БАЗЫ ДАННЫХ

Лекция 5

ПЛАН НА СЕГОДНЯ

- Завершение B-деревьев
- Hash-индексы
- BitMap-индексы
- Двумерные индексы

B/B+-ДЕРЕВЬЯ

Визуализация B-деревьев

ДЛИННЫЕ КЛЮЧИ

- Проблема произвольной длины
- Можно пытаться решать на уровне базы/индекса
- Например, хранить префикс фиксированной длины
- Придется чаще ходить в таблицу

ДЛИННЫЕ КЛЮЧИ

- Придется чаще ходить в таблицу
- Можно сглаживать Bloom-фильтром
- Или хранить продолжения длинных ключей в специальных страницах

ДЛИННЫЕ КЛЮЧИ

- Можно решать на уровне создания индекса
- И выбора ключей
- Не добавлять в индекс ключи типа 'Константинов Константин Константинович'
- Стоит задумать о производных полях для индексирования
- Пример: по названиям стран или регионов

ДУБЛИРУЮЩИЕСЯ КЛЮЧИ

- Приходится хранить несколько координат страниц при ключе
- Уменьшает вместимость страницы
- Может нарушать баланс в дереве

КОГДА B-ДЕРЕВЬЯ ПОЛЕЗНЫ

- Почти гарантированно: при точечном запросе
- С правильно сформулированным критерием
- Точечный - запрос, в котором ответ существенно короче, чем число страниц
- Правильность критерия - простое равенство/неравенство поля с константой

СЕЛЕКТИВНОСТЬ

- Совместное свойство запроса и данных
- Доля данных в таблице, подпадающих под запрос
- Чем ниже селективность, тем полезнее индекс
- При высокой полезность снижается
- С учетом издержек - индекс становится вредным

СЕЛЕКТИВНОСТЬ

- Возможна ситуация бесполезности индекса при низкой селективности
- Пример: в страницу помещаются 50 записей таблицы
- В таблице 10 000 страниц, т.е. 500 000 записей
- Делаем запрос с селективностью 0.02
- Что само по себе звучит неплохо

СЕЛЕКТИВНОСТЬ

- В ответ попадают 10000 записей
- Но выходит так, что они раскиданы примерно по записи на страницу
- Придется прочитать все страницы

СЕЛЕКТИВНОСТЬ

- Индекс поможет найти внутри
- Но это слабое утешение
- Решение: если такие запросы часты, то отсортировать таблицу
- Вывод: не факт, что индекс отменяет необходимость сортировки

ДРУГОЕ РЕШЕНИЕ

- К сортировке могут быть противопоказания
- Например, разные ключи требуют сортировки
- Подумать от денормализации
- Создавать другие таблицы - над теми же данными под другими ракурсами
- Оптимизировать их представление

