



# HBase

Михаил Денисенко  
Разработчик Яндекс

Презентация: Ивченко Олег, 094а

Москва, 2015 (Май)

# План лекции

- Введение – отличия от РСУБД
- Устройство HBase
- Примеры использования
- Общие советы по работе с HBase

# Отличия от РСУБД

- Для больших объёмов данных
- Работает поверх HDFS
  - распределённость
  - отказоустойчивость
- Колоночно ориентированная
  - колонки можно добавлять не меняя схему БД
  - произвольный доступ к ячейкам таблицы
- NoSQL-хранилище
  - Более скудный набор команд (Put, Get, Scan), нет Join
- Нет типов данных. Всё хранится в байтах
- Можно использовать MapReduce
  - Специальные классы `TableMapper` и `TableReducer`

# План лекции

- Введение – отличия от РСУБД
- **Устройство HBase**
- Примеры использования
- Общие советы по работе с HBase

# Команды HBase

- Put
  - Вставка данных
  - Можно вставлять batch, что экономит время

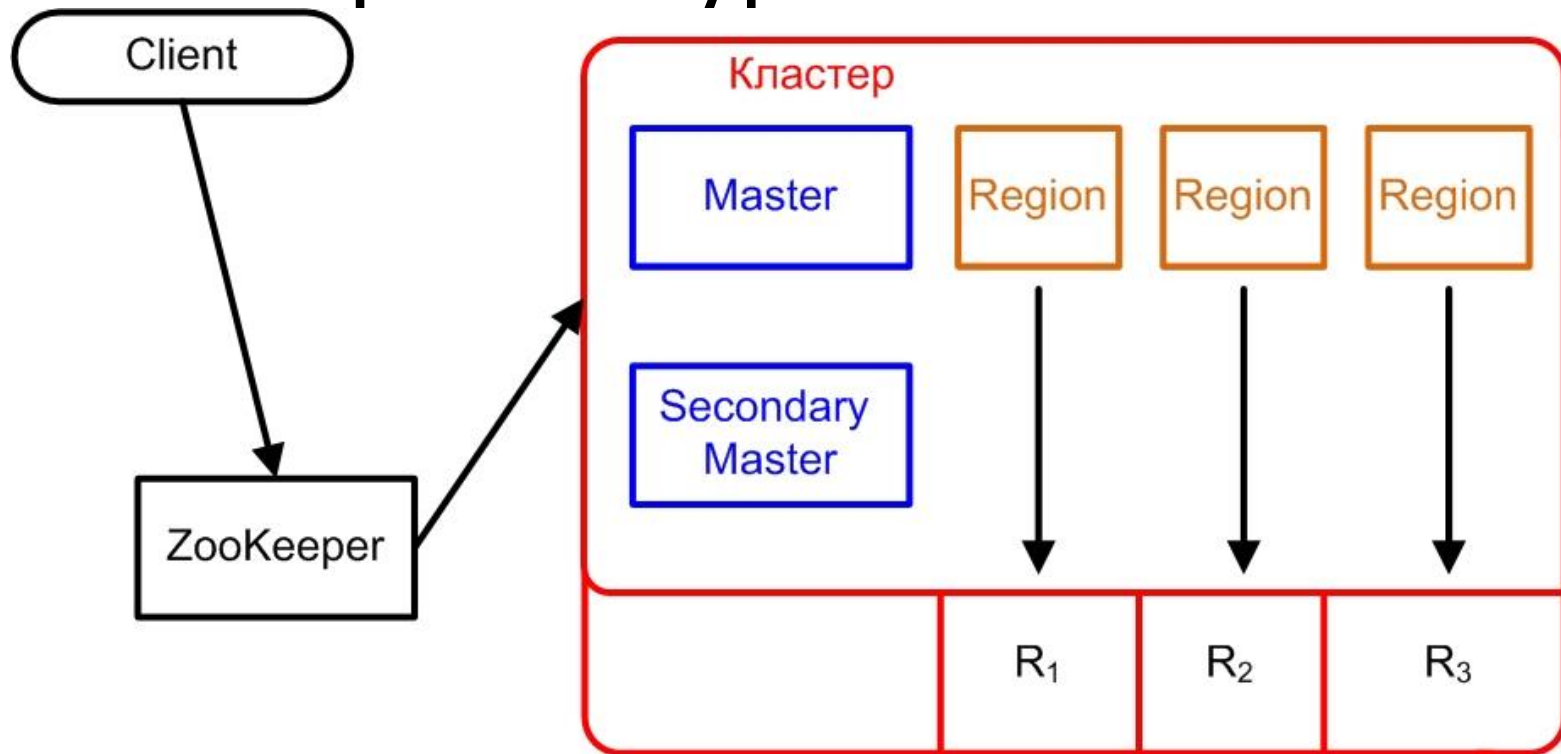
# Команды HBase

- Put
  - Вставка данных
  - Можно вставлять batch, что экономит время
- Get
  - Получение данных

# Команды HBase

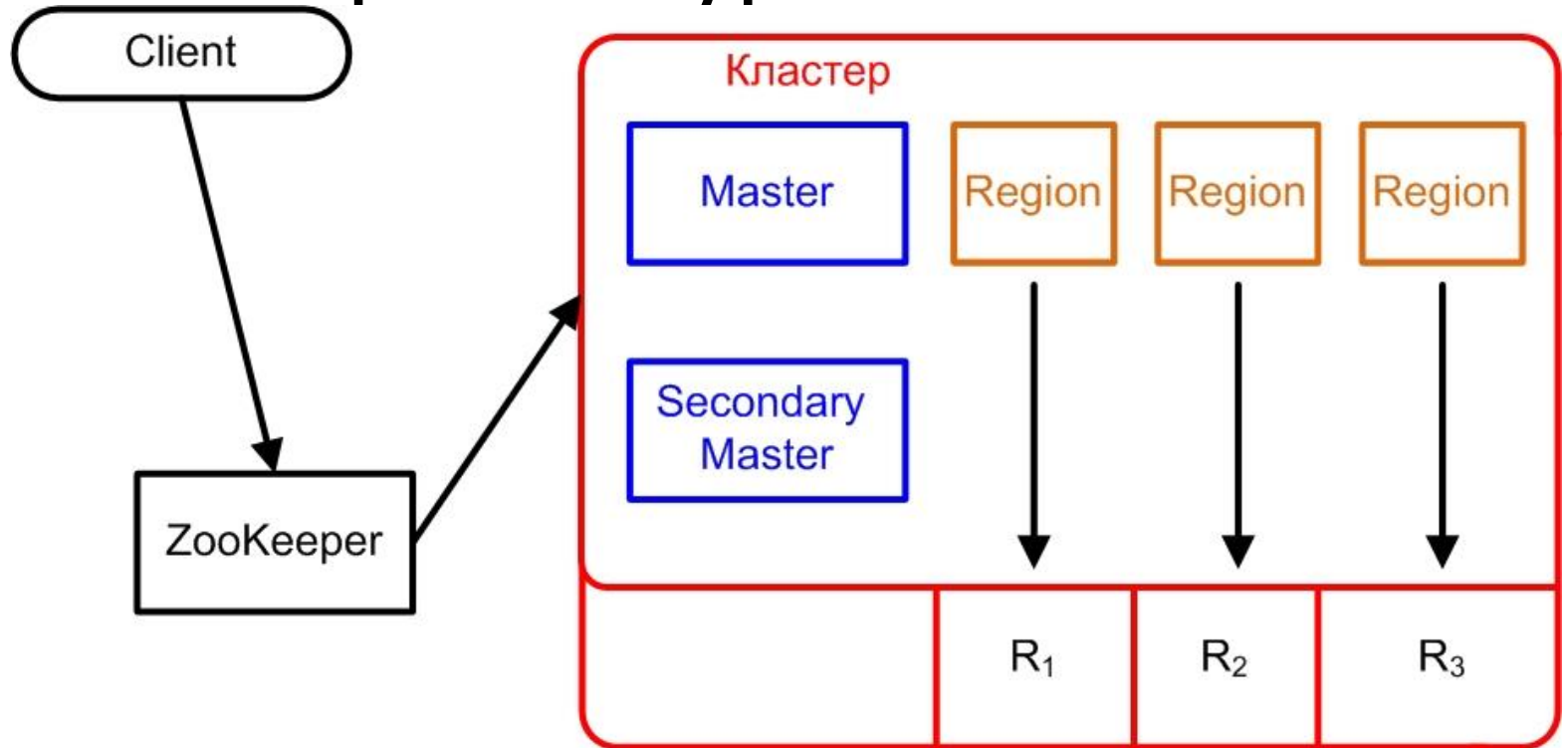
- Put
  - Вставка данных
  - Можно вставлять batch, что экономит время
- Get
  - Получение данных
- Scan
  - Итерируется по некоторому диапазону строк
  - Может выполняться очень долго
  - Указываются:
    - границы области сканирования. Если одна (или 2) границы не указаны, берутся крайние строки
    - условие (фильтр) на колонки

