УРОК ГЕОГРАФИИ ДЛЯ JAVA-РАЗРАБОТЧИКА. ГЕОГРАФИЯ И JAVA В ОДНОМ ЧАЙНИКЕ.

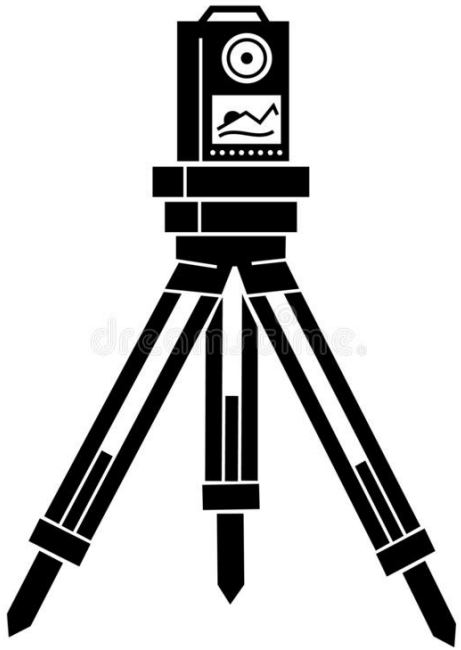
Ставь лайк если прогуливал географию, а оказалось, что не знаешь геодезию.


КТО Я?

Кислов Павел

- Работаю в компании Домклик.
- Около 6 лет в разработке.
- Люблю большой теннис
- Играю на нескольких музыкальных инструментах
- В свободное время преподаю Java.

О чем будем разговаривать?



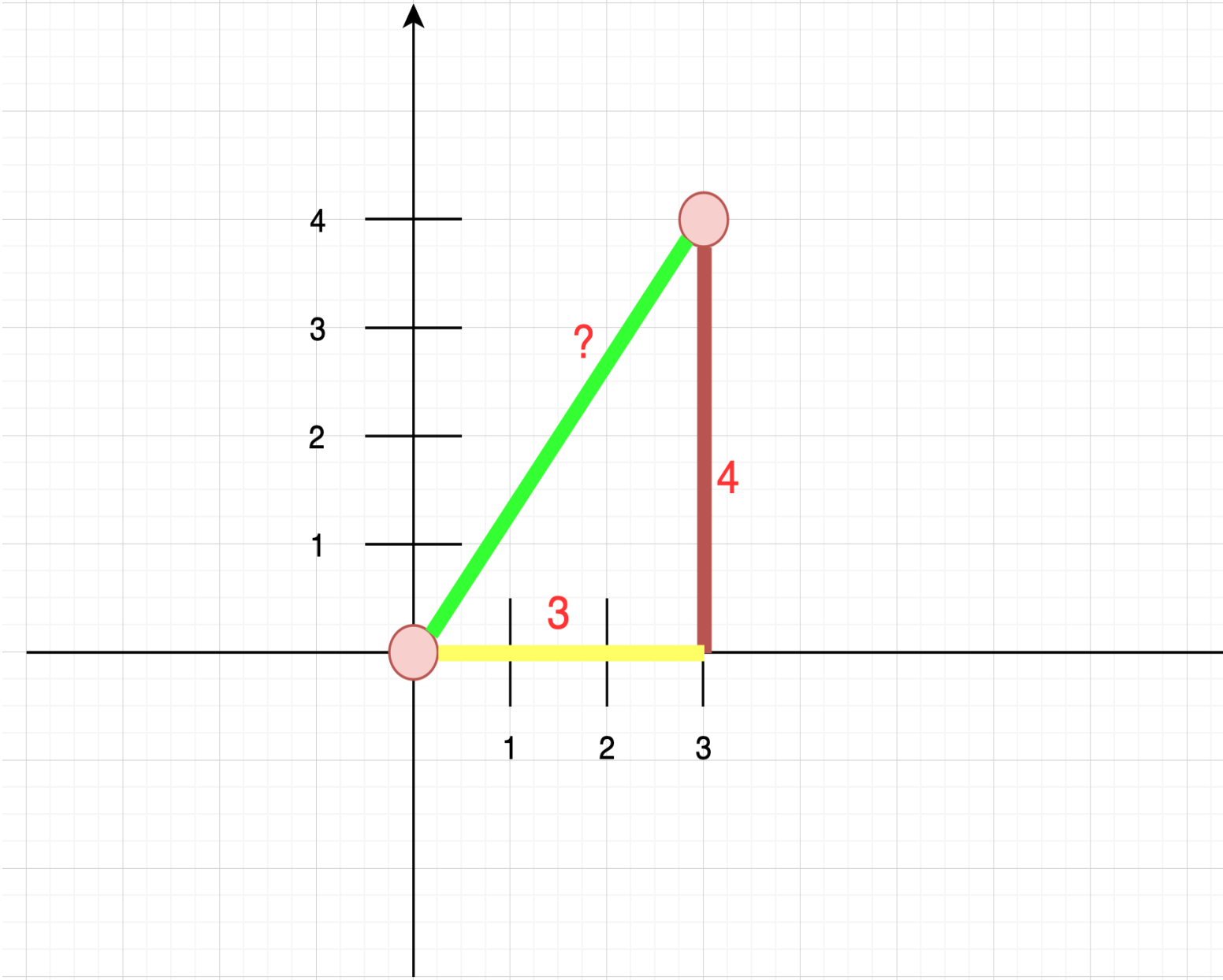
+

С чего все обычно начинается?

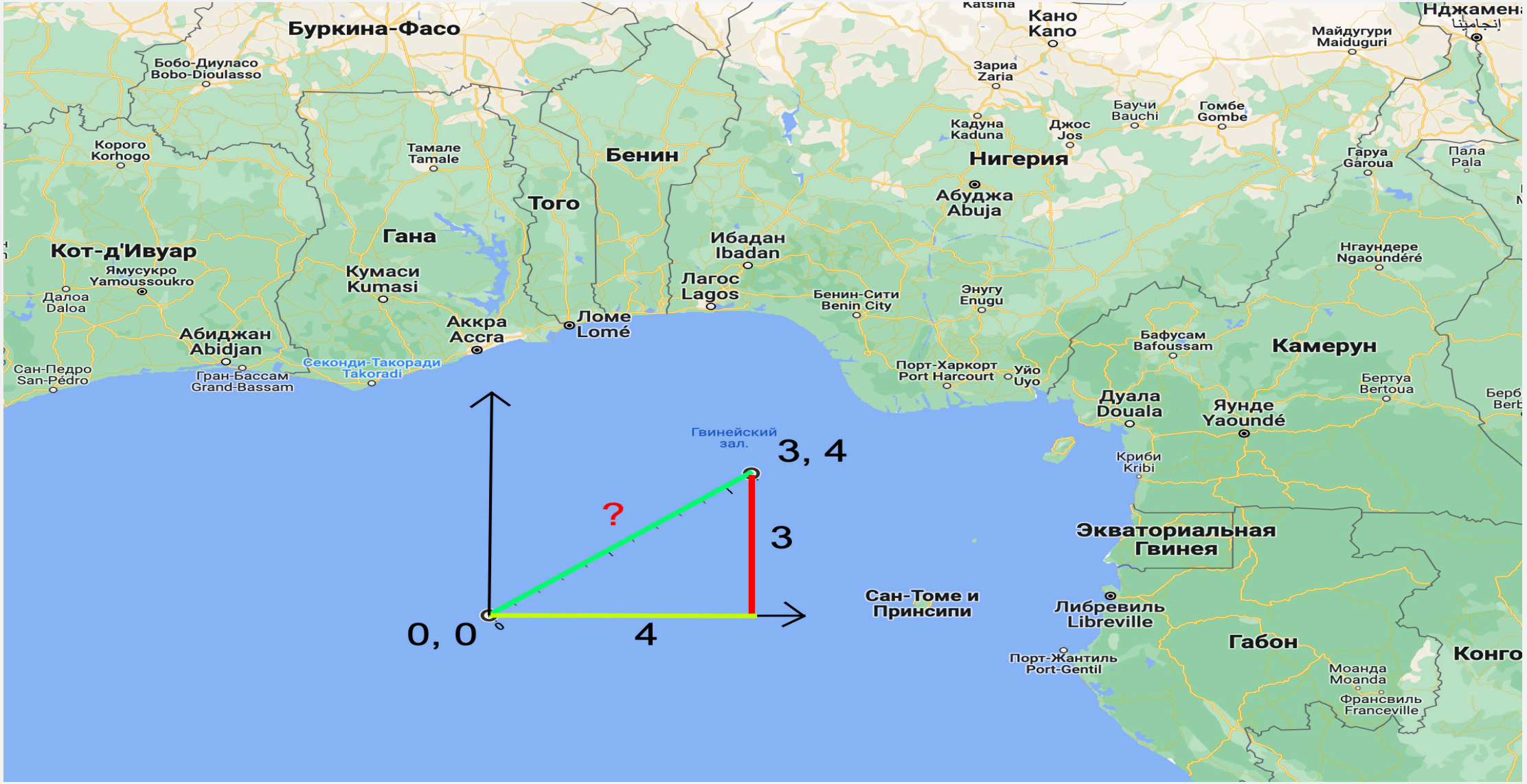
Эй, привет! Я все придумал. Теперь наша система будет уметь хранить точки, которые обозначают местоположение ларьков нашей сети и уметь считать расстояние между ними, а также показывать ближайший ларек к нашему клиенту!! Все понятно? Спасибо! Отлично! Надо к завтрашнему утру выкатить новый функционал в прод!!!