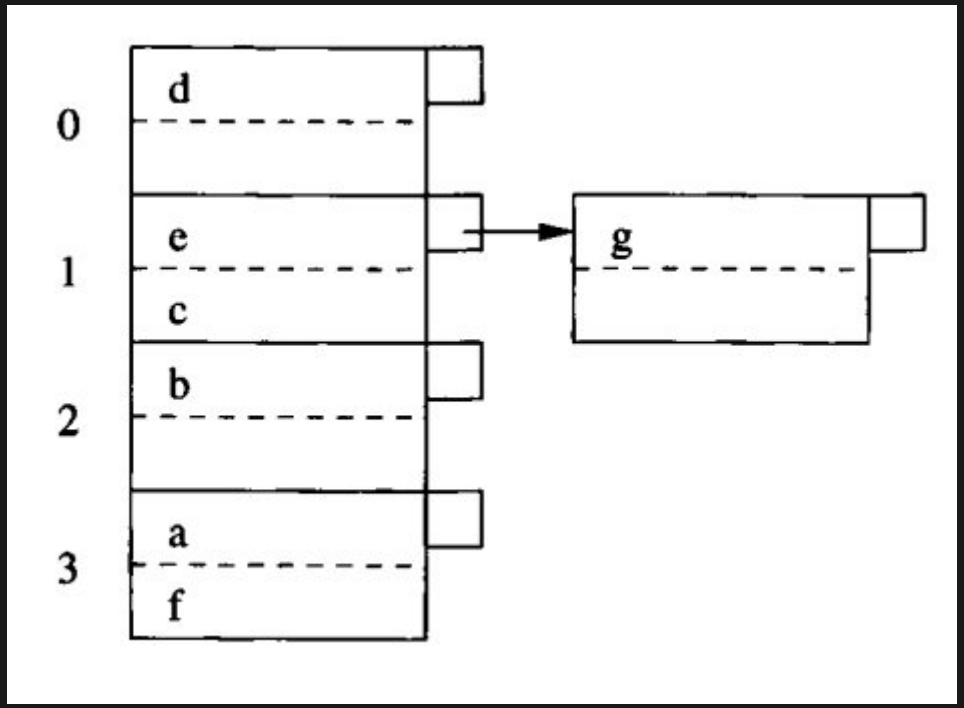
НА БОЛЬШИХ ЗАПРОСАХ

- Может помочь в планировании join/group/dedup
- Знанием о количестве ключей о количестве записей в ключах
- Но влияние на последующие шаги плана непростое
- Эффект первых шагов сложно предсказать

HASH-ИНДЕКСЫ

- Похоже на B-деревья
- Базовая идея из классических структур адатируется к внешней памяти
- Работаем страницами
- Каждому хеш-коду выделяем страницу
- Если не хватает - выделяем еще страницу

HASH-ИНДЕКСЫ



А ЧТО В СТРАНИЦАХ

- Вариант 1: ключ + координаты записи
- Минус: проблема длинных ключей
- Вариант 2: второй хеш (длинный) + координаты записи
- Длинный - для хеша, но маленький по сравнению с ключом

УДАЛЕНИЕ

- Находим элемент, удаляем в памяти
- Можем захотеть уплотнить страницы
- Например, при заполнении меньше, чем на половину
- Но не под завязку - чтобы не было "горячей точки" на создание-удаление

ПРОАНАЛИЗИРУЕМ

- Можем завести по 2 страницы на хеш-бакет
- Это могут быть сотни записей
- И несколько тысяч бакетов
- Это не уступает B-дереву по эффективности
- Но с неадекватным расходом памяти, пока таблица не выросла

"RE-HASH"

- Нужен какой-то аналог рехешинга
- Есть два подхода
  - Extensible hashing
  - Linear hashing

EXTENSIBLE HASHING

- Введем массив указателей на блоки
- Размер массива - степень двойки
- Сначала размер - 1
- Указывает на единственный блок

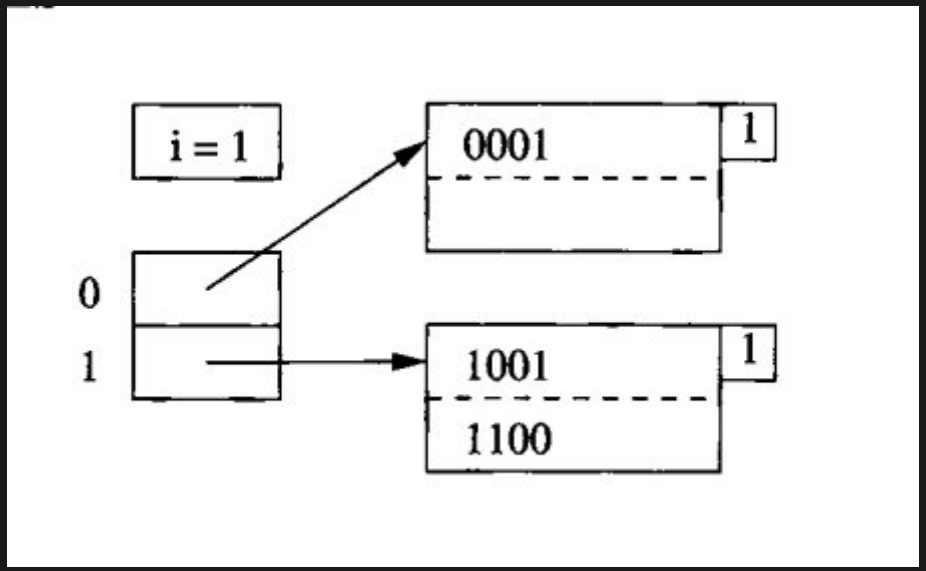
EXTENSIBLE HASHING

- По мере заполнения создаем новый блок
- Считаем хеши содержимого
- У кого заканчивается на 0 - отправляем в один блок
- У кого на 1 - в другой