# Архитектура HBase



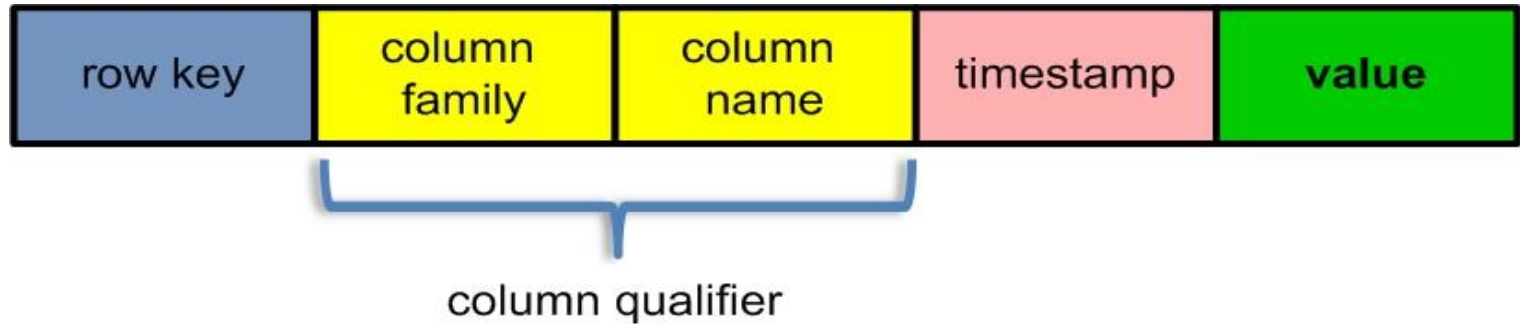


# Архитектура HBase



- Master-сервер – отвечает за распределение данных по регион-серверам
- На случай поломки Master иногда присутствует Secondary Master
- ZooKeeper – кластер из **нечётного** числа машин
  - ZooKeeper Quorum