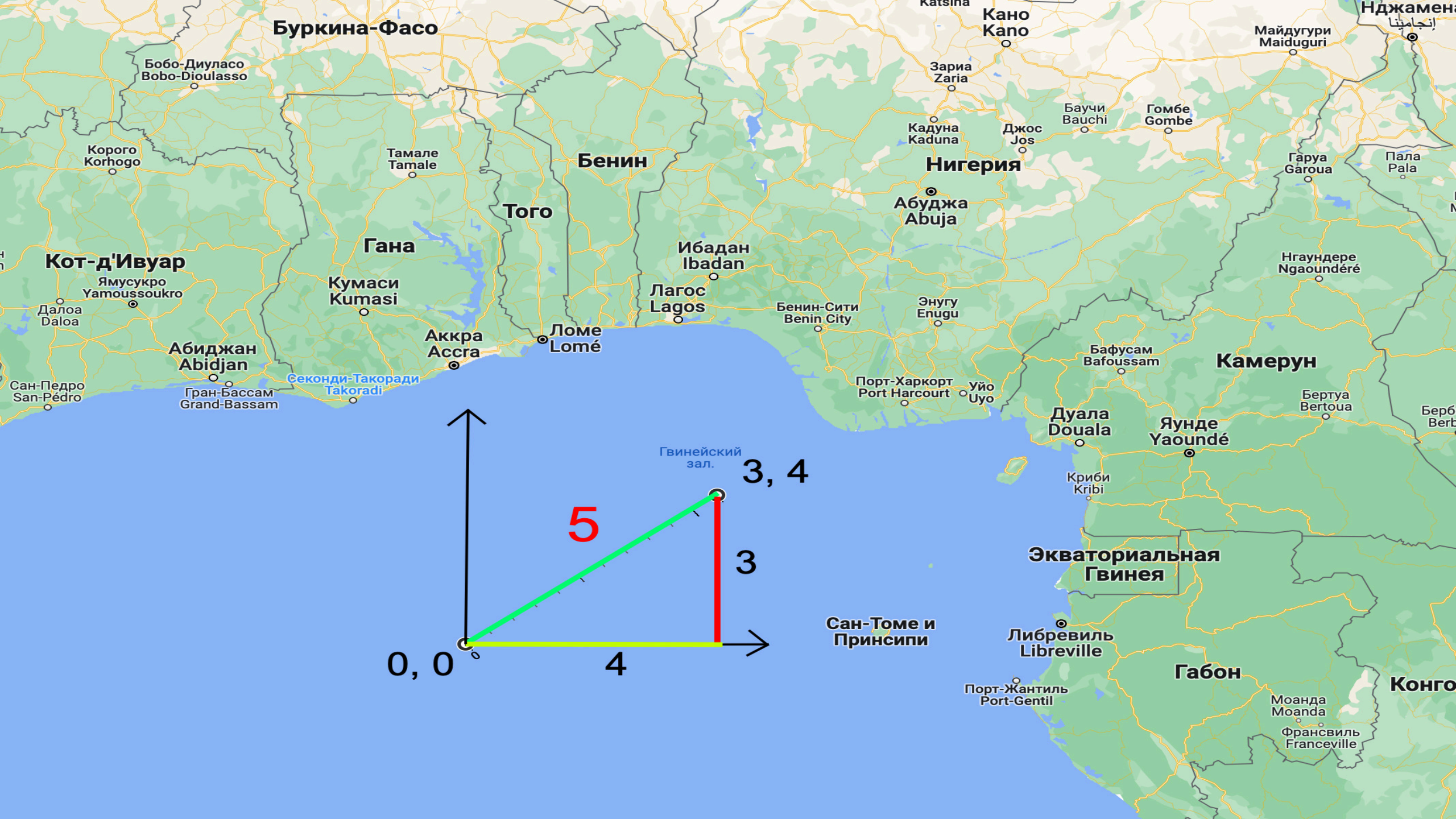
Где мы находимся в этой ситуации?

Приседание 1



Приседание 2

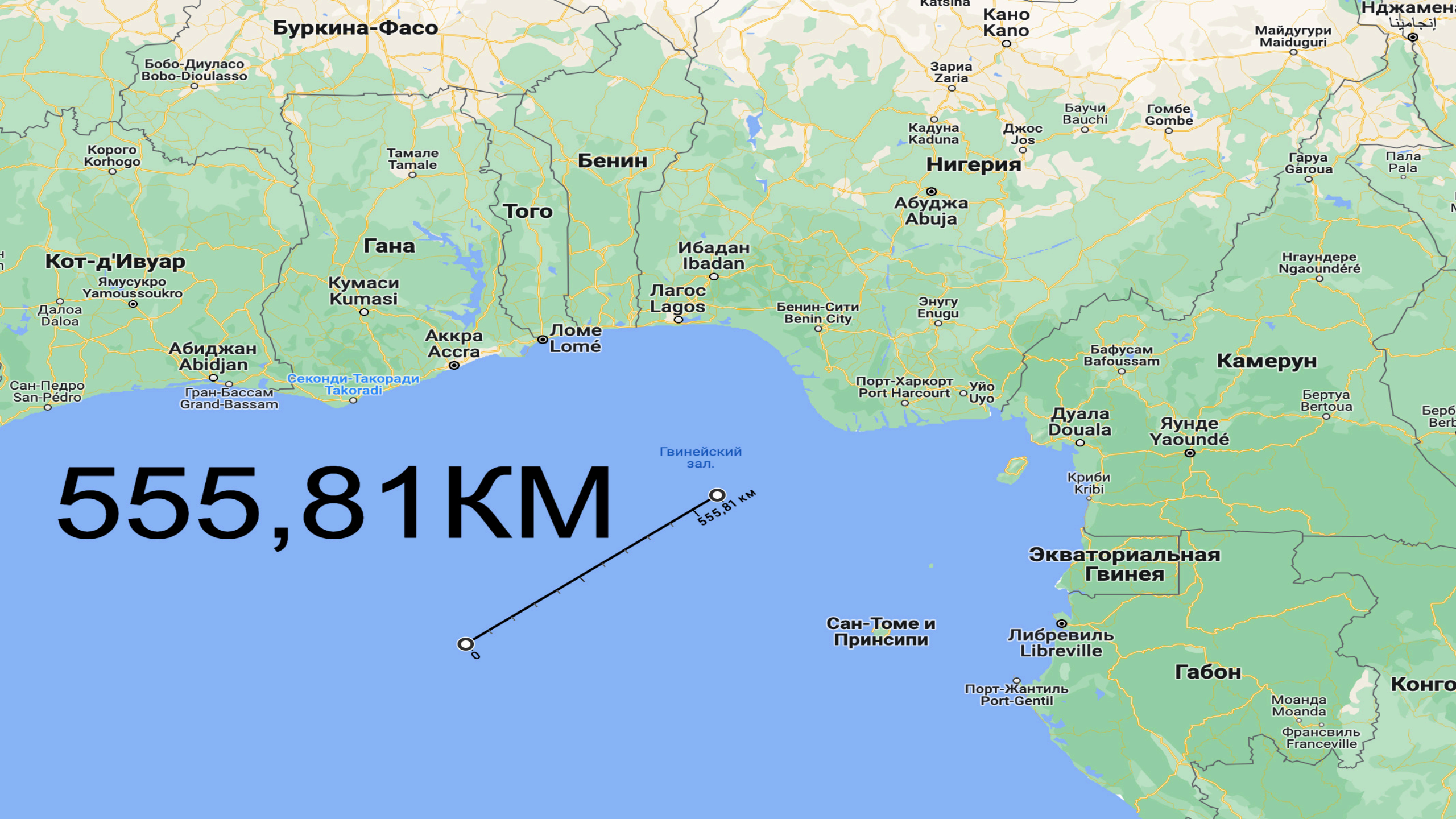



Буркина-Фасо
Бобо-Диуласо
Bobo-Dioulasso
Кано
Kano
Майдугури
Maiduguri
Нджамена
Зариа
Zaria
Баучи
Bauchi
Гомбе
Gombe
Кадуна
Kaduna
Джос
Jos
Корого
Korhogo
Тамале
Tamale
Бенин
Нигерия
Гаруа
Garoua
Пала
Pala
Того
Абуджа
Abuja
Гана
Ибадан
Ibadan
Нгаундере
Ngaoundéré
Кот-д'Ивуар
Ямусукро
Yamoussoukro
Кумаси
Kumasi
Лагос
Lagos
Энугу
Enugu
Далоа
Daloa
Бенин-Сити
Benin City
Аккра
Accra
Ломе
Lomé
Бафусам
Bafoussam
Камерун
Абиджан
Abidjan
Порт-Харкорт
Port Harcourt
Уйо
Uyo
Сан-Педро
San-Pédro
Секонди-Такоради
Takoradi
Гран-Бассам
Grand-Bassam
Дуала
Douala
Яунде
Yaoundé
Бертуа
Bertoua
Гвинейский зал.
Криби
Kribi
3, 4
5
3
Экваториальная Гвинея
Сан-Томе и Принсипи
Либревиль
Libreville
Габон
Порт-Жантиль
Port-Gentil
Моанда
Moanda
Франсвиль
Franceville
Конго
0, 0
4

Евклид и Ко


Пять чего?
5
3, 4
3
4
0, 0
Буркина-Фасо
Бобо-Диуласо
Bobo-Dioulasso
Корого
Korhogo
Тамале
Tamale
Бенин
Того
Гана
Кот-д'Ивуар
Ямусукро
Yamoussoukro
Далоа
Daloa
Кумаси
Kumasi
Аккра
Accra
Ломе
Lomé
Абиджан
Abidjan
Сан-Педро
San-Pédro
Гран-Бассам
Grand-Bassam
Секонди-Такоради
Takoradi
Ибадан
Ibadan
Лагос
Lagos
Бенин-Сити
Benin City
Энугу
Enugu
Порт-Харкорт
Port Harcourt
Уйо
Uyo
Кано
Kano
Зариа
Zaria
Кадуна
Kaduna
Джос
Jos
Баучи
Bauchi
Гомбе
Gombe
Нигерия
Абуджа
Abuja
Майдугури
Maiduguri
Гаруа
Garoua
Пала
Pala
Нджамена
Нгаундере
Ngaoundéré
Бафусам
Bafoussam
Камерун
Бертуа
Bertoua
Дуала
Douala
Яунде
Yaoundé
Криби
Kribi
Гвинейский зал.
Экваториальная Гвинея
Сан-Томе и Принсипи
Либревиль
Libreville
Габон
Порт-Жантиль
Port-Gentil
Моанда
Moanda
Франсвиль
Franceville
Конго

И снова галерка


555,81КМ
Буркина-Фасо
Бобо-Диуласо
Bobo-Dioulasso
Корого
Korhogo
Тамале
Tamale
Бенин
Того
Гана
Кот-д'Ивуар
Ямусукро
Yamoussoukro
Далоа
Daloa
Кумаси
Kumasi
Ибадан
Ibadan
Лагос
Lagos
Аккра
Accra
Ломе
Lomé
Абиджан
Abidjan
Сан-Педро
San-Pédro
Гран-Бассам
Grand-Bassam
Секонди-Такоради
Takoradi
Бенин-Сити
Benin City
Энугу
Enugu
Порт-Харкорт
Port Harcourt
Уйо
Uyo
Кано
Kano
Зариа
Zaria
Кадуна
Kaduna
Джос
Jos
Баучи
Bauchi
Гомбе
Gombe
Нигерия
Абуджа
Abuja
Майдугури
Maiduguri
Гаруа
Garoua
Пала
Pala
Нгаундере
Ngaoundéré
Бафусам
Bafoussam
Камерун
Бертуа
Bertoua
Дуала
Douala
Яунде
Yaoundé
Криби
Kribi
Гвинейский зал.
555,81 км
0
Экваториальная Гвинея
Сан-Томе и Принсипи
Либревиль
Libreville
Габон
Порт-Жантиль
Port-Gentil
Моанда
Moanda
Франсвиль
Franceville

Разбираемся в чем дело.

