходима конфигурация СУБД
Распределение по единственному ключу	Распределение по значениям любого атрибута
На основе hash	Hash, диапазоны, другие методы

Как обосновать аксиомы noSQL?

- Рассмотрим аксиому X
- Выберем какую-нибудь SQL-систему, например, mySQL
- Выбранная система подтверждает аксиому X
- Следовательно, **любая** SQL-система подтверждает аксиому X

Особенности noSQL

- Отсутствие стандартов
- Отсутствие явных моделей данных
- Возможность реализации любых функций СУБД в коде приложения

Как сделать БД медленной?

- **Проектирование на уровне объектов, а не множеств объектов**
- Гибкая модель данных (объект-атрибут-значение и т. п.)
- Предотвращение использования индексов
- Слишком много слишком мелких запросов
- Некорректное определение и использование представлений (view)

Воспоминание о будущем

- 1980+ Первые реляционные системы
- 1986–1995 Объектные БД
- 1997–2007 XML БД
- 2006–? noSenseQL

Многообразие: итоги

«Все системы хороши, выбирай на вкус»

Это еще не конец

Задания и инструкции по представлению результатов их выполнения находятся там же, где другие материалы по курсу

https://postgrespro.ru/education/university/dbtech