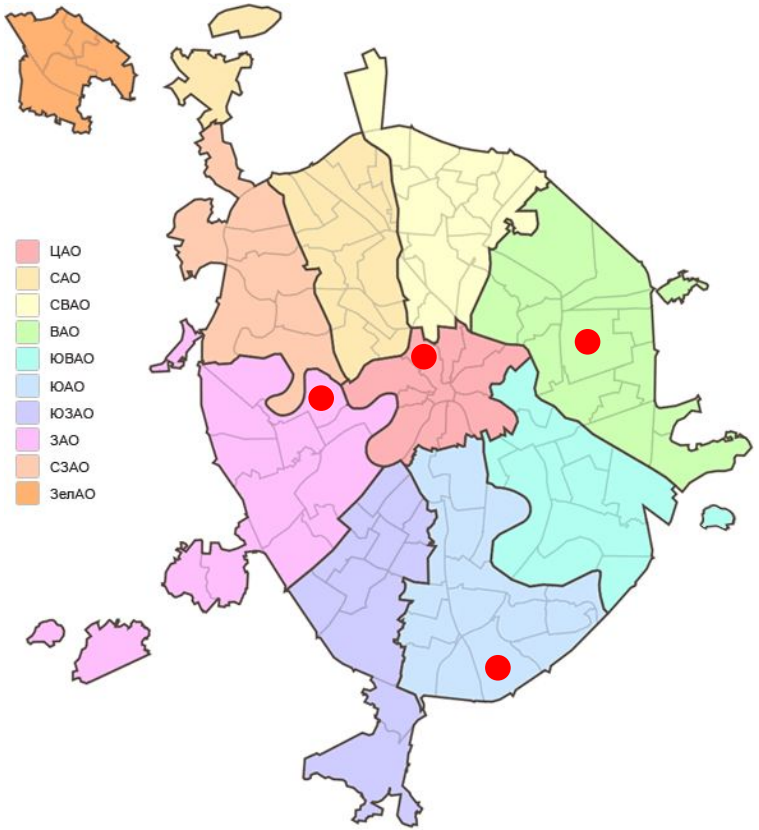
Словари полигонов и geospatial структур

Выполнили студенты 3 курса, 175 группы:
Петуховский Артур Михайлович
Чулков Андрей Сергеевич
Кваша Антон Михайлович

Научный руководитель:
Руководитель группы разработки ClickHouse,
Миловидов Алексей Николаевич

Пример из жизни

- Определение района города Москвы для большого количества геолокаций.
- Поиск полигона, покрывающего точку.



ClickHouse

- ClickHouse – аналитическая СУБД, разрабатывается на языке C++.
- ClickHouse не является geospatial СУБД, но имеет поддержку некоторых geospatial функций (например, pointInPolygon).
- Разрабатываемый функционал может использоваться для аналитики на основе местоположения.

Словари в ClickHouse

- Словарь это отображение (ключ → атрибуты), которое удобно использовать в запросах как справочник.
- Для использования словарей есть функция **dictGet(dict, attr, key)**.
- Словарем полигонов мы называем новый тип словаря, который по заданному ключу-точке возвращает атрибут найденного полигона.

Цель и задачи работы

Цель: Добавить в ClickHouse поддержку словарей полигонов.

Задачи:

- Реализовать алгоритмы для решения задачи (Артур Петуховский и Андрей Чулков).
- Реализовать интерфейс словаря полигонов (Андрей Чулков).
- Подготовить набор данных и тесты на их основе (Антон Кваша).
- Провести тестирование производительности (Антон Кваша).
- Написать пользовательскую документацию (Антон Кваша и Андрей Чулков).

Терминология

- Точка — в рамках данной работы рассматриваются точки на двумерной евклидовой плоскости.
- Кольцо — упорядоченный набор вершин, проведя ребра через которые получается замкнутый цикл без самопересечений.
- Полигон — внешнее кольцо, из которого вырезаны ноль или более непересекающихся колец, вложенных во внешнее.
- Мультиполигон — набор непересекающихся полигонов.