EXTENSIBLE HASHING

- Увеличиваем массив
- Нулевой элемент указывает в тот блок, хеши которого заканчиваются на 0
- А первый - в другой
- Последний бит хеша - индекс в массиве

HASH-ИНДЕКСЫ



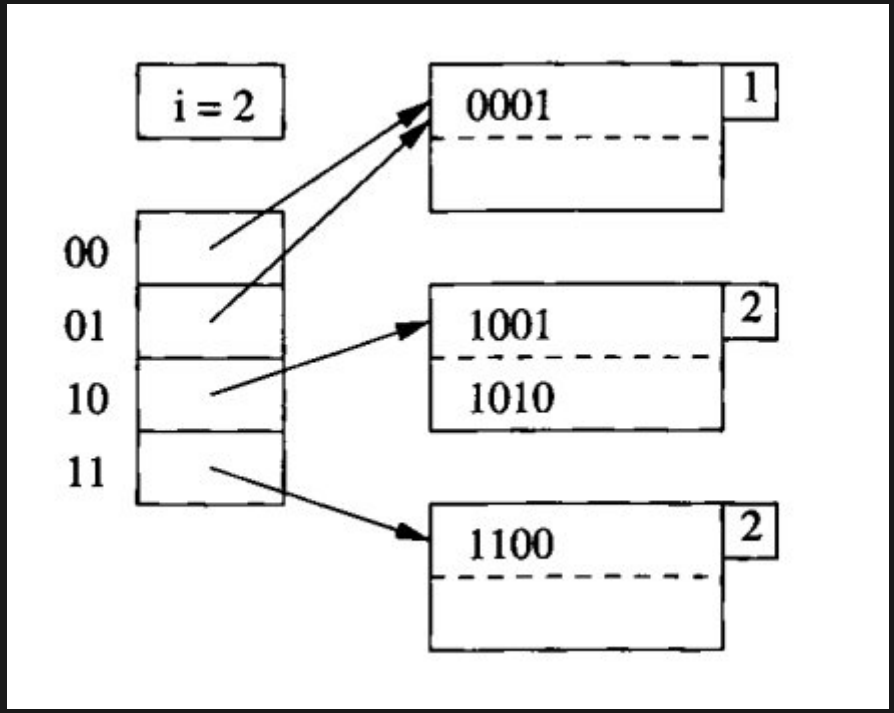
EXTENSIBLE HASHING

- Данные добавляются
- Какая-то из страниц переполняется
- Другая - пока еще нет
- Выделяем новую страницу - их теперь три

EXTENSIBLE HASHING

- Массив увеличиваем в два раза - теперь 4 элемента
- 2 бита хеша значимы
- Для одной части таблицы повторяем то, что уже делали на прошлом этапе
- А обе позиции во второй половинке ссылаются на один блок

КАРТИНКА



ПРОАНАЛИЗИРУЕМ

- Плюс: обходимся одним блоком на бакет
- Плюс: экономим память
- Минус: резкое переустройство при росте
- Минус: при дубликатах возможен "флюс" массива

LINEAR HASHING

- Избежим большого изменения при добавлении степени двойки
- За счет возможного блока переполнения
- Будем считать количество ключей и количество страниц
- И будем нечто делать, когда среднее количество ключей на страницы превысит заданный порог

LINEAR HASHING

- Пусть порог - 80 процентов от максимальной вместимости блока
- Начинаем с одного блока под номером 0
- Когда он заполнился на 80 процентов - завидим новый бакет со своим блоком
- Что было с 1 в младшем разряде хеша - переезжает туда

LINEAR HASHING

- Продолжаем заполнять блоки
- Новый создаем при достижении порога плотности хешей на бакет
- Если до этого у кого-то переполняется блок - создаем блок переполнения
- Когда достигаем порога - создаем бакет номер 2
- И хеш теперь двухбитный

LINEAR HASHING

- В блок 2 переезжают из блока 0 обладатели 1 в бите 1
- Продолжаем добавлять
- Все предназначенное для бакета 3 идет в бакет 1
- Когда создадим блок 3 - оно туда переедет

