ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ

https://github.com/kontur-web-courses/db

ЧТО ТАКОЕ БАЗА ДАННЫХ?

- Правильно называть – СУБД
  В разговорной речи часто СУБД = БД (DB)
- БД – хранит данные, отдает/обновляет по запросу
- БД – это обычно сервис, доступный по сети
- БД – сложная внутри, простая в использовании

БД – ЭТО НЕ МАГИЯ

Солянка **алгоритмов и структур данных**

- Hash-таблицы
- Деревья поиска
- Бинарный поиск
- Чтение из файла по offset

и многое другое…

ИСПОЛЬЗОВАТЬ ПРОСТО, НО ЕСТЬ НЮАНСЫ

~~Администрирование БД~~ != **Использование БД**

КЛАССИФИКАЦИЯ БД

МНОГО РАЗНЫХ ЦЕЛЕЙ –
МНОГО РАЗЛИЧНЫХ БАЗ ДАННЫХ

ПО МАСШТАБАМ

- Распределенные
- Централизованные

ПО СРЕДЕ ХРАНЕНИЯ

- В оперативной памяти
- На жестких дисках

ПО ТИПУ ЗАДАЧ

1. Распределенный кеш (данные могут потеряться)
2. Полнотекстовый поиск
3. Хранилища под метрики (специальная математика)
4. Хранилища под географию (геометрия)
5. Графовые БД
6. In-memory хранилища (очень быстро, но очень дорого)
7. …

~~SQL~~ VS NOSQL

- Реляционные
- NoSQL:
  - Документные: MongoDB)
  - Колоночные: HBase, Cassandra
  - Ключ-значение: Amazon DynamoDB)

ТРЕБОВАНИЯ К БД

ATOMICITY

Нельзя прочитать/записать половину записи («мама мыла ра»)

DURABILITY

Данные не должны теряться после успешного сохранения

Как? Репликация, рейд дисков, operation log, …

CONSISTENCY

После успешного или неуспешного выполнения запроса данные должны быть согласованы

Если Фред переводит Барни 100$, то после:

— либо у Фреда -100$ и у Барни +100$

— либо у Фреда -0$ и у Барни +0$

ISOLATION

Параллельно выполняемые запросы не должны оказывать влияние на результат

Один запрос распределяет материальную помощь
в размере X студентам группы A,
другой запрос переводит одного из студентов в группу B

ISOLATION

Запрос 1: Распределение матпомощи X для группе A

```
var count = students.Count(s => s.Group == "A");
var sumPerStudent = X / count;
foreach (var s in students.Where(s => s.Group == "A"))
    s.Money += sumPerStudent;
```

Запрос 2: Перевод студента в группу B из группы A

```
students.SingleOrDefault(s => s.Id == Id)?.Group = "B";
```

ISOLATION

Параллельно выполняемые запросы не должны оказывать влияние на результат

Один запрос распределяет материальную помощь
в размере X студентам группы A,
другой запрос переводит одного из студентов в группу B

Сложно хорошо изолировать запросы, каждая СУБД гарантирует некоторый уровень изоляции

ACID

Atomicity

Consistency

Isolation

Durability

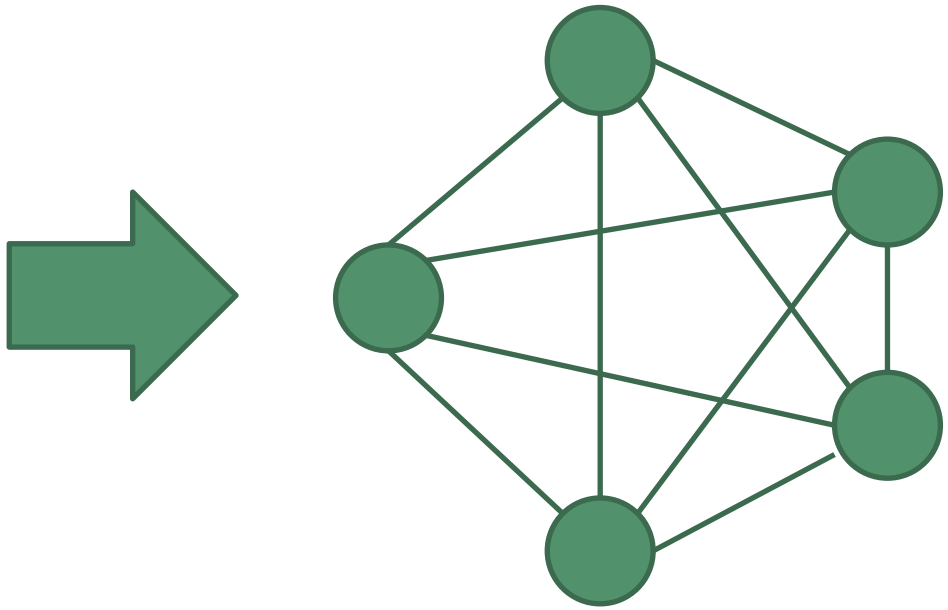
Желаемый набор требований
Сложно обеспечить, особенно в распределенных БД