Поиск полигона по точке

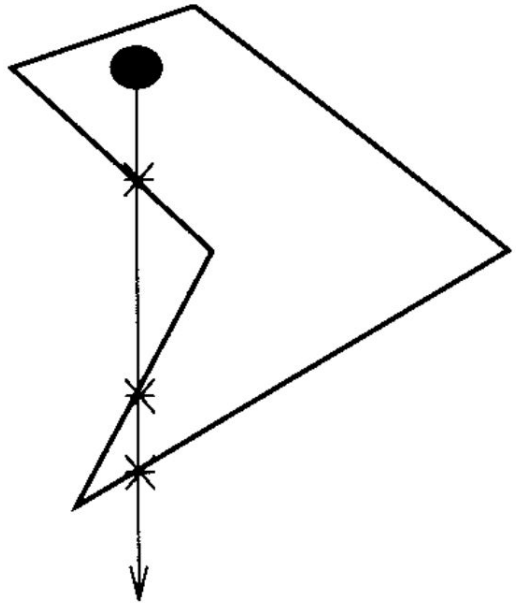
- Самое простое решение – проверить вхождение точки в каждый из полигонов.
- Большинство известных алгоритмов эффективно убирают лишние полигоны с помощью быстрых проверок, а затем точно также проверяют оставшиеся.
- Для быстрых проверок зачастую используются:
  - Q-деревья
  - R-деревья
  - kd-деревья
- Эти быстрые проверки оперируют прямоугольниками.

Обзор существующих методов

- Алгоритм проверки принадлежности точки полигону.
  - Тест четности.
  - Сетка.
  - Bounding box.
- Алгоритм локализации точки на планарном подразбиении.
- Поиск полигона, покрывающего точку.
  - RapidPolygonLookup
  - Vertica Database
  - PostGIS PostgreSQL

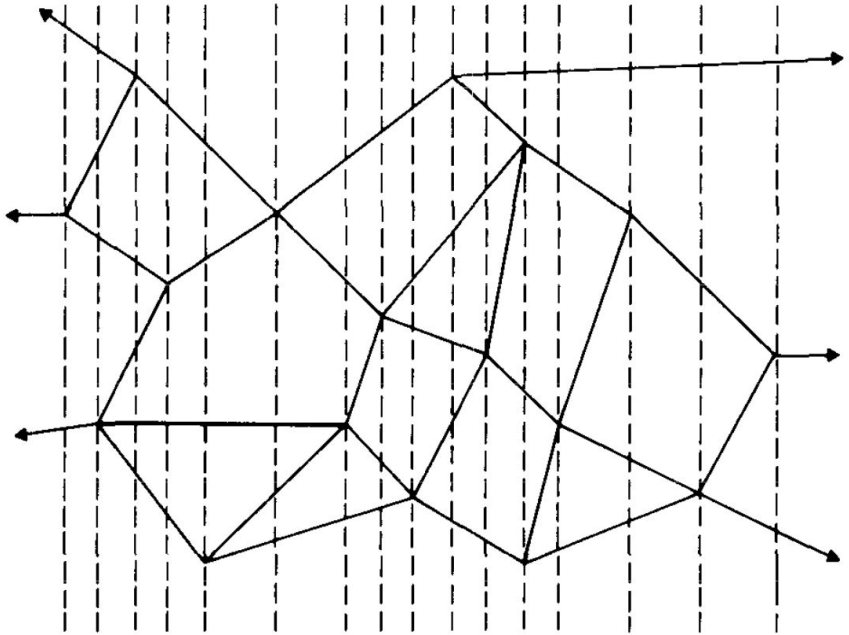
Тест четности

- Позволяет проверить принадлежность точки полигону.
- Для этого из точки направляется луч в любую сторону, и считается количество пересечений.
- Нечетное количество пересечений означает то, что точка внутри.
- Можно реализовать очень эффективно, если луч параллелен одной из осей.