ДУБЛИРУЕМОСТЬ

- Небольшая дублируемость решаема несложной модификацией структуры страницы
- Штучные выбросы решаемы заведением специальных страниц для хранения перечня данных о страницах
- Частный случай: NULL
- Можно его вывести из структуры индекса и просто отдельно хранить страницы с данными, где значение пусто

BITMAP-ИНДЕКСЫ

- А можно списки ссылок на страницы заменить на битовые флаги
- И сильно уплотнить
- Это может быть приложением к другому индексу
- Но можно весь индекс построить на этом
- Это называется bitmap-индекс

НЕОЧЕВИДНЫЕ ПРИМЕНЕНИЯ

- Можем разбить область числовых значений на диапазоны
- И назначить им категориальные значения
- Хранить в отдельном поле с bitmask-индексом
- Тоже своего рода денормализация

СЛОЖНОСТИ BITMAP-ИНДЕКСА

- Трудно обновлять
- Сложнее всего с UPDATE
- Нужен какой-то переход от числового номера к локации страниц

ДВУМЕРНЫЕ ИНДЕКСЫ

- В одномерном случае мы разделяем точечные и диапазонные запросы
- В двумерном случае они обобщаются
- Хочется сослаться в условии на два точечных критерия по разным измерениям
- Или в одном запросе на одно, в другом - на другое
- И чтобы один индекс в обоих случаях помог

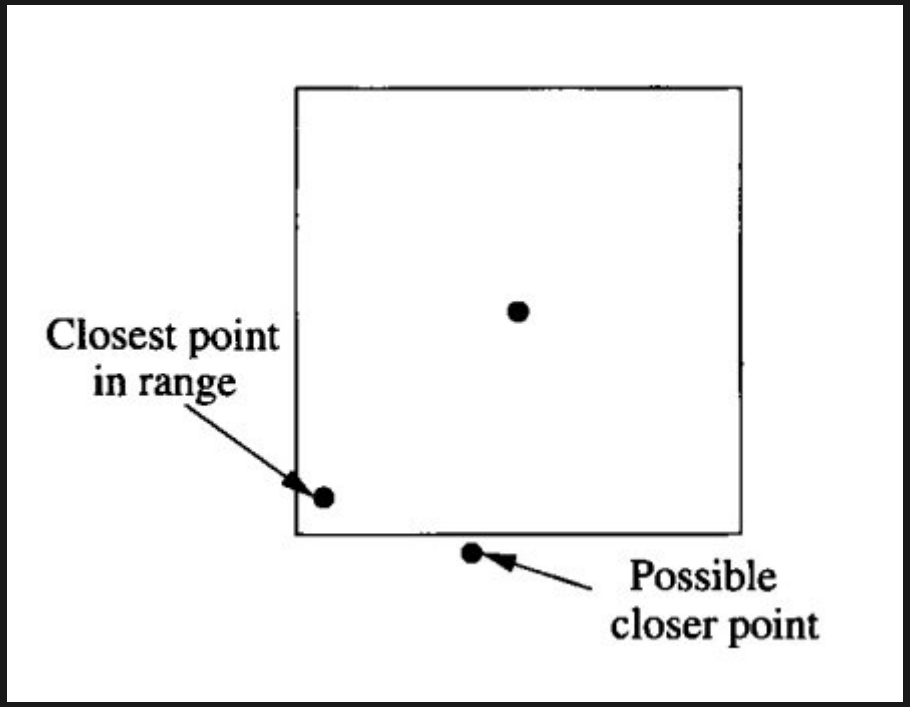
ДВУМЕРНЫЕ ИНДЕКСЫ

- Аналогично для диапазонных
- Можем захотеть чего-то смешанного
- Чтобы по одному полю диапазонное условие, а по другому - точечное
- И новая задача - поиск ближайшего (один/несколько, точно/приблизительно)
- В одномерном случае обычно не стоит отдельно

ПРОСТЕЙШЕЕ РЕШЕНИЕ

- Построить два отдельных индекса
- Получить в запросе наборы записей по каждому
- Посчитать пересечение в памяти
- Если памяти хватает - но зачастую хватает
- С ближайшим посложнее

ПРОБЛЕМА



РЕШЕНИЕ

- Находим по измерениям
- Для одного можно одним допзапросом найти кандидатов и отобрать
- Для нескольких - можно одним не обойтись
- Реально нескольких должно хватить