AVAILABILITY

БД должна быть доступна 99.(9)% времени

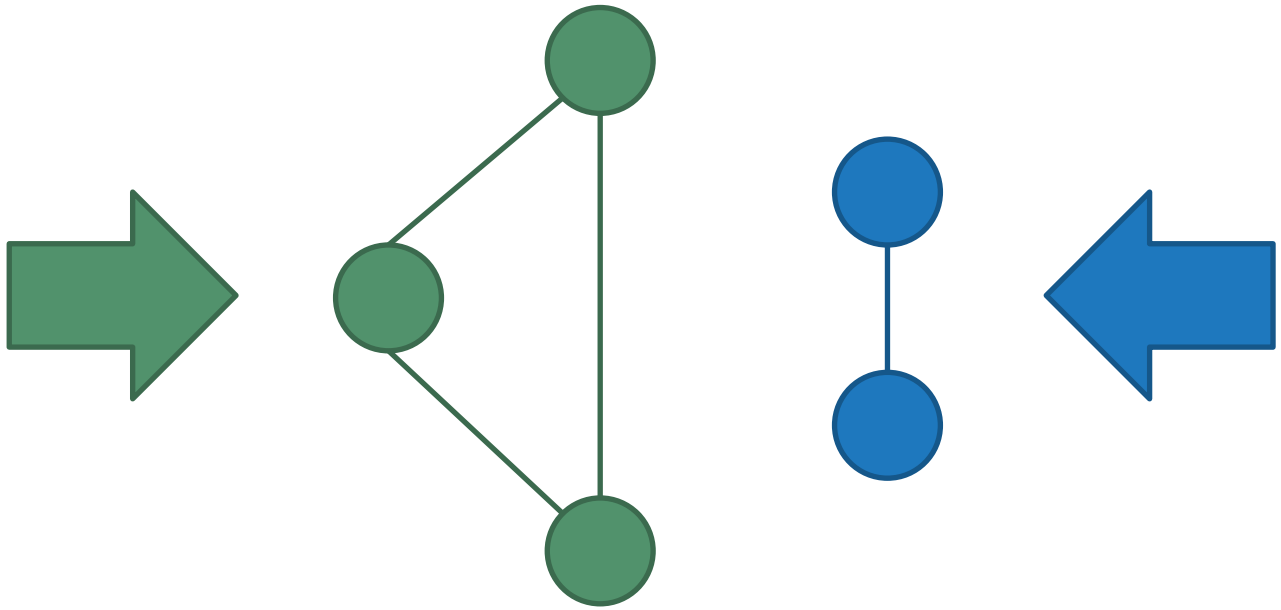
PARTITION TOLERANCE

Распределенная БД должна быть устойчива к brain-split



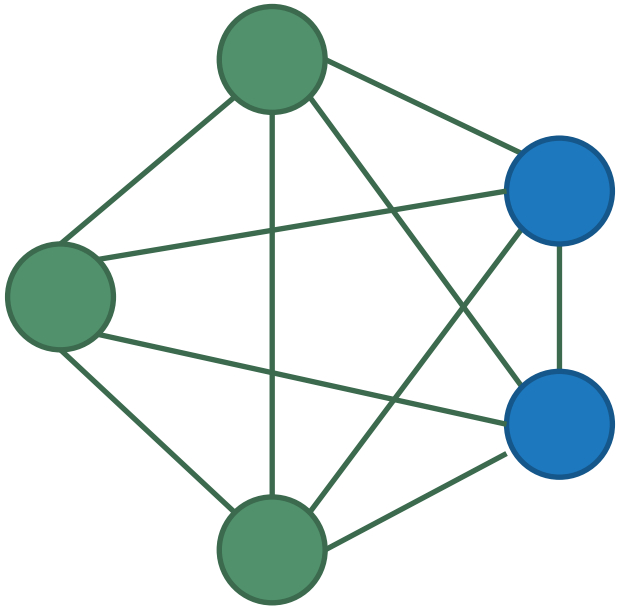
PARTITION TOLERANCE

Распределенная БД должна быть устойчива к brain-split



PARTITION TOLERANCE

Распределенная БД должна быть устойчива к brain-split



CAP-ТЕОРЕМА БРЮЕРА

В распределенной системе **невозможно** обеспечить одновременное выполнение:

- **Consistency** (Целостности)
- **Availability** (Доступности)
- **Partition Tolerance** (Устойчивости к сбоям узлов)

ДОКАЗАНО МАТЕМАТИКАМИ

ПРИМЕР C+A

Система рассчитывает, что сеть надежна, либо не распределенная

ПРИМЕР C+PT

Система с несколькими мастер-базами, которые обновляются синхронно

Всегда доступна на чтение, но запрещает запись при разрывах сети

ПРИМЕР A+PT

Система с несколькими серверами, каждый из которых может принимать данные, но не обязуется в тот же момент распространять их по всему кластеру

Система переживает падения части серверов, но когда они входят в строй, они будут выдавать старые данные

ОБХОД CAP-ТЕОРЕМЫ

Теорема доказана с конкретными формулировками понятий C, A и PT

Можно попытаться ослабить формулировки, получив что-то жизнеспособное

BASE

Брюер предложил оказаться от Consistency,
но мягко:

Basically **A**vailable

Soft state

Eventual consistency

BASE

Basically **A**vailable

= Availability в CAP

Soft state

состояние меняется даже без внешних воздействий, чтобы прийти к согласованности

Eventual consistency

реплики сходятся к одинаковому состоянию и в конце концов станут согласованными

BASE BMECTO ACID

ТРЕБОВАНИЯ К БД

Обычно, самое важное:

1. Данные не должны теряться (ACID)
2. Данные должны быть согласованы (ACID, CAP, BASE)
3. Устойчива к brain-split (CAP)

Как это достигается — за рамками этого блока

ТРЕБОВАНИЯ К БД

БД должна работать быстро

Это сильно зависит в том числе и от того, как ей пользоваться и как спроектировать данные в ней

Об этом далее!

ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БД

ТОЛЬКО ОСНОВЫ

О чем поговорим:

- Коллекции, поиск по индексам
- Примеры на БД «MongoDB»

MONGODB

- Документная
- Масштабируемая
- Гибкая

КОЛЛЕКЦИИ

Users:

- {*"Login":* "Ciceron", *"Role":* "Owner"}
- {*"Login":* "Popper", *"Role":* "Admin", *"BanHammer":* "true"}
- {*"Login":* "Freid", *"Role":* "User", *"Status":* "Ban"}
- {*"Login":* "ImmanuelKant", *"Role":* "User"}
- ...

Messages:

- {*"Author":* "Freid", *"Text":* "Hi all", *"Timestamp":* "2019-02-21"}

КАК ХРАНИТЬ ДОКУМЕНТЫ?

А как их искать?

Найти пользователя с заданным логином

А быстро?