Локализация точки на планарном подразбиении

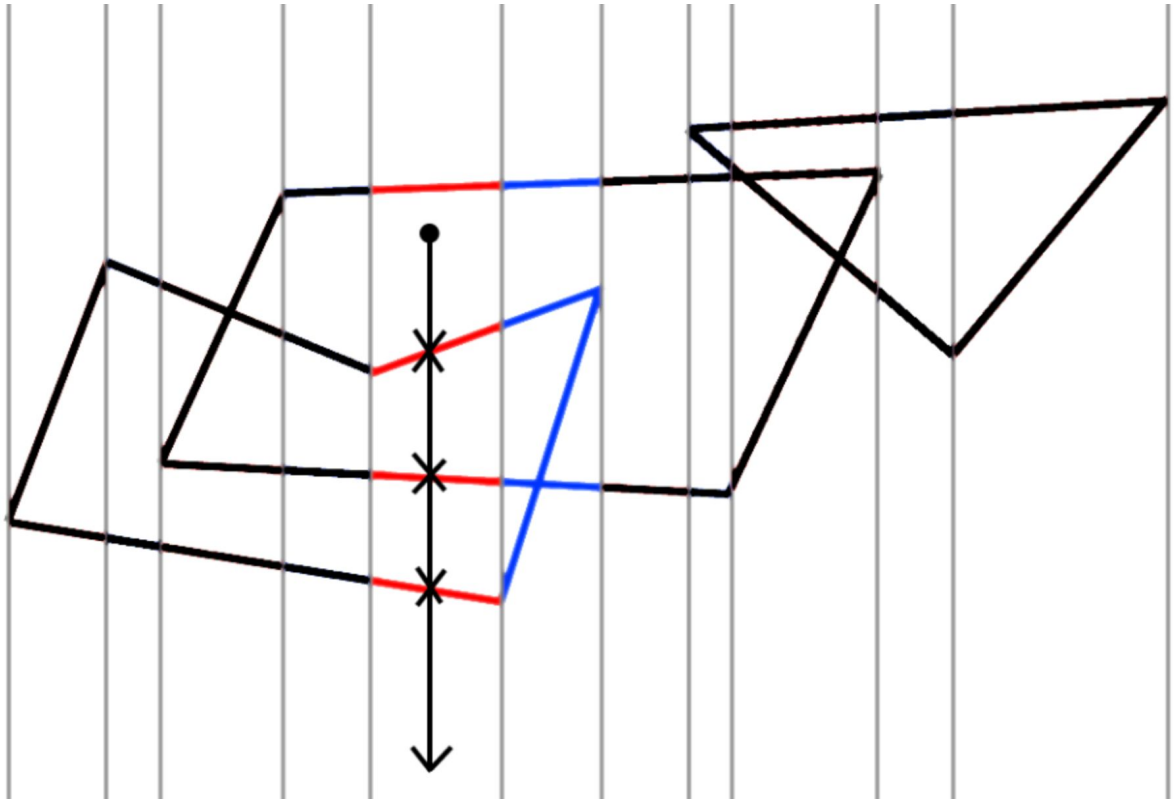
- Суть алгоритма в проведении вертикальных прямых через все точки.
- После этого строится индекс для всех вертикальных плит.
- Данный алгоритм имеет сложность O(log n) на запрос и работает только на областях без пересечений.



Алгоритм SlabsPolygonIndex

- Используем идеи теста четности и алгоритма локализации точки.
- Разделим плоскость на вертикальные плиты, проведя прямые через все точки.
- Будем использовать тест четности с лучом, направленным вниз.
- Важная идея оптимизации: для такого теста четности достаточно проверять пересечение только с элементами текущей плиты.

Иллюстрация вертикальных плит



Как хранить плиты?

- Если хранить все плиты явно, может получиться $O(n^2)$ памяти в особых случаях.
- Каждое ребро полностью лежит на отрезке из плит.
- Можно использовать дерево отрезков для хранения ребер в плитах.
- Отрезок из плит для одного ребра будет занимать $O(\log n)$ вершин дерева отрезков.
- Итого $O(n \log n)$ памяти.

Что также стоит отметить

- Алгоритм лучше всего работает на выпуклых многоугольниках.
- Оптимизации хранения, используя тип Float32 и структуры без лишних полей.
- Оптимизация проверки пересечения, заранее вычисляя коэффициенты прямой для ребер.
- Алгоритм реализован на языке C++ в виде отдельного класса SlabsPolygonIndex.
- Этот класс доступен для использования другими алгоритмами, а также может быть использован как самостоятельное решение исходной задачи.

Интерфейс внешних словарей в ClickHouse

- Словари задаются специальной xml-конфигурацией, или же DDL запросом. В частности:
  - Описываются данные: типы ключей и атрибутов.
  - Указывается источник данных — например, файл на диске.
  - Задается способ размещения в памяти — главное различие разных типов словарей. Может принимать дополнительные параметры.
- Построенные внутренние структуры, как правило, хранятся в оперативной памяти.
- Поддерживаются в основном readonly-запросы.
- Запросы к словарю автоматически распараллеливаются.

В каком формате принимать геоданные?