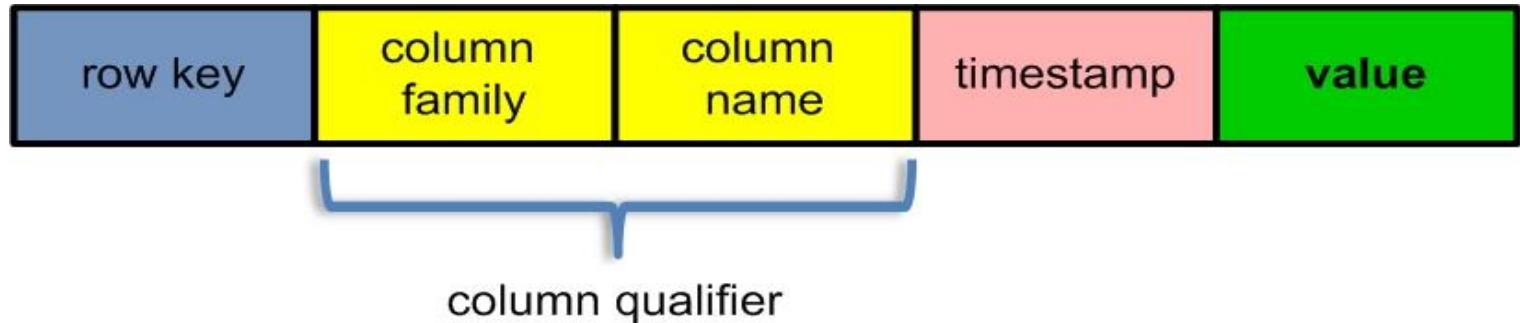
# Структура строки HBase



## — Первичный ключ (row key)

- Данные в таблице сортируются по row key в алфавитном порядке

# Структура строки HBase



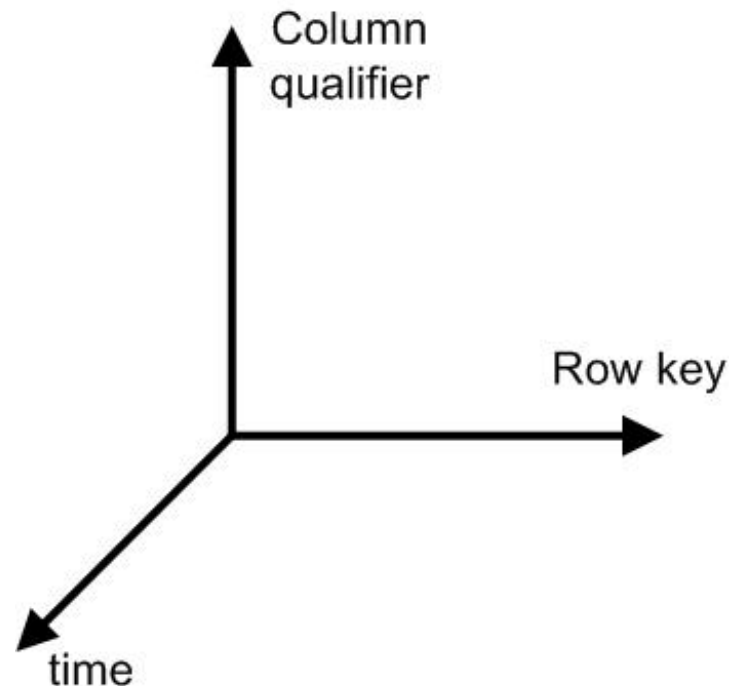
- Первичный ключ (row key)
  - Данные в таблице сортируются по row key в алфавитном порядке
- Семейство колонок (column family)
  - Нельзя добавлять и удалять после создания таблицы
  - Каждое семейство хранится в отдельном файле
  - Семейств должно быть не больше 10 (обычно 3-5)

# Структура строки HBase

- Название колонки (column name)
  - Можно добавлять и удалять не меняя схему
  - Может быть очень много
  - Приспособлены для хранения разряжённых данных (пустые колонки память не занимают)

# Структура строки HBase

- Название колонки (column name)
  - Можно добавлять и удалять не меняя схему
  - Может быть очень много
  - Приспособлены для хранения разряжённых данных (пустые колонки память не занимают)
- Timestamp позволяет однозначно определить ячейку