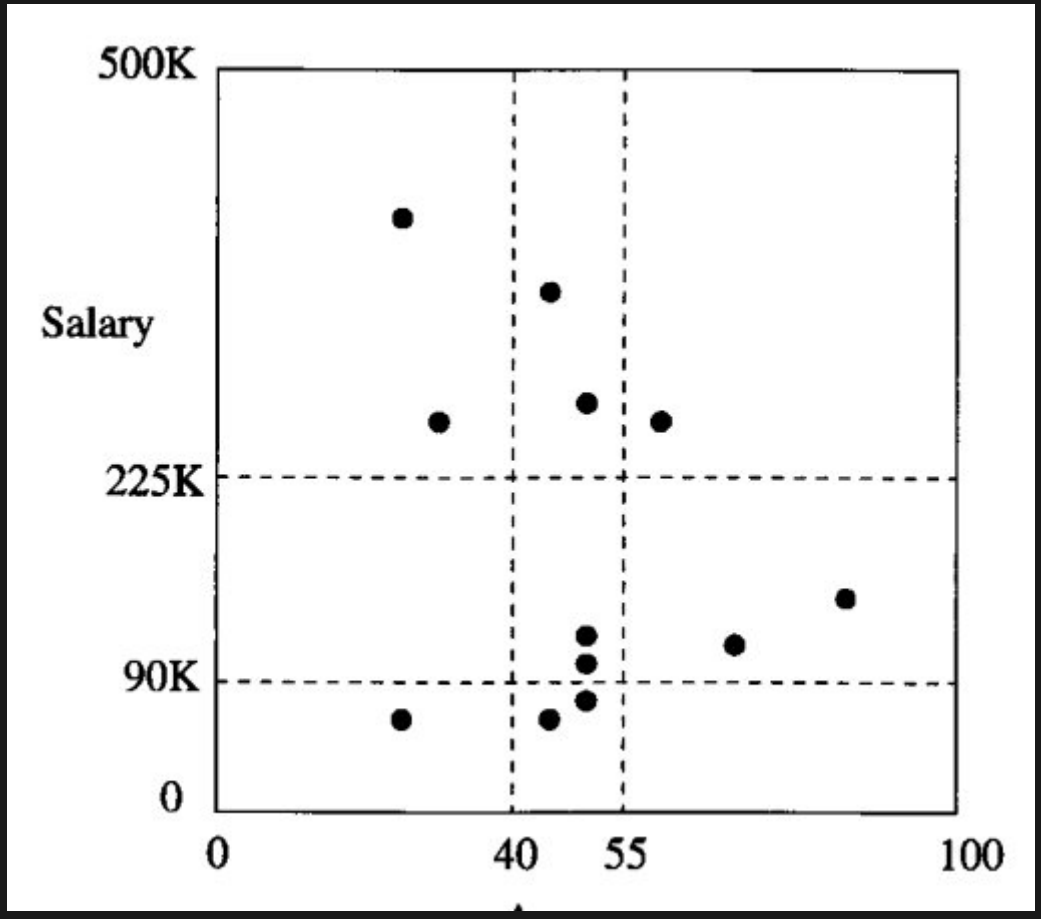
TRUE-МНОГОМЕРНЫЕ ИНДЕКСЫ

- Обобщение хешей
- Обобщение B-деревьев
- Специально двумерные

GRID: ИДЕЯ

- Представим область значений прямоугольником
- Набросаем точек, соответствующих записям
- Проведем сетку
- Чтобы примерно поровну и примерно по вместимости бакета
- Номер элемента сетки - это хеш-функция