Земля имеет форму подобную шару и Евклидова геометрия, которая отлично прокатывала на плоскости, здесь не работает, а теорема Пифагора вообще вышла из чата.

Разбираемся в чем дело.

Земля имеет форму подобную шару и Евклидова геометрия, которая отлично прокатывала на плоскости, здесь не работает, а теорема Пифагора вообще вышла из чата.

Карта – это плоская проекция земного шара. Таких проекций существует несколько, например, проекция Меркатора и проекция Гаусса-Крюгера.

Разбираемся в чем дело.

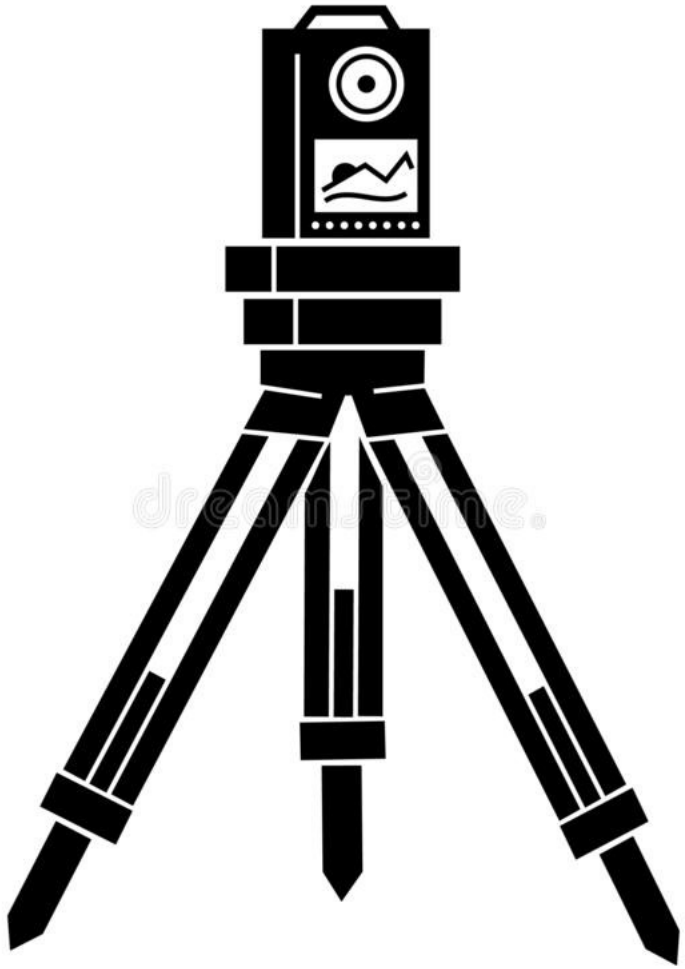
Земля имеет форму подобную шару и Евклидова геометрия, которая отлично прокатывала на плоскости, здесь не работает, а теорема Пифагора вообще вышла из чата.

Карта – это плоская проекция земного шара. Таких проекций существует несколько, например, проекция Меркатора и проекция Гаусса-Крюгера.

Нашими координатами были широта и долгота.

Геодезия vs география

VS



Системы координат в геодезии

- Эллипсоидальные(lat, lon, att)

Системы координат в геодезии

- Эллипсоидальные(lat, lon, att)
- Прямоугольные
  - Пространственные (X, Y, Z)
  - Плоские(X, Y)

Минутка теории

- Референц-эллипсойд

Минутка теории

- Референц-эллипсойд
- SRID

Минутка теории

- Референц-эллипсойд
- SRID
- EPSG

Минутка теории

- Референц-эллипсойд
- SRID
- EPSG
- Проекции

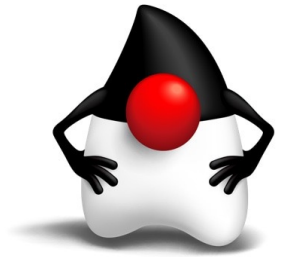
Минутка теории

- Референц-эллипсойд
- SRID
- EPSG
- Проекции
- Растровые и векторные данные

Что начет Java?

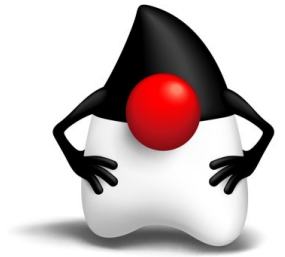
Что у нас есть?

- JTS



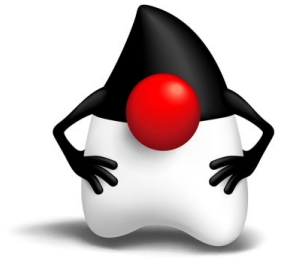
Что у нас есть?

- JTS
- Geolatte-geom



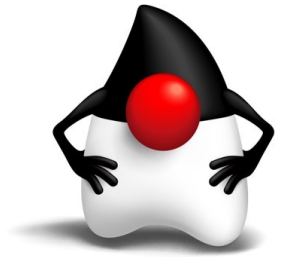
Что у нас есть?

- JTS
- Geolatte-geom
- Hibernate Spatial



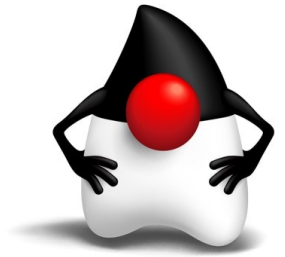
Что у нас есть?

- JTS
- Geolatte-geom
- Hibernate Spatial
- Geotools



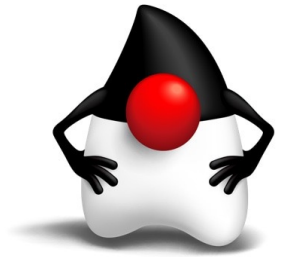
Что у нас есть?

- JTS
- Geolatte-geom
- Hibernate Spatial
- Geotools
- GeoServer



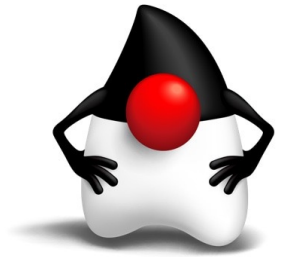
Что у нас есть?

- JTS
- Geolatte-geom
- Hibernate Spatial
- Geotools
- GeoServer
- Apahe SIS



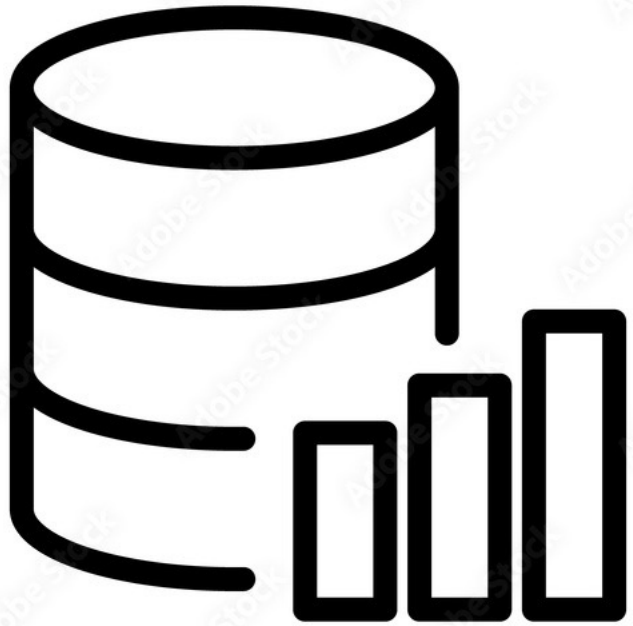
Что у нас есть?

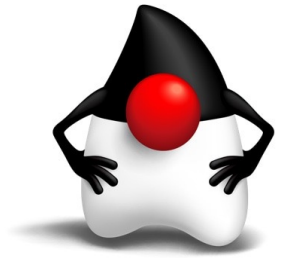
- JTS
- Geolatte-geom
- Hibernate Spatial
- Geotools
- GeoServer
- Apahe SIS



Где храним?

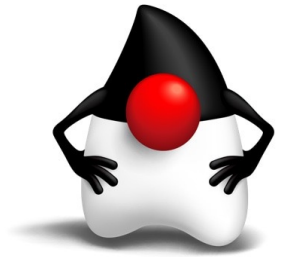
- PostGIS
- Oracle Spatial





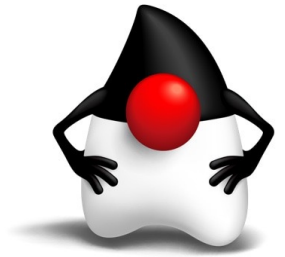
Где еще можно?

- H2
- Redis
- Tarantool
- MySQL
- AllegroGraph
- GeoMesa
- Microsoft SQL Server
- IBM Db2
- Neo4j
- RethinkDB
- ArangoDB



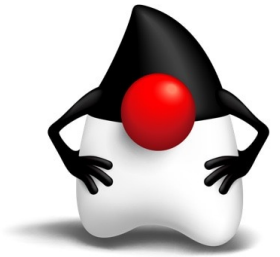
Наиболее частые задачи?

- Найти расстояние между точками



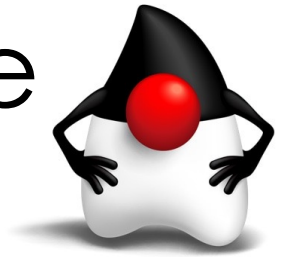
Наиболее частые задачи?

- Найти расстояние между точками
- Выяснить принадлежит ли точка многоугольнику(полигону) или выбрать все точки внутри некоторого пространства



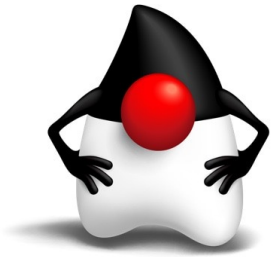
Наиболее частые задачи?