- С географическими границами бывают интересные ситуации:
  - группы островов
  - заморские территории
  - анклавы
- Нужно дать возможность представлять объекты, имеющие в себе дырки, или несколько несвязных частей.
- Лучше всего для этого подходит формат мультиполигона — используется в форматах OSM, GeoJSON, WKT, а также в PostGIS и пр.
- Для удобства работы с простыми многоугольниками реализована также поддержка данных в формате колец.
- В рамках ClickHouse мультиполигоны и кольца задаются как многомерные и одномерные массивы точек соответственно.

Интерфейс словарей полигонов в ClickHouse

- Словари полигонов реализованы как новые способы размещения словарей в памяти.
- Ключами в конфигурации должны быть "полигоны". Атрибуты могут быть любыми.
- После чтения:
  - Столбцы с атрибутами сохраняются в оперативной памяти в чистом виде.
  - Все данные сохраняются в виде списка полигонов, используются классы из Boost Geometry.
- Полигоны сортируются по площади. Все реализации возвращают первый ответ в порядке этой сортировки.

Наивная реализация

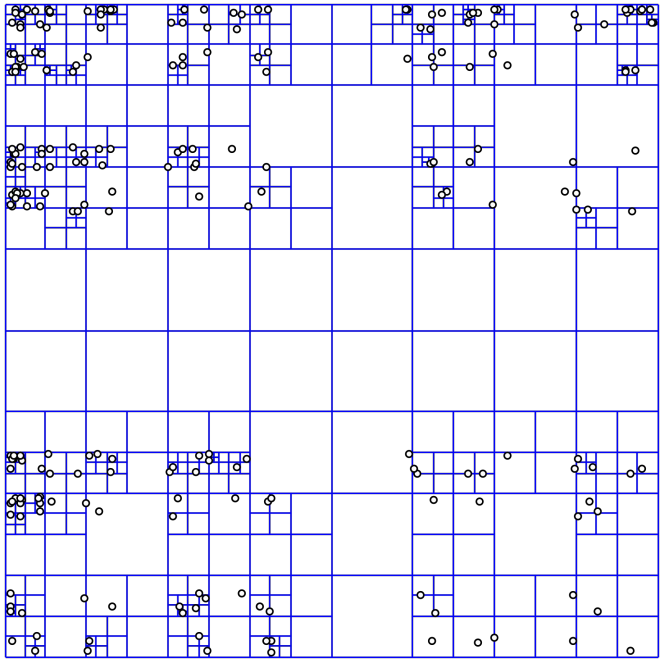
- Выполняет полный проход по всем сохраненным полигонам, проверяя точку на принадлежность к каждому из них.
- Использует линейную проверку принадлежности точки полигону из Boost Geometry.
- Полезна для проверки корректности более продвинутых реализаций.

Вспомогательная сетка: идея и мотивация

- На плоскость накладывается сетка. В ней легко найти ячейку, содержащую данную точку.
  - Известная идея для оптимизации запроса принадлежности точки полигону.
- Ячейки сетки рекурсивно разбиваются с помощью более мелкой сетки.
  - Обобщение идеи Q-деревьев, где каждая ячейка разбивается на 4 равных ячейки.
- В листовых вершинах сохраняется информация о пересекающих её полигонах.
  - В этом главное отличие от Q-деревьев — хранится не нульмерная структура, поэтому полигон может учитываться в нескольких ячейках.
- Позволяет сузить число потенциальных ответов.
- Полигоны могут иметь более простую структуру внутри листовых ячеек.

Вспомогательная сетка: иллюстрация

- Разбиение плоскости с помощью Q-дерева.
- В нашей реализации вместо точек рассматриваются целые полигоны.



Вспомогательная сетка: построение

- Начинаем с минимального (по площади) прямоугольника, содержащего данные полигоны.
- Ячейка делится на 16 равных ячеек.
- Критерии остановки рекурсии:
  - Достигнута заданная максимальная глубина — параметр MAX_DEPTH.
  - Ячейка пересекается с малым числом полигонов — параметр MIN_INTERSECTIONS. Здесь не учитываются полигоны, целиком содержащие ячейку.
- Строится параллельно, используя до 16 потоков.

Вспомогательная сетка: структура листовых ячеек

- Сохраняется индекс первого полигона, целиком содержащего ячейку.
  - Все полигоны с этим или большим индексом игнорируются.
- Для оставшихся полигонов, пересекающих ячейку:
  - *Реализация 1.* Сохраняются индексы этих полигонов.
  - *Реализация 2.* Вычисляются их пересечения с ячейкой и на получившемся множестве полигонов строится структура SlabsPolygonIndex.