ПОИСК ПО ТОЧНОМУ СОВПАДЕНИЮ

Рядом с файлом данных коллекции хранить **HashTable**:

Login → смещение в файле данных, по которому записан пользователь с таким Login

ПОИСК ПО ДИАПАЗОНУ

Найти все сообщения с января по февраль

Любая ordered структура

ИНДЕКСЫ

- Может быть много индексов у одной коллекции
- Занимает дополнительное место
- Замедляет операции обновления данных
- Создаются под запросы, которые придётся выполнять часто

ЗАДАЧА «ФОРУМ»

Пользователь ввел логин и пароль, нужно его аутентифицировать

ForumDB:

- Users:
  - Index on Login

ЗАДАЧА «ФОРУМ»

Пользователь прошел по ссылке на конкретное сообщение, нужно его отобразить

ForumDB:

- Users:
  - Index on Login
- Messages:
  - Index on MessageId

ЗАДАЧА «ФОРУМ»

Показать список самых популярных сообщений

ForumDB:

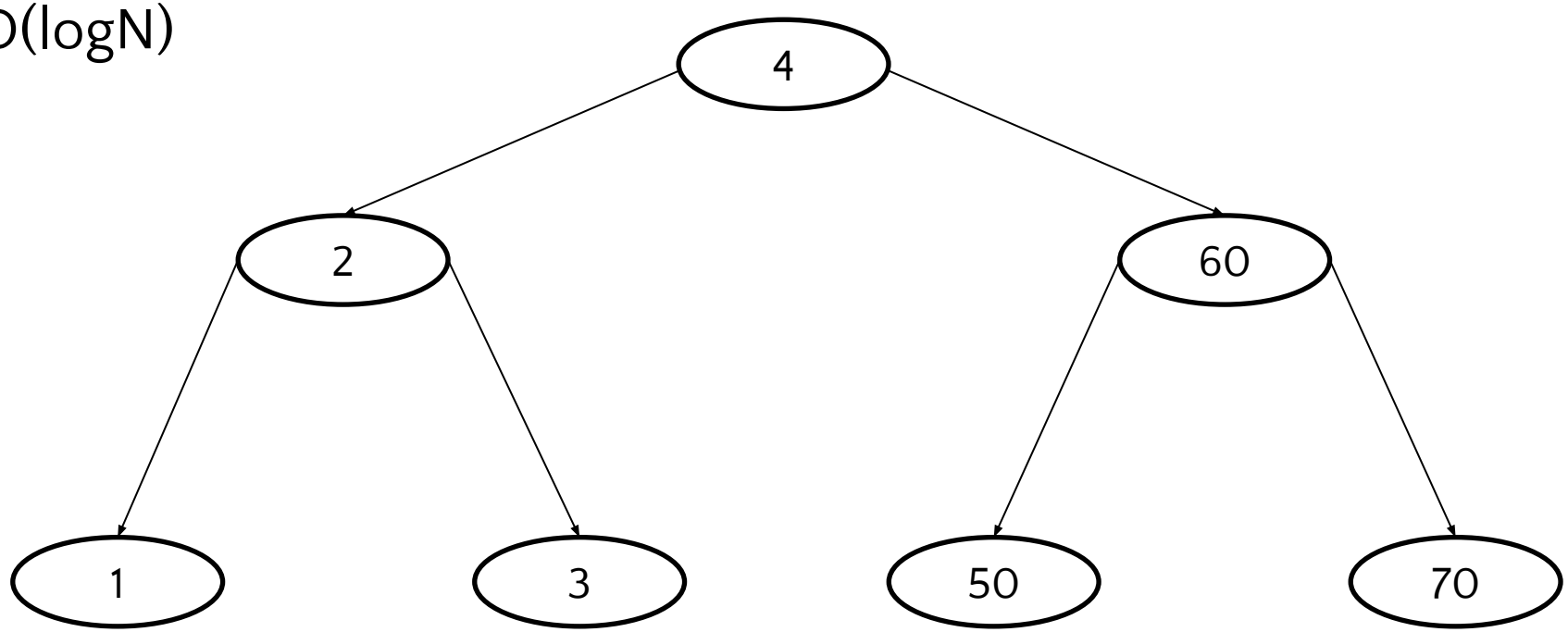
- Users:
  - Unordered index on Login
- Messages:
  - Unordered index on MessageId
  - Ordered index on LikesCount

ORDERED INDEX ≈ СОРТИРОВАННЫЙ СПИСОК

1. 1
2. 2
3. 3
4. 4
5. 50
6. 60
7. 70

ORDERED INDEX — ОБЫЧНО ДЕРЕВО

- Задан порядок (collation)
- Поиск левой / правой границы O(logN)
- Переход к предыдущему/следующему в среднем O(1)
- Поиск i-ого O(logN)