- Найти расстояние между точками
- Выяснить принадлежит ли точка многоугольнику(полигону) или выбрать все точки внутри некоторого пространства
- Найти все многоугольники, которые содержат точку



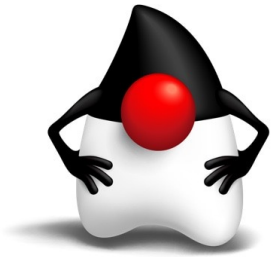
Задача 1
Расстояние между точками

- JTS
- Geolatte-geom
- Hibernate Spatial
- SpringData
- PostGIS



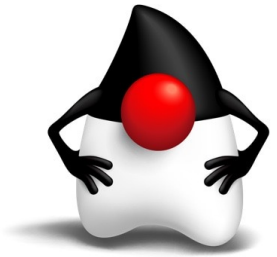
Зависимости

```
implementation("org.springframework.boot:spring-boot-starter-data-jpa")
implementation("org.springframework.boot:spring-boot-starter-web")
implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
implementation("org.jetbrains.kotlin:kotlin-reflect")
implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
implementation("org.liquibase:liquibase-core")
implementation( "org.hibernate:hibernate-spatial:5.4.30.Final")
testImplementation("org.springframework.boot:spring-boot-starter-test")
testImplementation("org.testcontainers:junit-jupiter")
testImplementation
("org.testcontainers:testcontainers:$testContainersVersion")
testImplementation
("org.testcontainers:postgresql:$testContainersVersion")
testImplementation("io.mockk:mockk:1.9.1")
```



YML-файл

```
spring:
  datasource:
    url: jdbc:tc:postgis:9.6.8-2.5://localhost/jpa-spatial
    username: portal
    password: uhSBTvdHCnS7pyUgOTDX
    driver-class-name: org.testcontainers.jdbc.ContainerDatabaseDriver
    hikari:
      driver-class-name: org.testcontainers.jdbc.ContainerDatabaseDriver
  jpa.properties.hibernate:
    show-sql: true
    format_sql: true
    jdbc.lob.non_contextual_creation: true
    dialect: org.hibernate.spatial.dialect.postgis.PostgisDialect
```



Сущность

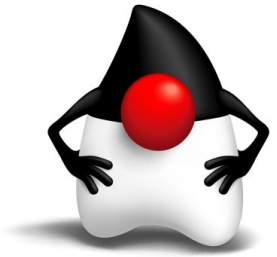
```
@Entity
@Table(name = "shop")
open class Shop {
    // какие-то поля и методы
    @Column(columnDefinition = "geometry", nullable = false)
    val position: Point
}
```

Можно

```
import org.geolatte.geom.Point – с определенными танцами
import org.locationtech.jts.geom.Point – просто и понятно
```

Нельзя

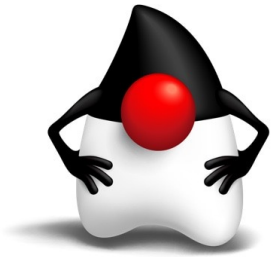
```
import org.springframework.data.geo.Point – класс нужен не для
этого.
```



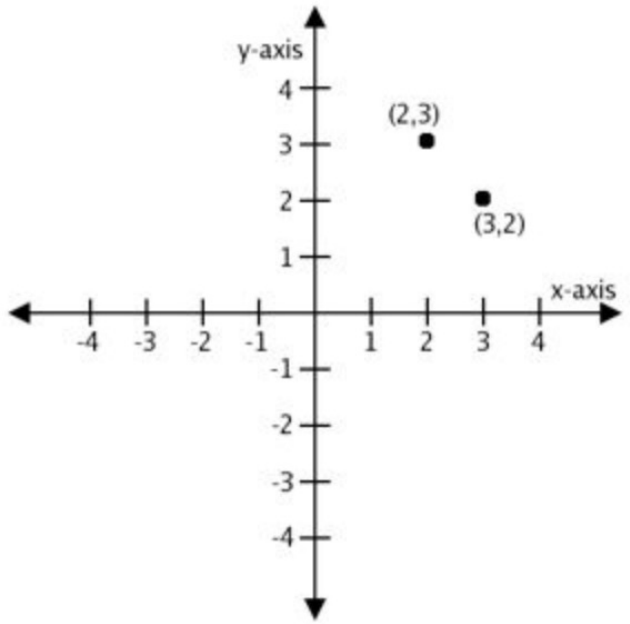
Таблица

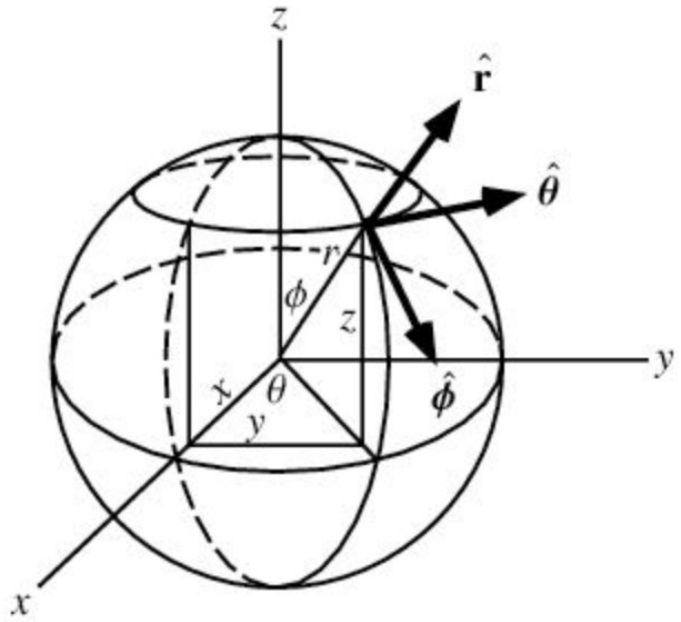
```
CREATE TABLE shop
(
    id            BIGINT                    NOT NULL PRIMARY KEY,
    name          VARCHAR,
    address       VARCHAR,
    working_hours VARCHAR,
    position      geometry(POINT, 4326) NOT NULL
);
```

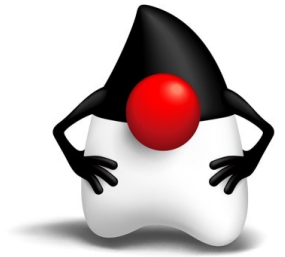
Можно:
geometry
geography – при работе с postGIS, hibernate поддерживает работу с этим типом.
Но важно помнить, про разницу геометрии и географии.
В общем случае, при использовании jpql, не стоит так делать.



Geometry vs Geography







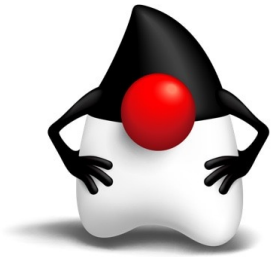
Geometry vs Geography

Основа для типа Geography – сфера. Кратчайший путь между двумя точками на сфере — дуга. Координаты – широта и долгота.

Основа для типа Geometry - плоскость. Кратчайший путь между двумя точками на плоскости — прямая. Координаты для 4326 - широта и долгота, единица измерения градусы. Для подсчета расстояния «как на сфере» используются мат. преобразования. В случае, когда нет пространственной привязки – это тип, который работает с обычной Евклидовой геометрией.

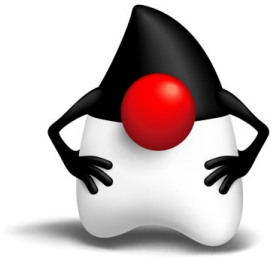
Функций которые умеют работать с geography меньше.
Расстояния с которыми работает geography – это сразу метры, футы и прочие понятные величины.

Функций для geometry много.
Величины возвращаются в единицах, с которыми работает проекция(Например, радианы, градусы и т.д.)



Конфиг

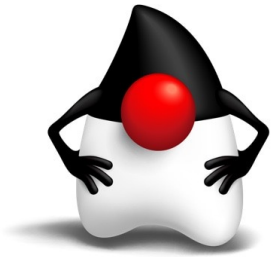
```
@Configuration
class BeansConfig {
    @Bean
    fun geometryFactory() = GeometryFactory(PrecisionModel(), SRID)

    companion object {
        private const val SRID = 4326
    }
}
```



Репозиторий

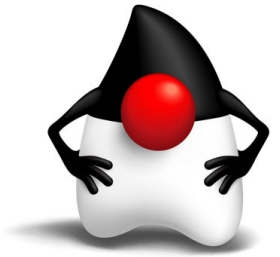
st_distancesphere – для того чтобы получить расстояние в метрах.

```
@Repository
interface ShopRepository : JpaRepository<Shop, Long> {
    @Query(
        value = "select s.id, " +
                "s.name, " +
                "s.address, " +
                "s.working_hours, " +
                "round(" +
                "cast(st_distancesphere(position, :point) as numeric), 2) as distance, " +
                "s.lat, s.lon from shop s order by distance",
        nativeQuery = true
    )
    fun findAllShopsOrderedByDistance(@Param("point") point: Point): List<ShopProjection>
}
```



Заполняем базу тестовыми данными

```
INSERT INTO SHOP(id, name, address, working_hours, position)
VALUES (1, 'Липовый мед Илъича', 'г. Москва, Красная площадь',
        'Пн - Пт: 9:00 - 21:00 Сб, Вс - выхоные дни’,
        st_setsrid(st_makepoint(37.6211812, 55.7538337), 4326));

INSERT INTO SHOP(id, name, address, working_hours, position)
VALUES (2, 'Холодный холод', 'г. Санкт - Петербург, Дворцовая площадь',
        'Пн - Пт: 9:00 - 21:00 Сб, Вс - выхоные дни’,
        st_setsrid(st_makepoint(30.315212, 59.938879), 4326));

…и какие-то еще точки
```



Тестируем

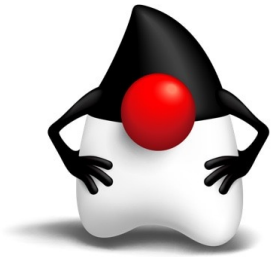
Задаем точку относительно которой будем считать расстояние

val pointToOrderFrom = geometryFactory.createPoint(Coordinate(37.6211812, 55.7538337))

Запускаем наш метод

shopRepository.findShopsOrderedByDistance(pointToOrderFrom)

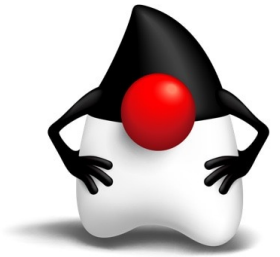
Полученный результат:
Липовый мед Ильича = 0.0 метров от исходной точки
Холодный холод = 634495.16 метров от исходной точки



JPQL МОЖНО

```
@Query("select shop from Shop shop order by distance(:point, shop.position)")
fun findShopsOrderedByDistance(@Param("point") point: Point): List<Shop>
```

Для геометрии при работе с SRID 4326 функция distance считает расстояние в градусах по аналогии с функцией st_distance у postgis при тех же вводных.