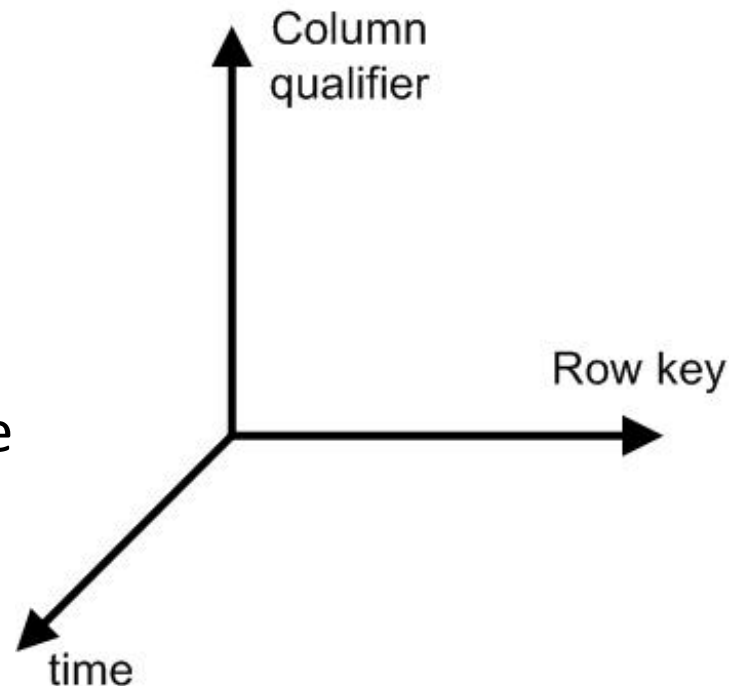


# Структура строки HBase

- Название колонки (column name)
  - Можно добавлять и удалять не меняя схему
  - Может быть очень много
  - Приспособлены для хранения разряжённых данных (пустые колонки память не занимают)
- Timestamp позволяет однозначно определить ячейку

Если timestamp не указан

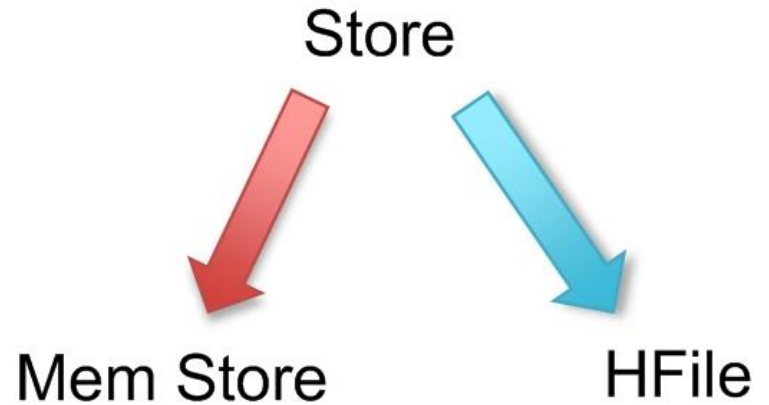
- Get возьмёт последнюю версию
- Put присвоит ячейке текущее время



# Хранение данных

## — Компактификации

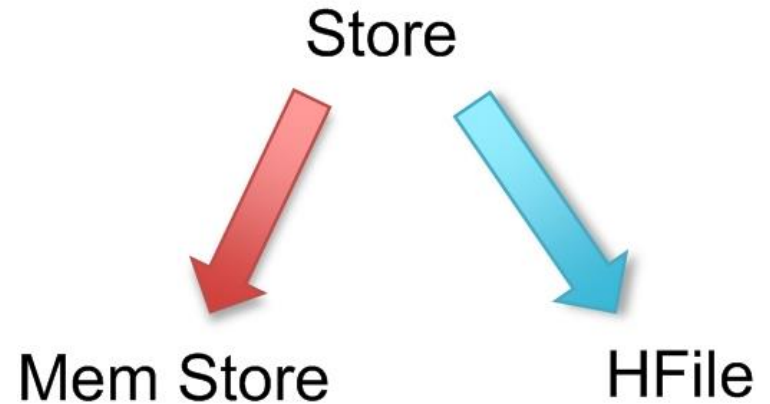
- **Minor** compaction –  
сохранение MemStore в  
Hfile
- **Major** compaction –  
слияние Hfile



# Хранение данных

## — Компактификации

- **Minor** compaction –  
сохранение MemStore в  
Hfile
- **Major** compaction –  
слияние Hfile



## — Каждая операция записывается в Write-ahead log



# Bloom filter

- Ускоряет работу с диском
- Для каждого HFile берётся  $M$  бит  $n_1, \dots, n_m$  и  $k$  функций хеширования  $h_1, \dots, h_k$
- Если  $h_i = n_j$ , то на позицию ставится 1
- Действия при запросе **Get**:
  - Данные ищутся в MemStore.
  - Если их там нет, вычисляем Bloom-фильтр и сравниваем его с Bloom-фильтрами для каждого HFile.
  - Если фильтры не совпадают хотя бы **по 1 биту**, в данном файле нужной информации нет