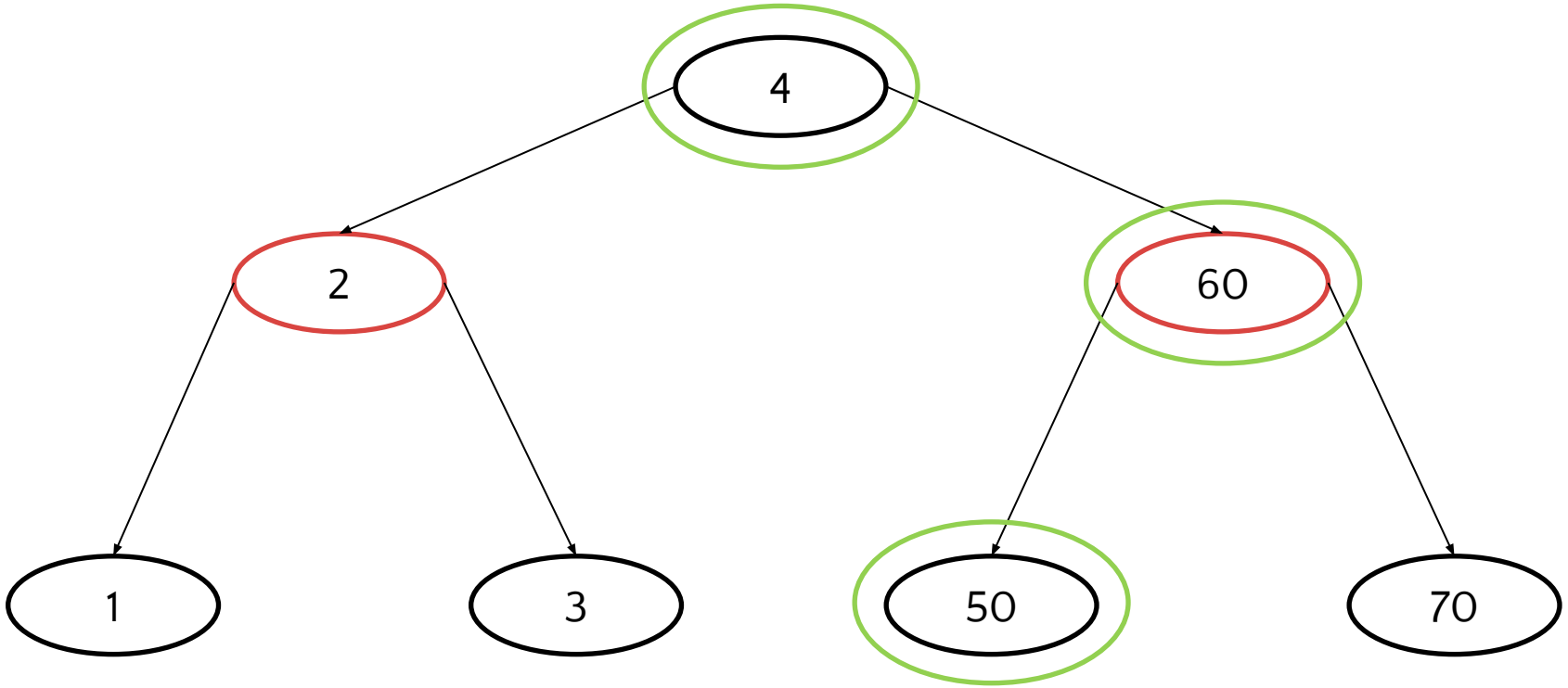


SKIP/TAKE

1. 1
2. 2
3. 3
4. **4**
5. **50**
6. **60**
7. 70

skip 1, take 3, с конца

SKIP/TAKE

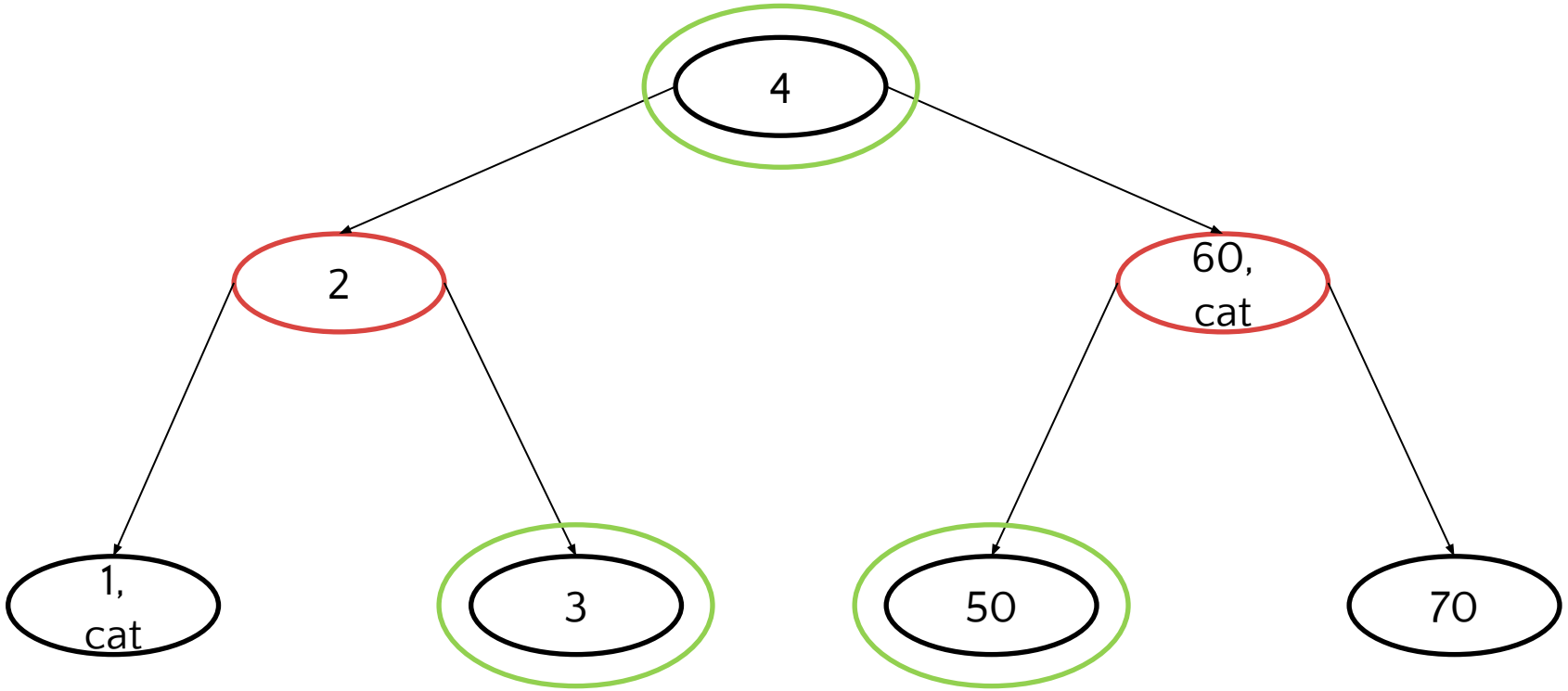


skip 1, take 3, с конца

ФИЛЬТРАЦИЯ

1. ~~1, cat~~
2. 2
3. **3**
4. **4**
5. **50**
6. ~~60, cat~~
7. 70

ФИЛЬТРАЦИЯ



ORDERED INDEX + ФИЛЬТРАЦИЯ

- Топ M сообщений = взять первое и M раз перейти к следующему: O(M + logN)
- Топ M без картинок = взять первое и, пропуская картинки, переходить к следующему, пока не наберем M в результат: O(M + K + logN),
  K – количество сообщений с картинками
  в первых M + K сообщениях.
  Если мы знаем, что K мало, то хорошо

ЗАДАЧА «ФОРУМ»

Показать последние T сообщений из заданного треда обсуждений

- Тредов много
- Могут обратиться как к старому треду, так и к новому

СОСТАВНЫЕ ИНДЕКСЫ

TopicId, PublicationDate

1, 2019-02-10

1, 2019-02-15

6, 2019-01-01

6, 2019-02-02

7, 2019-02-18

9, 2019-01-05

Найти последние сообщения в топике 6:

Найти левую границу (6, Date.MaxValue), и взять M предыдущих значений, пока TopicId=6

СОСТАВНОЙ ИНДЕКС

ForumDB:

- Users:
  - Unordered index on Login
- Messages:
  - Unordered index on MessageId
  - Ordered index on Likes
  - **Ordered index on TopicId, PublicationDate**

ВЫВОДЫ ПО ПРОЕКТИРОВАНИЮ

- Стратегия поиска в БД
- Стратегия проектирования структуры БД
- Стратегия выбора БД

СТРАТЕГИЯ ПОИСКА В БД

1. Максимально сильно сузить выборку
   с помощью поиска по индексу до малого числа документов
2. Отфильтровать оставшееся

Идеально, если поиски будут происходить
по точечному, известному ключу

СТРАТЕГИЯ ПРОЕКТИРОВАНИЯ СТРУКТУРЫ БД

1. Заранее выяснить какие запросы БД должна уметь обрабатывать эффективно
2. Понять, какие будут коллекции
3. Спланировать, где нужны индексы
4. А где можно просто отфильтровать, опираясь на знание природы данных и сэкономить на индексах

СТРАТЕГИЯ ВЫБОРА БД

- Понимать специфику своих потребностей
- Понимать ограничения и сильные стороны разных СУБД
- Возможно, даже использовать несколько СУБД в одном проекте
- Это умение приходит с опытом и кругозором

- *Не поддавайтесь хайпу*
- *Вы не Google*

ПРАКТИКА ПРОЕКТИРОВАНИЯ БД

СЕРВИС ДЛЯ ОТЕЛЕЙ

```
public interface IHotelRepository
{
    HotelId AddHotel(string name);
    void RemoveHotel(HotelId hotelId);
    RoomId AddRoom(HotelId hotelId, RoomDescription roomDescription);
    void RemoveRoom(RoomId roomId);
    void RentRoom(RoomId roomId, DateTime from, DateTime to, Guest[] guests);
    Guest[] GetArrivedGuests(DateTime day);
    (From, To, Guest[])[] GetRoomSchedule(
        RoomId roomId, DateTime from, DateTime to);
    RoomDescription[] GetFreeRooms(HotelId hotelId, DateTime from, DateTime to);
}
```

Проектировать будем тут: http://bit.ly/db-shpora

КАК ПРОЕКТИРОВАТЬ БД?

- Фиксируем результат проектирования в документе
- Описываем коллекцию как набор полей
- Если над полем надо построить индекс, явно пишем об этом в колонке атрибутов поля
- Указываем, какой индекс: ordered/unordered
- Отмечаем поле для первичного ключа
- Запросы описываем словами
  Коротко и ясно, как и где происходит запрос

КОЛЛЕКЦИЯ КОМНАТ

- Нам понадобится коллекция «отели»
- Пусть у каждого отеля будет уникальный индекс
- Флаг «удалено»
- Где хранить комнаты?
- Сделаем еще коллекцию «комнаты»
- Флаг «доступности» комнаты, ведь комната может быть закрыта на ремонт

ПРИМЕР: ДОБАВЛЕНИЕ КОМНАТЫ

```
RoomId AddRoom(
   HotelId hotelId,
   RoomDescription roomDescription);
```

- Добавить комнату в коллекцию «комнаты»
- Привязать комнату к отелю

ЗАДАЧА: БРОНИРОВАНИЕ КОМНАТ

```
void RentRoom(
   RoomId roomId,
   DateTime from,
   DateTime to,
   Guest[] guests);
```

http://bit.ly/db-shpora

Скопируйте лист преподавателя
и переименуйте его вашими фамилиями

ЗАДАЧА: ОТЧЕТНОСТЬ В МВД

```
Guest[] GetArrivedGuests(DateTime day)
```

ПРОЕКЦИИ

Можно попросить СУБД отдать не весь документ, а только отдельные его поля = проекцию

ЗАДАЧА: ИСТОРИЯ КОМНАТЫ

```
(From, To, Guest[])[] GetRoomSchedule(
  RoomId roomId,
  DateTime from,
  DateTime to);
```

Запрос должен работать быстро!