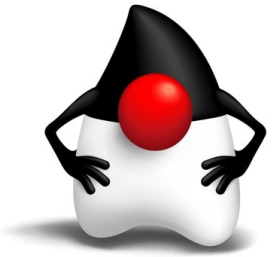


Hibernate Spatial functions

int dimension(Geometry)
String geometrytype(Geometry)
int srid(Geometry)
Geometry envelope(Geometry)
String astext(Geometry)
byte[] asbinary(Geometry)
boolean isempty(Geometry)
boolean issimple(Geometry)
Geometry boundary(Geometry)
boolean equals(Geometry, Geometry)
boolean disjoint(Geometry, Geometry)
boolean intersects(Geometry, Geometry)
boolean touches(Geometry, Geometry)

boolean crosses(Geometry, Geometry)
boolean within(Geometry, Geometry)
boolean contains(Geometry, Geometry)
boolean overlaps(Geometry, Geometry)
boolean relate(Geometry, Geometry, String)
double distance(Geometry, Geometry)
Geometry buffer(Geometry, double)
Geometry convexhull(Geometry)
Geometry intersection(Geometry, Geometry)
Geometry geomunion(Geometry, Geometry)
Geometry difference(Geometry, Geometry)
Geometry symdifference(Geometry, Geometry)
boolean dwithin(Geometry, Geometry, double)
Geometry transform(Geometry, int)
Geometry extent(Geometry)

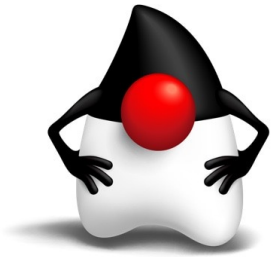
Для любителей подробностей есть документация:
http://www.hibernatespatial.org/documentation/03-dialects/01-overview/



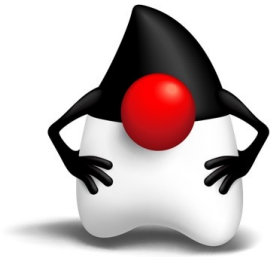
PostGIS functions

- SQL-MM Compliant Functions
- Aggregate Functions
- Geography Support Functions
- Geometry Dump Functions
- Box Functions
- Functions that support 3D
- Curved Geometry Support Functions

Для любителей подробностей есть документация:
https://postgis.net/docs/manual-1.5/ch08.html

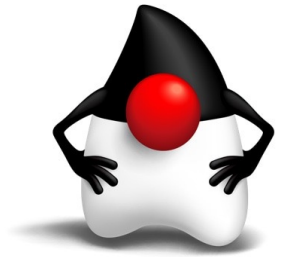


Задача 3
Найти все многоугольники, которые содержат точку.



Как выяснить, что точка принадлежит многоугольнику?

- Метод трассировки лучей
- Метод с вычислением индекса точки относительно многоугольника
- Метод вычисления ближней точки и ее нормали

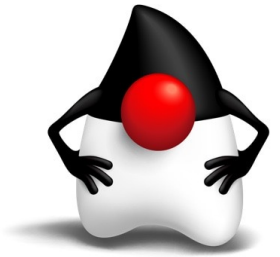


Точка принадлежит многоугольнику, если?

Математики скажут:

- Находится внутри многоугольника
- Находится на одном из ребер или в месте их пересечения

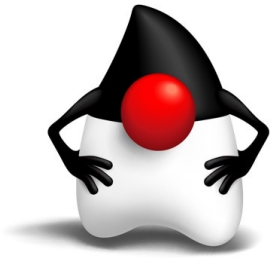
Однако функции PostGIS так не считают или, считают, но не все….



ST_WITHIN

Возвращает TRUE, если геометрия A полностью находится внутри геометрии B. Чтобы эта функция имела смысл, обе исходные геометрии должны иметь одну и ту же координатную проекцию и иметь одинаковый SRID. Принято считать, что если ST_Within(A,B) истинно и ST_Within(B,A) истинно, то две геометрии считаются пространственно равными.

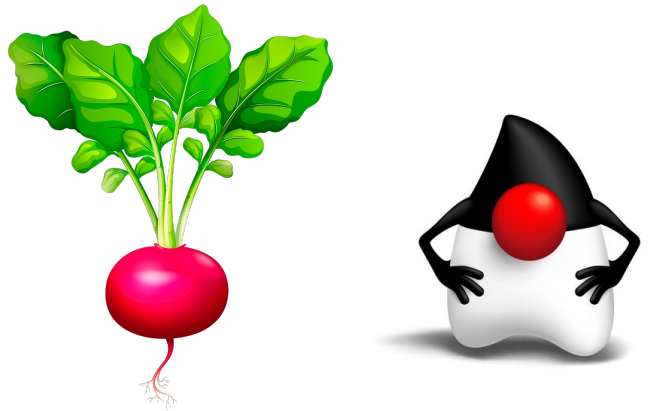
Тонкость этого определения в том, что граница геометрии не находится внутри геометрии. Это означает, что линии и точки, лежащие на границе многоугольника или линии, не входят в геометрию.



ST_WITHIN

Возвращает TRUE, если геометрия A полностью находится внутри геометрии B. Чтобы эта функция имела смысл, обе исходные геометрии должны иметь одну и ту же координатную проекцию и иметь одинаковый SRID. Принято считать, что если ST_Within(A,B) истинно и ST_Within(B,A) истинно, то две геометрии считаются пространственно равными.

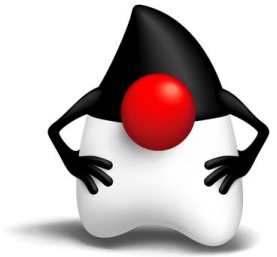
Тонкость этого определения в том, что граница геометрии не находится внутри геометрии. Это означает, что линии и точки, лежащие на границе многоугольника или линии, не входят в геометрию.



ST_CONTAINS

Возвращает TRUE, если геометрия B полностью находится внутри геометрии A. A содержит B тогда и только тогда, когда никакие точки B не лежат снаружи A, и по крайней мере одна точка внутри B лежит внутри A.

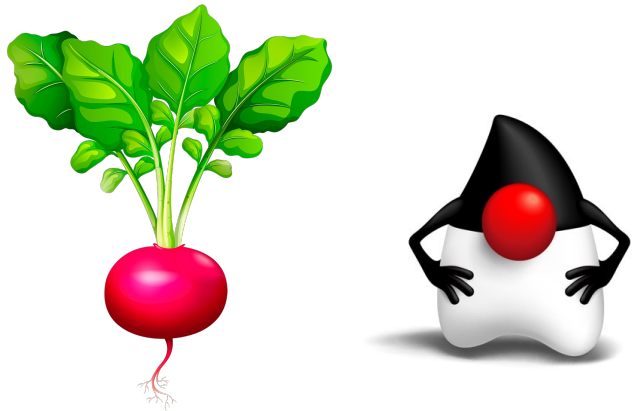
Тонкость этого определения в том, что граница геометрии не находится внутри геометрии. Это означает, что линии и точки, лежащие на границе многоугольника или линии, не входят в геометрию.



ST_CONTAINS

Возвращает TRUE, если геометрия B полностью находится внутри геометрии A. A содержит B тогда и только тогда, когда никакие точки B не лежат снаружи A, и по крайней мере одна точка внутри B лежит внутри A.

Тонкость этого определения в том, что граница геометрии не находится внутри геометрии. Это означает, что линии и точки, лежащие на границе многоугольника или линии, не входят в геометрию.



ST_COVERS

Возвращает true, если ни одна точка в геометрии/географии B не находится за пределами геометрии/географии A. Эквивалентно, проверяет, находится ли каждая точка геометрии B внутри (т. е. пересекает внутреннюю часть или границу) геометрии A.

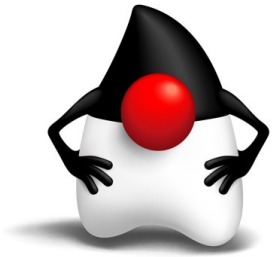
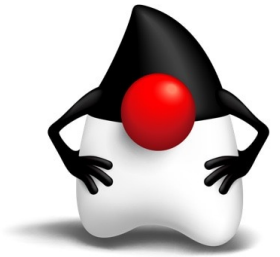


Таблица и сущность

```
create table polygon_entity
(
    id          BIGINT          NOT NULL PRIMARY KEY,
    name        VARCHAR,
    coordinates geometry NOT NULL
);
```

```
@Entity
@Table(name = "polygon_entity")
open class PolygonEntity {
 // какие-то поля и методы
    @Column(columnDefinition = "geometry", nullable = false)
    val coordinates: Polygon
}
```



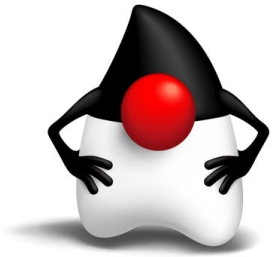
Репозиторий и скрипт наполнния

```
interface PolygonRepository : JpaRepository<PolygonEntity, Long> {
    @Query(value = "select * from polygon_entity pe " +
            "where st_covers(pe.coordinates, :point) = true",
        nativeQuery = true)
    fun selectPoligonsWithPoint(@Param("point") point: Point): List<PolygonEntity>
}
```

Заполним нашу таблицу данными подобными этим:

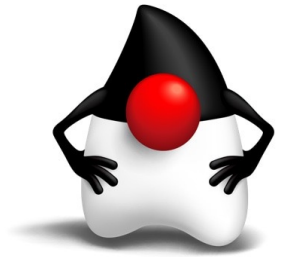
```
INSERT INTO polygon_entity(id, name, coordinates)
VALUES (1, 'Какой-то многоугольник',
    ST_GeometryFromText(
        'POLYGON((50.6373 3.0750,50.6374 3.0750,50.6374 3.0749,50.63 3.07491,50.6373 3.0750))', 4326
    ))
```

Результирующий запрос вернет нам набор всех многоугольников, которые содержат точку.