# Bloom filter

- Ускоряет работу с диском
- Для каждого HFile берётся  $M$  бит  $n_1, \dots, n_m$  и  $k$  функций хеширования  $h_1, \dots, h_k$
- Если  $h_i = n_j$ , то на позицию ставится 1
- Особенности:
  - Фильтр с гарантией 100% отлавливает отсутствие данных
  - В случае положительного ответа, их присутствие не гарантируется
  - Чем больше  $M$ , тем выше точность
  - Если данных не найдено в MemStore, считывание с диска занимает время (до 10с)

# План лекции

- Введение – отличия от РСУБД
- Устройство HBase
- **Примеры использования**
- Общие советы по работе с HBase

# Применение HBase

– **Пример №1:** хранение web-страниц

Row key: название страницы

Семейства колонок:

1. контент
2. свойства (булевы значения): спам / неспам, рекламная, форум, ...
3. статистика: посещаемость, PageRank, ...

# Применение HBase

## – Особенности:

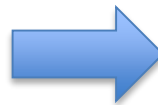
- в строке может быть много колонок, но многие могут быть пусты
- нужно будет добавлять новые колонки
- Сортировка по row key => страницы одного и того же сайта будут находиться рядом

[www.market.yandex.ru](http://www.market.yandex.ru)

[www.i.ua](http://www.i.ua)

[www.google.com](http://www.google.com)

[www.disk.yandex.ru](http://www.disk.yandex.ru)



[ru.yandex.market.www](http://ru.yandex.market.www)

[ua.i.www](http://ua.i.www)

[com.google.www](http://com.google.www)

[ru.yandex.disk.www](http://ru.yandex.disk.www)

[ru.yandex.disk.www](http://ru.yandex.disk.www)

[ru.yandex.market.www](http://ru.yandex.market.www)

[ua.i.www](http://ua.i.www)

[com.google.www](http://com.google.www)



# Применение HBase

– **Пример №2:** социальные сети:

- FaceBook – до  $10^9$  пользователей
- Twitter - меньше

# Применение HBase

## – Пример №2: социальные сети:

- FaceBook – до  $10^9$  пользователей
- Twitter - меньше

## – Данные: пользователи

$F_a \rightarrow F_{a1}, \dots, F_{a2} // F_a$  подписан на  $F_{a1}, \dots, F_{a2}$

$F_b \rightarrow F_{b1}, \dots, F_{b2}$

$F_c \rightarrow \dots$

# Применение HBase

## – Пример №2: социальные сети:

- FaceBook – до  $10^9$  пользователей
- Twitter - меньше

## – Данные: пользователи

$F_a \rightarrow F_{a1}, \dots, F_{a2} // F_a$  подписан на  $F_{a1}, \dots, F_{a2}$

$F_b \rightarrow F_{b1}, \dots, F_{b2}$

$F_c \rightarrow \dots$

## – Требования:

1. получить список **подписок** пользователя
2. проверить, подписан ли  $F_i$  на  $F_j$
3. получить список **подписчиков** пользователя



# Применение HBase

– Примитивное решение:

col_fam = followers					
$F_a$	1: $F_{a1}$	2: $F_{a2}$	3: $F_{a3}$	...	n: $F_{an}$

– Недостатки?

# Применение HBase

– Примитивное решение:

col_fam = followers					
$F_a$	1: $F_{a1}$	2: $F_{a2}$	3: $F_{a3}$	...	n: $F_{an}$

– Недостатки

- при добавлении (удалении) пользователем нового человека, нужно линейно просматривать всю строку

– Решение?