СОСТАВНОЙ ИНДЕКС VS ДВА ИНДЕКСА

Можно ли одновременно искать с условиями на From и на To?

С нашей моделью упорядоченного индекса НЕТ!

Но можно изобрести структуру данных для индекса, которая сможет. Например, K-d дерево, R-дерево, квадродерево.

Только, скорее всего, в СУБД она не реализована

ЗАДАЧА: СВОБОДНЫЕ КОМНАТЫ

```
RoomDescription[] GetFreeRooms(
  HotelId hotelId,
  DateTime from,
  DateTime to);
```

НУЖНО ЛИ ОБА СОСТАВНЫХ?

(RoomId, To)

(HotelId, To)

А нельзя ли вместо двух, иметь один такой?

(HotelId, RoomId, To)

НУЖНЫ ОБА СОСТАВНЫХ!

HotelId, RoomId, To

1, 100, 2019-01-10

2, 200, 2019-01-02

2, 200, 2019-01-05

2, 201, 2019-01-03

...

2, 299, 2019-01-10

3, 300, 2019-01-01

`(HotelId, RoomId, To)` заменит `(RoomId, To)`

`(HotelId, RoomId, To)` НЕ заменит `(HotelId, To)`

ХРАНЕНИЕ СВЯЗИ «ОДИН-МНОГО»

Один «родитель», много «детей»

- Полностью в родителе
  - Guests в Rent
- Идентификаторы в родителе
  - RoomIds в Hotel
- Идентификатор родителя в ребенке
  - RoomId в Rents

НОРМАЛИЗАЦИЯ

В реляционных БД принято «нормализовать»,
хранить связи в отдельной таблице
с записями вида **(Id, ParentId, ChildId)**

Предыдущие подходы к хранения
связи «Один-Много» — «денормализация»

ОТЛИЧИЯ РЕЛЯЦИОННЫХ БД

MONGO VS SQL

- Хранит коллекцию документов
- Структура документов не фиксирована

- Табличная структура
- Поля заранее зафиксированы, изменение их состава – отдельная процедура

MONGO VS SQL

- Документы могут иметь произвольную вложенность
- Денормализация для уменьшения числа запросов
- Простые клиенты, простая конвертация объектов в BSON и обратно
- В ячейках таблицы примитивные значения
- Принято нормализировать таблицы
- Умные ORM, собирающие нормализованные данные обратно в объекты

MONGO VS SQL

- uniq – индексы в рамках шарда
- Распределенная система

- Транзакции, триггеры, ограничения, каскадные операции и прочее, завязанное на локальность
- Локальная система

ПРАКТИКА ИСПОЛЬЗОВАНИЯ БД

ЗАДАЧА: GAME

- 2 игрока
- Несколько туров
- Уже реализована
- Хранит всё в памяти

СЦЕНАРИИ

- Зарегистрировать нового пользователя
- Отредактировать / удалить пользователя
- Создать планируемую игру
- Добавить пользователя в игру
- Начать / закончить игру
- Сделать пользователем ход в игре
- Показать текущую информацию по игре
- Заново проиграть историю игры

СУЩНОСТИ

Пользователь:

логин, имя, статистика, …

Игра:

игроки, статус, номер тура, текущий счёт, …

Тур:

в какой игре, номер тура,
кто как ходил, кто выиграл

DEMO GAME

- Game – реализация предметной области
- Tests – тесты
- ConsoleApp – реализация логики приложения

DEMO MONGODB COMPASS

- Создание БД
- Создание документа
- Поиск документа
- Индексы

ФОРМУЛИРОВКА ЗАДАНИЯ

https://github.com/kontur-web-courses/db/blob/main/README.md

БОНУС-ИДЕИ

1. Bson-based язык запросов MongoDB
2. Специализированные методы репозиториев
3. Отвязывание сущностей БД от классов, реализующих логику предметной области (Automapper)

РЕШЕНИЯ

- Отели: http://bit.ly/db-shpora-solved
- Камень-ножницы-бумага, в ветке **solved**