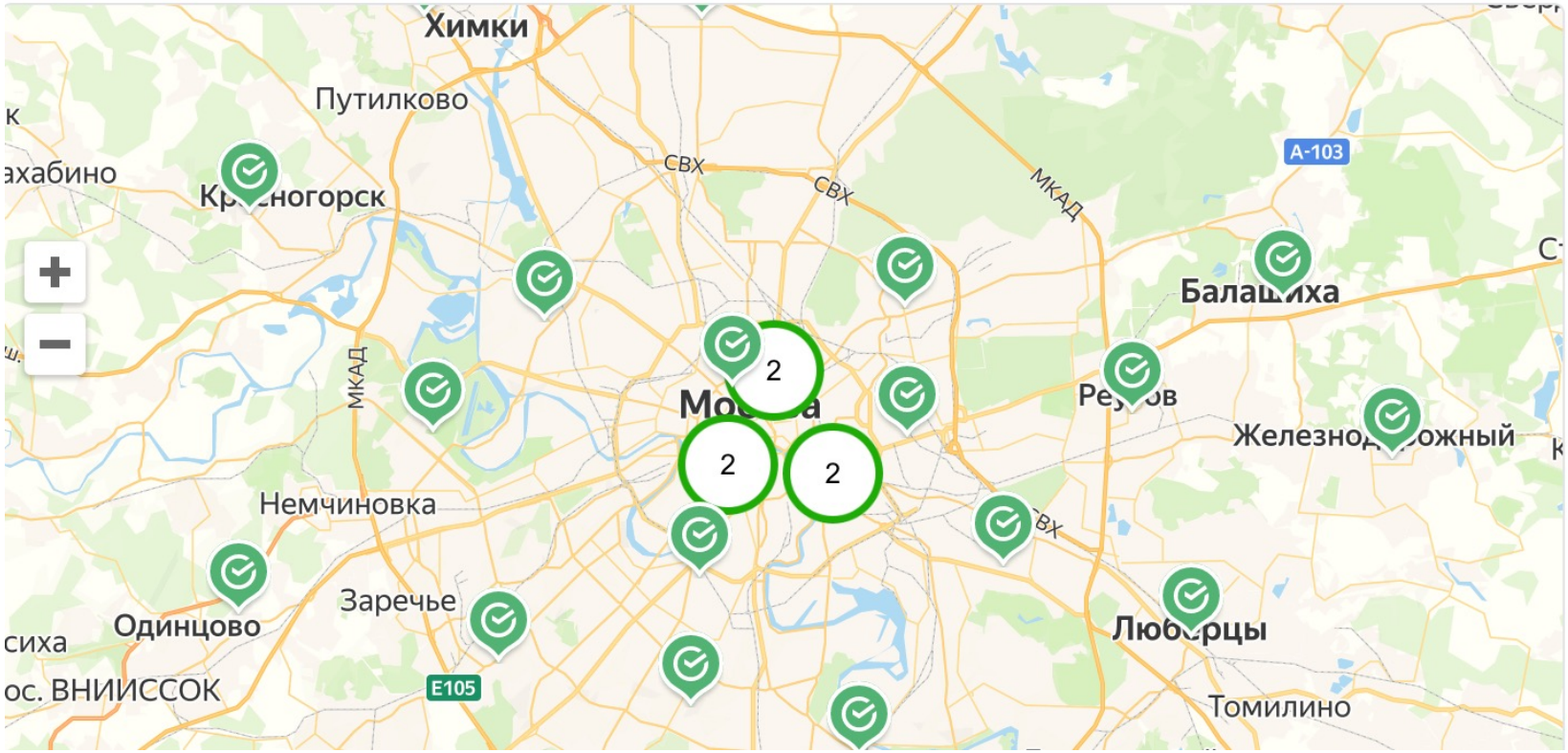


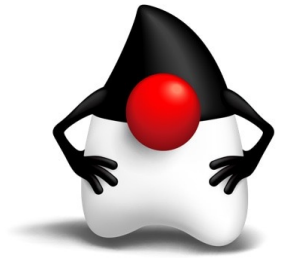
Задача 2

Выяснить принадлежит ли точка многоугольнику(полигону) или выбрать все точки внутри некоторого пространства



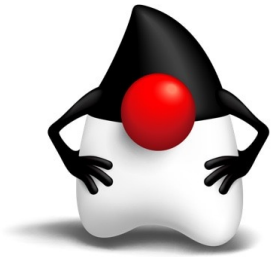
А когда это бывает надо?





Запрос

```
@Query(
    value = "select * from shop s
where st_within(s. position, st_makeenvelope(:xmin,:ymin,:xmax,:ymax, 4326)) = true",
    nativeQuery = true
)
fun findAllShopsInViewPort(
    @Param("xmin") xmin: Double,
    @Param("ymin") ymin: Double,
    @Param("xmax") xmax: Double,
    @Param("ymax") ymax: Double
): List<Shop>
```

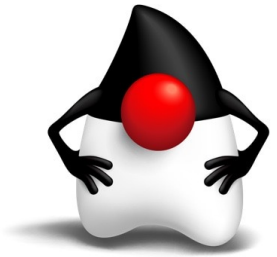


Запрос

```
@Query(
    value = "select * from shop s
where st_within(s. position, st_makeenvelope(:xmin,:ymin,:xmax,:ymax, 4326)) = true",
    nativeQuery = true
)
fun findAllShopsInViewPort(
    @Param("xmin") xmin: Double,
    @Param("ymin") ymin: Double,
    @Param("xmax") xmax: Double,
    @Param("ymax") ymax: Double
): List<Shop>
```

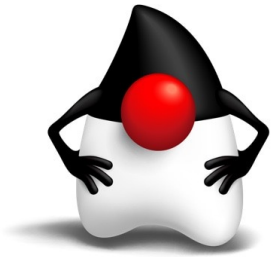
Если используете Hibernate 6, то можно вот так:

```
@Query("select shop from Shop shop where within(shop.position, :viewPort)")
fun findAlShopsInViewPort(@Param("viewPort")viewPort: Polygon): List<Shop>
```



Из двух координат в полигон

```
val envelope = Envelope(xMin, yMin, xMax, yMax)

fun toPolygon(env: Envelope): Polygon {
    val coords = arrayOa<Coordinate>(5)
       Coordinate(env.minX, env.minY)
       Coordinate(env.minX, env.maxY)
       Coordinate(env.maxX, env.maxY)
       Coordinate(env.maxX, env.minY)
       Coordinate(env.minX, env.minY)
    val shell: LinearRing =
geometryFactory.createLinearRing(coords)
    return geometryFactory.createPolygon(shell, null)
}
```



Индексы PostGIS

Пространственные индексы используются в PostGIS для быстрого поиска объектов в пространстве. Практически это означает очень быстро отвечать на вопросы вида:

- Найди все объекты внутри этой штуки
- Найди все штуки рядом с этой штукой

Главная проблема гео и пространственных данных в том, что когда они есть в реальной жизни, например, внутри гис-системы, то их обычно умопомачительно много и они тяжелые. Нам на помощь приходят индексы.

С пространственными объектами и геоданными умеют работать:

- GIST
- SP-GIST
- BRIN

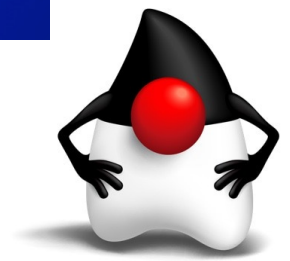
Про индексы просто и круто

JPoint

Индексы в PostgreSQL. Как понять, что создавать

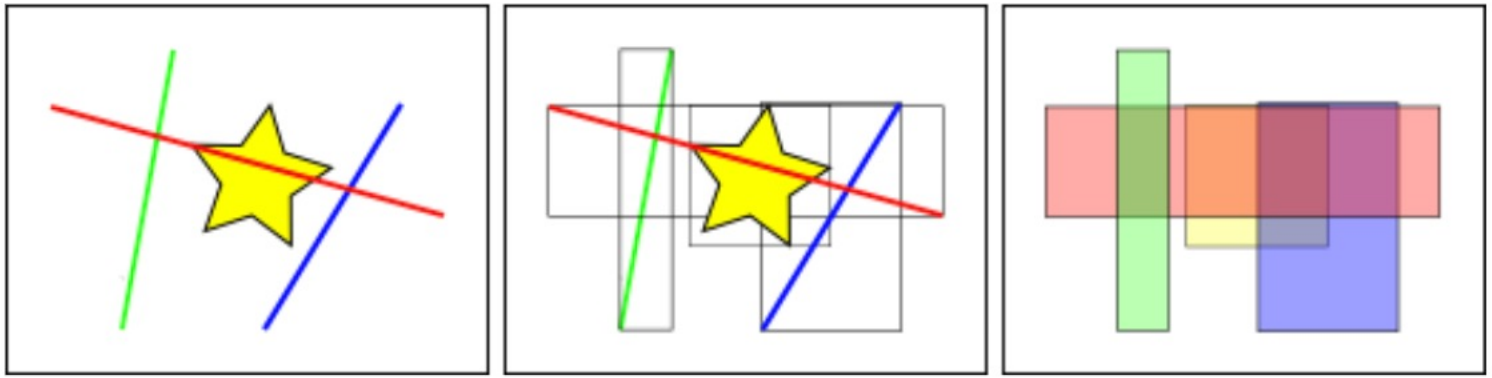
Андрей Сальников

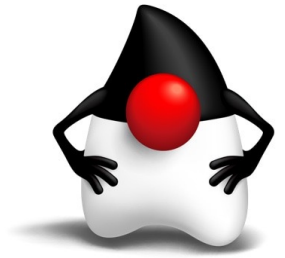
Data Egret



Как это работает?

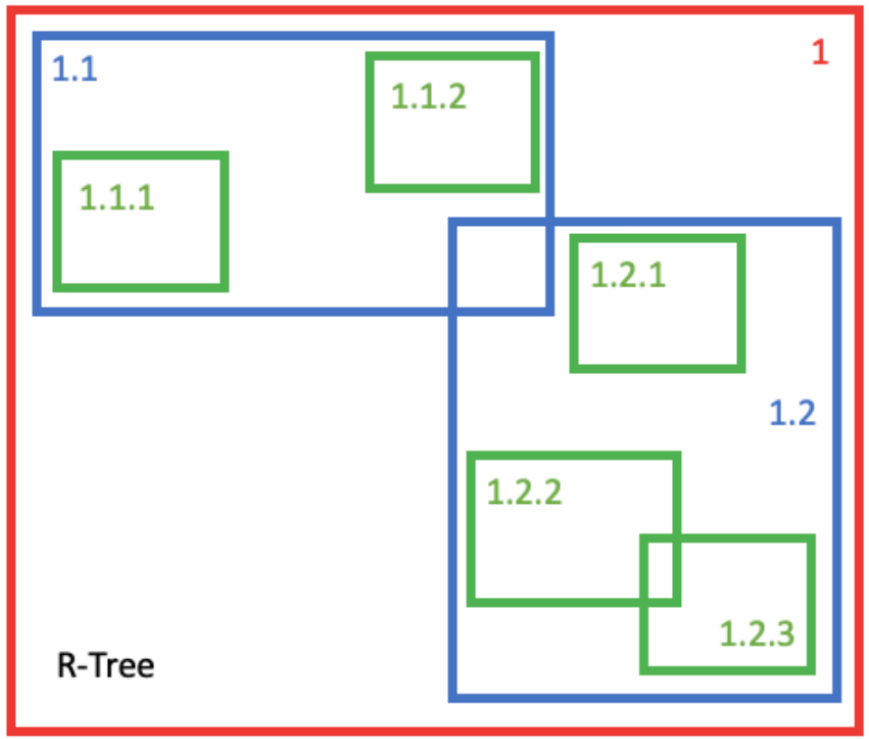
Индексируются не сами объекты, а «охватывающие» их прямоугольные рамки(bounding boxes). Документация PostGIS дает нам классный графический пример на этот счет.