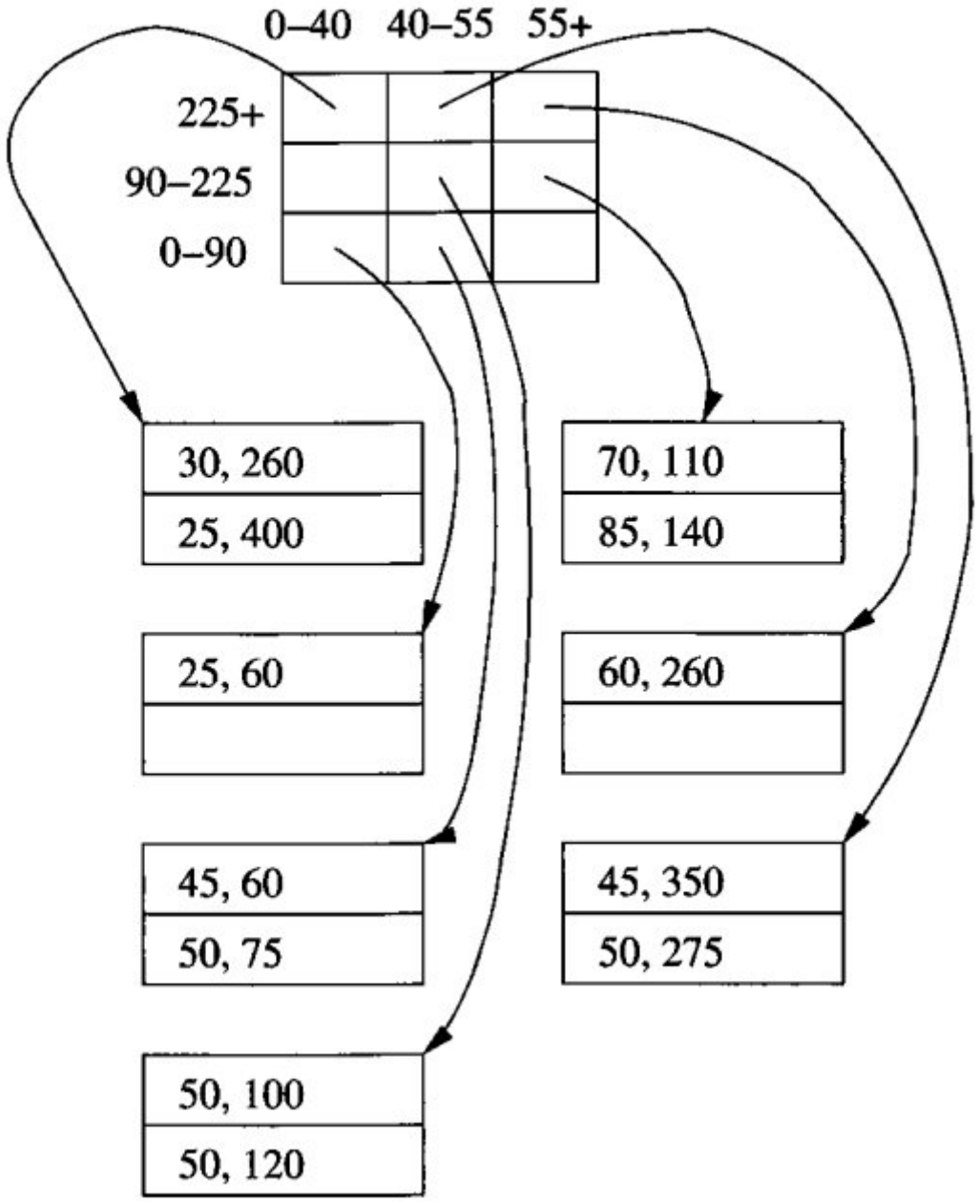
КАРТИНКА



УТОЧНЯЕМ

- По любому измерению могут идти и "категориальные" значения
- Тогда уточняются схемы запросов
- И снимается вопрос о поиске ближайшего

КАРТИНКА


0–40
40–55
55+
225+
90–225
0–90
30, 260
25, 400
70, 110
85, 140
25, 60
60, 260
45, 60
50, 75
45, 350
50, 275
50, 100
50, 120

ДОБАВЛЕНИЕ

- При переполнении создаем overflow
- Когда их накопилось много, ищем стратегию разбиения
- Проблема в том, что надо разбивать сеткой
- Нетривиальная оптимизационная задача
- Можно применять эвристики
- Основываясь на вспомогательных структурах

ПОИСК

- Знаем ячейку - идет туда и проверяем
- Диапазонный двумерный - ищем пересечение интересующей области и ячеек
- Проверяем в ячейках
- При точном запросе попадем в 1-2-3 ячейки

ПОИСК

- Если при неточном получаем много данных
  - Работает долго, но это не проблема реализации
- Если при неточном получаем проверяем много ячеек, но мало данных
  - Проблема сетки или распределения