Реализации продвинутых индексов

- INDEX_EACH
  - Используется сетка с первой реализацией листовых ячеек.
  - Для каждого полигона по-отдельности строится SlabsPolygonIndex.
  - Выполнение запроса:
    - Находим ячейку, содержащую данную точку.
    - Проходим по сохраненным индексам и проверяем принадлежность с помощью построенных SlabsPolygonIndex.
- INDEX_CELL
  - Используется сетка со второй реализаций листовых ячеек.
  - Выполнение запроса:
    - Находим ячейку, содержащую данную точку.
    - Используем сохраненный в ней индекс для нахождения соответствующего полигона.

Подготовка тестовых данных

- Данные в формате GeoJson с OpenStreetMap.
- Изменены в подходящий для ClickHouse формат JsonEachRow.


```
{
  "type" : "FeatureCollection",
  "features" : [{
    "type":"Feature",
      "properties" : {
        "srid"          : "4326",
        "id"            : "60189",
        "name"          : "Russia",
        "localname"     : "Россия",
        "official_name" : "Российская Федерация",
        "boundary"      : "administrative",
        "admin_level"   : "2",
        "note"          : "",
        "wikidata"      : "Q159",
        "wikipedia"     : "ru:Россия",
        "timestamp"     : "2020-01-18 02:10:02",
        "rpath"         : "60189,0",
        "alltags"       : {
          "flag" : "http://upload.wikimedia.org/wikipedia/commons/f/f3/Flag_of_Russia.svg",
          "int_ref" : "RU",
          "name:ab" : "Урыстәыла",
           …
        },
        "bbox"    : [-180, 41.185097, 180, 81.858719],
        "geometry": {"type":"MultiPolygon","coordinates":[[[[...]]]]}
  }]
}
```

Тестирование корректности

- Подготовленные вручную маленькие тесты.
- Случайные точки на небольшом реальном датасете.
- Сравнение результата работы с наивным алгоритмом.

Тестирование производительности

- Датасет большего размера — границы районов России. Занимает около 400 MB.
- 20000 полигонов, 12 миллионов вершин.
- Миллион случайных запросов, выполняемых в один поток.
- Опция --time, htop, стандартный time в Unix.

Тестирование производительности, INDEX_EACH

MAX_DEPTH	MIN_INTERSECTIONS	Время построения (в секундах)	Время выполнения запроса (в секундах)	Потребление памяти (в гигабайтах)	Скорость ответа на запросы (запросов в секунду)
2	1	7.55	15.2	2.2	65000
3	1	8.423	3.134	2.19	320000
4	1	12.8	1.7	2.18	588000
5	1	48.3	1.324	2.15	755000
6	3	75	1.372	2.1	729000

Тестирование производительности, INDEX_CELL

MAX_DEPTH	MIN_INTERSECTIONS	Время построения (в секундах)	Время выполнения запроса (в секундах)	Потребление памяти (в гигабайтах)	Скорость ответа на запросы (запросов в секунду)
0	1	8.944	7.381	2.94	135000
1	1	6.432	3.212	2.45	311000
2	1	3.907	1.711	2.17	584000
3	1	4.95	1.301	1.87	768000
4	1	11.5	1.122	1.88	891000
5	1	66	1.106	2.07	904000
6	1	610	1.115	2.7	897000
6	3	147	1.095	2.63	913000

Выводы по производительности

- Достигнута производительность в около **900000** QPS и около **2** GB оперативной памяти для данного датасета, при времени построения меньше **60** секунд.
- Уменьшая глубину сетки, можно уменьшить время построения, не сильно проиграв по времени работы запросов.
- В реальности запросы будут распараллеливаться, то есть время выполнения большого числа запросов будет ещё меньше.

Достигнутые результаты

- Изучены существующие решения поставленной задачи.
- Добавлена поддержка новых словарей.
- Написана пользовательская документация.
- Реализованы несколько разных алгоритмов, решающих задачу.
- Подготовлены тесты.
- Проведены бенчмарки.
- Достигнута ожидаемая производительность.

Направления дальнейшей разработки

- Добавление тестов производительности в CI ClickHouse (performance tests).
- Тестирование на реальных запросах — например, на анонимизированных геолокациях пользователей.
- Унификация чтения и хранения геометрических типов данных с другими геопространственными проектами, выполненными в этом году.
- Дальнейшее ускорение и поиск новых подходов.