Про деревья

R-TREE



GIST

QUAD-TREE

SP-GIST

Индексы PostGIS есть ли разница?

Вводные:
Таблица с 10000 записей, каждая из которых содержит в себе полигон с определенным количеством точек и просто таблица в которой хранится 135 точек.

Запросы

- SELECT *
  FROM polygon_entity pe
  WHERE *st_within*(
    st_setsrid(*st_makepoint*(59.28819, 44.08043), 4326),
    pe.coordinates) = TRUE

- ```
  SELECT *
  FROM polygon_entity pe
  WHERE st_covers(
    pe.coordinates,
    st_setsrid(st_makepoint(59.28819, 44.08043), 4326)) = TRUE
  ```

- ```
  SELECT * FROM points
  WHERE active = true
  ORDER BY st_distancesphere(
      geography,
      st_setsrid(st_makepoint(54.58266, 39.72378), 4326
          )
      )
  ```

Запрос 1

Без индексов

	QUERY PLAN text
1	Seq Scan on polygon_entity pe (cost=0.00..250461.00 rows=1 width=258) (actual time=0.019..15.645 rows=4978 loops=1)
2	Filter: st_within('0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry, coordinates)
3	Rows Removed by Filter: 5022
4	Planning Time: 1.038 ms
5	Execution Time: 15.995 ms

15,995 ms

GIST

	QUERY PLAN text
1	Bitmap Heap Scan on polygon_entity pe (cost=4.23..289.33 rows=1 width=258) (actual time=2.751..17.813 rows=4978 loops=1)
2	Filter: st_within('0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry, coordinates)
3	Rows Removed by Filter: 3390
4	Heap Blocks: exact=361
5	-> Bitmap Index Scan on polygons_index (cost=0.00..4.23 rows=10 width=0) (actual time=2.671..2.671 rows=8368 loops=1)
6	Index Cond: (coordinates ~ '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
7	Planning Time: 0.769 ms
8	Execution Time: 18.193 ms

18,193 ms

Запрос 1

BRIN

	QUERY PLAN text
1	Seq Scan on polygon_entity pe (cost=0.00..250461.00 rows=1 width=258) (actual time=0.018..15.672 rows=4978 loops=1)
2	Filter: st_within('0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry, coordinates)
3	Rows Removed by Filter: 5022
4	Planning Time: 0.102 ms
5	Execution Time: 16.022 ms

16,022 ms

SP-GIST

	QUERY PLAN text
1	Bitmap Heap Scan on polygon_entity pe (cost=4.23..289.33 rows=1 width=258) (actual time=3.122..18.287 rows=4978 loops=1)
2	Filter: st_within('0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry, coordinates)
3	Rows Removed by Filter: 3390
4	Heap Blocks: exact=361
5	-> Bitmap Index Scan on polygons_index (cost=0.00..4.23 rows=10 width=0) (actual time=3.037..3.037 rows=8368 loops=1)
6	Index Cond: (coordinates ~ '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
7	Planning Time: 0.107 ms
8	Execution Time: 18.653 ms

18,653 ms

Запрос 2

Без индексов

	QUERY PLAN text
1	Seq Scan on polygon_entity pe (cost=0.00..250461.00 rows=1 width=258) (actual time=0.019..14.837 rows=4979 loops=1)
2	Filter: st_covers(coordinates, '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
3	Rows Removed by Filter: 5021
4	Planning Time: 0.083 ms
5	Execution Time: 15.172 ms

15,172 ms

GIST

	QUERY PLAN text
1	Bitmap Heap Scan on polygon_entity pe (cost=4.23..289.33 rows=1 width=258) (actual time=2.968..20.895 rows=4979 loops=1)
2	Filter: st_covers(coordinates, '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
3	Rows Removed by Filter: 3389
4	Heap Blocks: exact=361
5	-> Bitmap Index Scan on polygons_index (cost=0.00..4.23 rows=10 width=0) (actual time=2.824..2.825 rows=8368 loops=1)
6	Index Cond: (coordinates ~ '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
7	Planning Time: 0.103 ms
8	Execution Time: 21.353 ms

21,353 ms

Запрос 2

BRIN

	QUERY PLAN text
1	Seq Scan on polygon_entity pe (cost=0.00..250461.00 rows=1 width=258) (actual time=0.014..12.620 rows=4979 loops=1)
2	Filter: st_covers(coordinates, '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
3	Rows Removed by Filter: 5021
4	Planning Time: 0.093 ms
5	Execution Time: 12.915 ms

12,915 ms

SP-GIST

	QUERY PLAN text
1	Bitmap Heap Scan on polygon_entity pe (cost=4.23..289.33 rows=1 width=258) (actual time=2.795..18.534 rows=4979 loops=1)
2	Filter: st_covers(coordinates, '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
3	Rows Removed by Filter: 3389
4	Heap Blocks: exact=361
5	-> Bitmap Index Scan on polygons_index (cost=0.00..4.23 rows=10 width=0) (actual time=2.710..2.711 rows=8368 loops=1)
6	Index Cond: (coordinates ~ '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geometry)
7	Planning Time: 0.104 ms
8	Execution Time: 18.918 ms

18,918 ms

Запрос 3

Без индексов

	QUERY PLAN text
1	Sort (cost=3366.42..3366.75 rows=134 width=255) (actual time=1.777..1.792 rows=134 loops=1)
2	Sort Key: (st_distance(geography(geography), '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geography, false))
3	Sort Method: quicksort Memory: 86kB
4	-> Seq Scan on bank_office (cost=0.00..3361.68 rows=134 width=255) (actual time=1.020..1.520 rows=134 loops=1)
5	Filter: active
6	Rows Removed by Filter: 1
7	Planning Time: 0.160 ms
8	Execution Time: 1.929 ms

1,929 ms

GIST

	QUERY PLAN text
1	Sort (cost=3366.42..3366.75 rows=134 width=255) (actual time=1.382..1.396 rows=134 loops=1)
2	Sort Key: (st_distance(geography(geography), '0101000020E6100000FB57569A944A4B40DBBFB2D2A4DC4340'::geography, false))
3	Sort Method: quicksort Memory: 86kB
4	-> Seq Scan on bank_office (cost=0.00..3361.68 rows=134 width=255) (actual time=0.744..1.132 rows=134 loops=1)
5	Filter: active
6	Rows Removed by Filter: 1
7	Planning Time: 0.212 ms
8	Execution Time: 1.549 ms

1,549 ms

В ЧЕМ ДЕЛО?

-> Seq Scan on bank_office (cost=0.00..3361.68 rows=134 width=255) (actual time=1.020..1.520 rows=134 loops=1)

КОГДА ДАННЫХ МНОГО

Добавим в нашу таблицу полигонов 3588712 полигона и попробуем снова.

- SELECT *
  FROM polygon_entity pe
  WHERE *st_within*(
  st_setsrid(*st_makepoint*(59.28819, 44.08043), 4326),
  pe.coordinates) = TRUE

БЕЗ ИНДЕКСОВ

9429,678 ms

Statistics per Node Type

Node type	Count	Time spent	%% of query
Gather	1	-6230.331 ms	-194.73%
Seq Scan	1	9429.678 ms	294.74%

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
polygon_entity	1	9429.678 ms	294.74%
Seq Scan	1	9429.678 ms	100%

BRIN

10423,152 ms

Statistics per Node Type

Node type	Count	Time spent	%% of query
Bitmap Heap Scan	1	10423.152 ms	294.86%
Bitmap Index Scan	1	13.329 ms	0.38%
Gather	1	-6901.443 ms	-195.22%

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
polygon_entity	1	10423.152 ms	294.86%
Bitmap Heap Scan	1	10423.152 ms	100%

GIST

3362,491 ms

Statistics per Node Type

Node type	Count	Time spent	%% of query
Bitmap Heap Scan	1	3362.491 ms	237.12%
Bitmap Index Scan	1	522.716 ms	36.87%
Gather	1	-2467.143 ms	-173.97%

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
polygon_entity	1	3362.491 ms	237.12%
Bitmap Heap Scan	1	3362.491 ms	100%

SP-GIST

2993,261 ms

Statistics per Node Type

Node type	Count	Time spent	%% of query
Bitmap Heap Scan	1	2993.261 ms	248.87%
Bitmap Index Scan	1	417.823 ms	34.74%
Gather	1	-2208.341 ms	-183.6%

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
polygon_entity	1	2993.261 ms	248.87%
Bitmap Heap Scan	1	2993.261 ms	100%

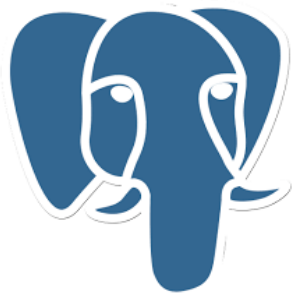
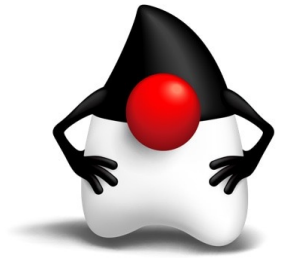