# Применение HBase

## — Улучшенное решение:

col_fam = followers					
$F_a$	$F_{a1:k}$	$F_{a2:k}$	$F_{a3:k}$	...	$F_{an:k}$

- в качестве имён столбцов берём id пользователя
- $k$  – любое значение (если  $F_a$  подписан на данного пользователя)
- теперь при добавлении (удалении) нового человека, обращаемся к конкретной колонке

## — Недостатки?

# Применение HBase

## — Улучшенное решение:

col_fam = followers					
$F_a$	$F_{a1:k}$	$F_{a2:k}$	$F_{a3:k}$	...	$F_{an:k}$

- в качестве имён столбцов берём id пользователя
- $k$  – любое значение (если  $F_a$  подписан на данного пользователя)
- теперь при добавлении (удалении) нового человека, обращаемся к конкретной колонке

## — Недостатки

- Для получения подписчиков  $F_i$  нужно делать линейный просмотр всей таблицы по столбцу  $F_i$

## — Решение?

# Применение HBase

## — Оптимизированное решение

- ключ – пара  $F_i F_j$  ( $F_i$  подписан на  $F_j$ )
- $k$  – любое значение

## — Недостатки?

	CF <sub>1</sub>
F <sub>1</sub> F <sub>2</sub>	c:k
F <sub>1</sub> F <sub>3</sub>	c:k
...	...

# Применение HBase

## — Оптимизированное решение

- ключ – пара  $F_i F_j$  ( $F_i$  подписан на  $F_j$ )
- $k$  – любое значение

## — Недостатки

- слишком много строк:  $n \cdot (n - 1)$
- row key может быть слишком длинным

## — Решения?

	CF <sub>1</sub>
F <sub>1</sub> F <sub>2</sub>	c:k
F <sub>1</sub> F <sub>3</sub>	c:k
...	...

# Применение HBase

## — Оптимизированное решение

- ключ – пара  $F_i F_j$  ( $F_i$  подписан на  $F_j$ )
- $k$  – любое значение

## — Недостатки

- слишком много строк:  $n \cdot (n - 1)$
- row key может быть слишком длинным

## — Решения

- добавить симметричную таблицу: ключ -  $F_i$   
подписчик  $F_j$
- или добавить новое семейство колонок в старую схему
- Вместо пары  $F_i F_j$  брать их хеши:  
 $\text{hash}(F_i) . \text{concat}(\text{hash}(F_j))$

	CF <sub>1</sub>
F <sub>1</sub> F <sub>2</sub>	c:k
F <sub>1</sub> F <sub>3</sub>	c:k
...	...

# Применение HBase

## — Решения

- добавить симметричную таблицу:
  - ключ -  $F_i$  подписчик  $F_j$



# Применение HBase

## — Решения

- добавить симметричную таблицу:
  - ключ -  $F_i$  подписчик  $F_j$
- или добавить новое семейство колонок в старую схему

	CF <sub>1</sub>	CF <sub>2</sub>
F <sub>1</sub> F <sub>2</sub>	c:k	c:k
F <sub>1</sub> F <sub>3</sub>	c:k	c:k
...	...	...

# Применение HBase

## — Решения

- добавить симметричную таблицу:
  - ключ -  $F_i$  подписчик  $F_j$
- или добавить новое семейство колонок в старую схему
- вместо пары  $F_i F_j$  брать их хеши:  
`hash( $F_i$ ) . concat (hash( $F_j$ ))`

	CF <sub>1</sub>	CF <sub>2</sub>
F <sub>1</sub> F <sub>2</sub>	c:k	c:k
F <sub>1</sub> F <sub>3</sub>	c:k	c:k
...	...	...

# Применение HBase

## — Пример №3: поиск по изображениям

- поиск изображения, похожих на заданное
- поиск нечётких дубликатов

# Применение HBase

- **Пример №3:** поиск по изображениям
  - поиск изображения, похожих на заданное
  - поиск нечётких дубликатов
- Хранение данных в виде списка смежности

# План лекции

- Введение – отличия от РСУБД
- Устройство HBase
- Примеры использования
- Общие советы по работе с HBase

# Общие советы по работе с HBase

- Если в реляционных БД нужно стремиться к нормализации, то в HBase наоборот – к **денормализации**

# Общие советы по работе с HBase

- Если в реляционных БД нужно стремиться к нормализации, то в HBase наоборот – к **денормализации**
- **Row key** и **column family** должны быть компактными т.к. передаются по сети