Таблица на миллион

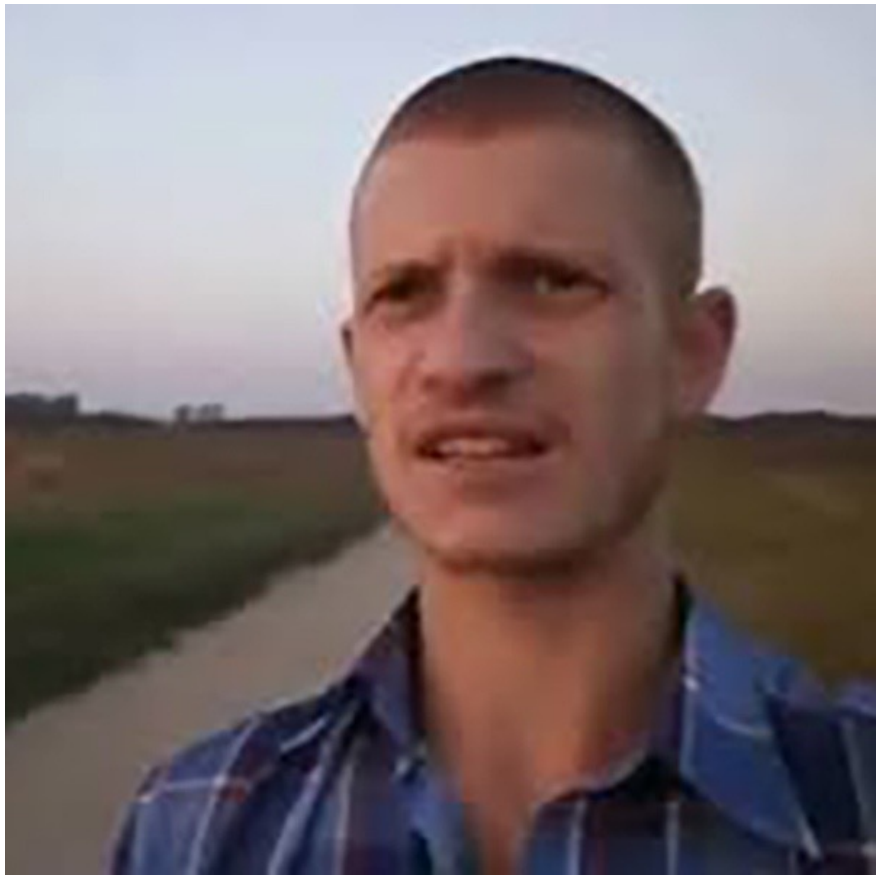
```
CREATE TABLE random_points AS
  SELECT ST_MakePoint(
    1000*random(),
    1000*random(),
    1000*random())::geometry(PointZ) AS geom,
    pk
  FROM generate_series(1,1000000) pk;
ALTER TABLE random_points ADD PRIMARY KEY (pk);
```

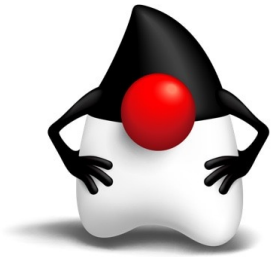
Запрос ПОЗЛЕЕ

```
WITH xy AS (
SELECT 1000*random() AS x, 1000*random() AS y
FROM generate_series(1, 1000)
),
boxes AS (
  SELECT ST_MakeEnvelope(x-5, y-5, x+5, y+5) AS geom
  FROM xy
)
SELECT Count(*)
FROM random_points, boxes
WHERE random_points.geom && boxes.geom;
```



Успеваем
преисполниться
пока
это отработает





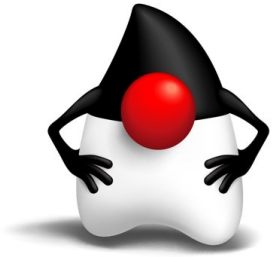
Реально жестко

Statistics per Node Type

Node type	Count	Time spent	%% of query
Aggregate	1	102.57 ms	0.01%
CTE Scan	1	145000 ms	11.04%
Function Scan	1	0.491 ms	0.01%
Nested Loop Inner Join	1	1168189.016 ms	88.92%
Seq Scan	1	522.445 ms	0.04%

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
public.random_points	1	522.445 ms	0.04%
Seq Scan	1	522.445 ms	100%



GIST

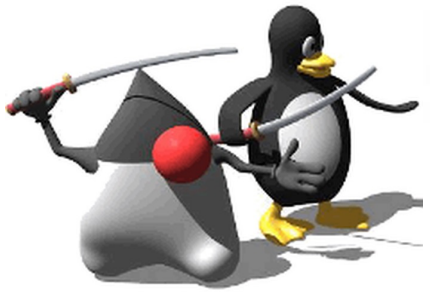
Statistics per Node Type

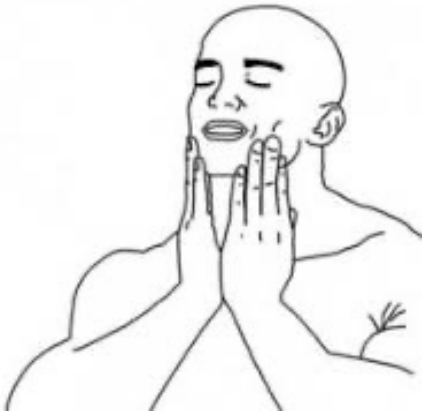
Node type	Count	Time spent	%% of query
Aggregate	1	11.505 ms	4.85%
CTE Scan	1	1.575 ms	0.67%
Function Scan	1	0.696 ms	0.3%
Index Scan	1	202 ms	85.04%
Nested Loop Inner Join	1	21.776 ms	9.17%

Statistics per Table

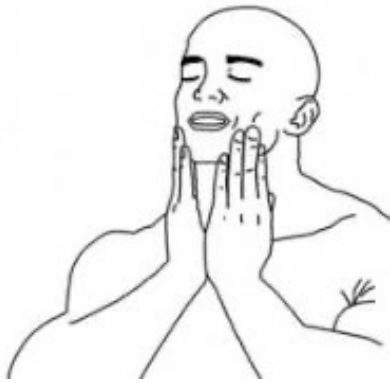
Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
random_points	1	202 ms	85.04%
Index Scan	1	202 ms	100%

202 ms





SP-GIST



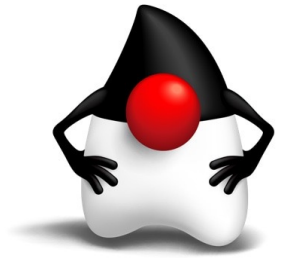
Statistics per Node Type

Node type	Count	Time spent	%% of query
Aggregate	1	12.201 ms	6.17%
CTE Scan	1	1.496 ms	0.76%
Function Scan	1	0.667 ms	0.34%
Index Scan	1	162 ms	81.89%
Nested Loop Inner Join	1	21.467 ms	10.86%

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
random_points	1	162 ms	81.89%
Index Scan	1	162 ms	100%

162 ms



BRIN

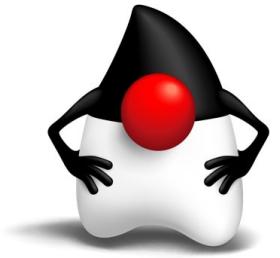
Statistics per Node Type

Node type	Count	Time spent	%% of query
Aggregate	1	110.947 ms	0.01%
Bitmap Heap Scan	1	1323896 ms	99.95%
Bitmap Index Scan	1	457 ms	0.04%
CTE Scan	1	5.42 ms	0.01%
Function Scan	1	2.443 ms	0.01%
Nested Loop Inner Join	1	106.551 ms	0.01%

Statistics per Table

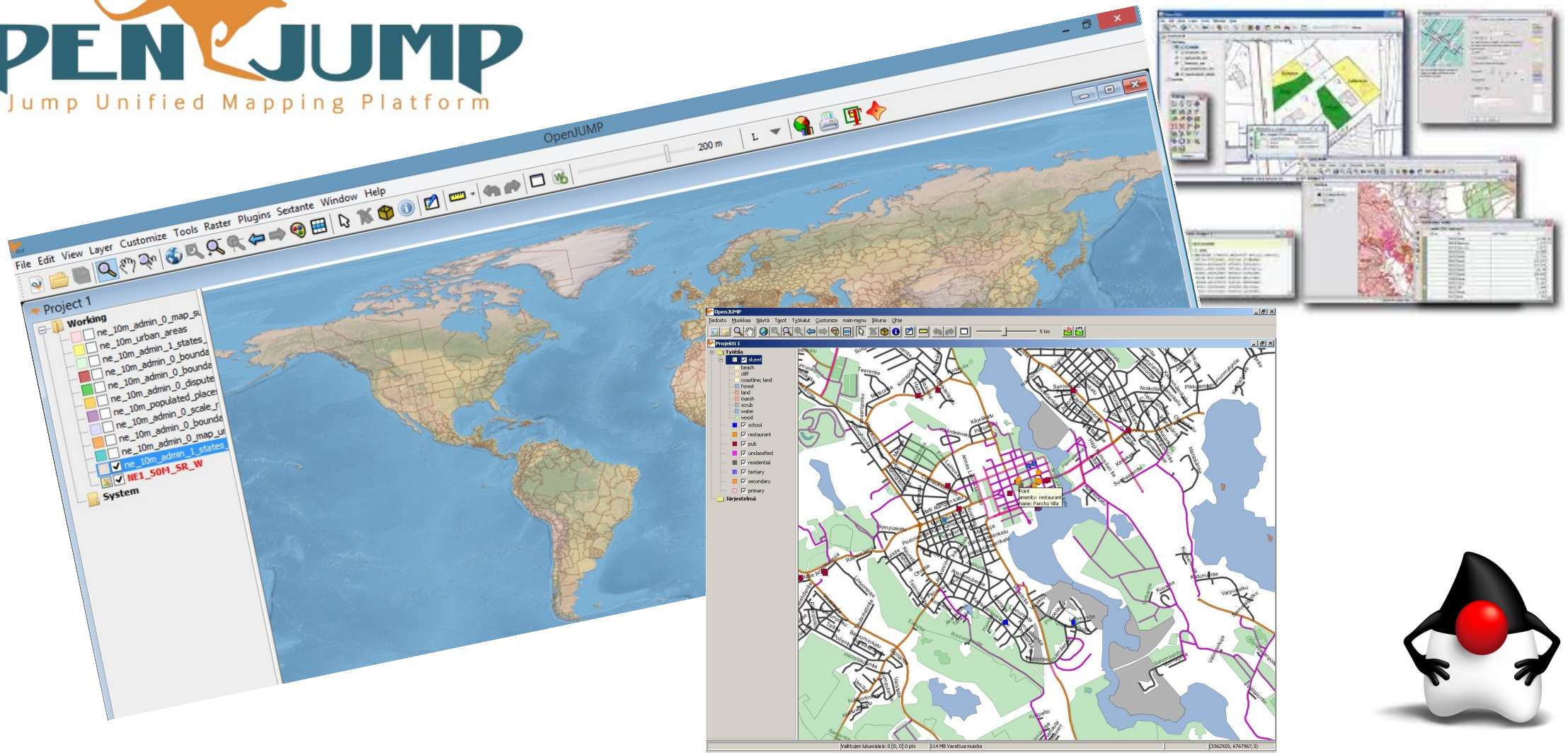
Table name	Scan count	Total time	%% of query
Node type	**Count**	**Sum of times**	**%% of table**
random_points	1	1323896 ms	99.95%
Bitmap Heap Scan	1	1323896 ms	100%

1323896 ms



Для пытливых умов

OPEN JUMP
Jump Unified Mapping Platform



На затравочку