Introduction to MySQL Spatial

Features and Real-World Use Cases

Vissarion Fisikopoulos

Friday 21 November 2025

Outline

A brief histoy of GIS in MySQL

Spatial Reference Systems

Spatial data types and functions

Storage and indexing

Examples and use cases

Outline

History of MySQL GIS

Milestones

- MySQL $\leq$ 5.6: Basic functions, homegrown algorithms
- MySQL 5.7: "Reboot", Boost Geometry, standard compiant (OGC, SQL/MM), only cartesian
- MySQL 8.0: Spatial reference systems

Boost Geometry as a MySQL geometry engine

- Part of Boost C++ libraries
- Open source; supported by a small but vibrant community
- All (heavy) geometric computations are done at Boost Geometry side
- Separation between geometry and database internals
- Generic design; works with MySQL data types; no conversions!

Build-in GIS support

- No plugin, works out of the box
- Fully integrated with other MySQL functionality
  - Data dictionary tables
  - INFORMATION_SCHEMA views (spatial metadata)
  - GeoJSON parsed by general JSON parser

Alternatives to MySQL Spatial

PostgreSQL + PostGIS

- OGC and SQL/MM standards
- Extensive function set: topology, raster, routing, 3D, geography
- Advanced indexing and optimizer integration

SQLite + SpatiaLite

- Supports most OGC functions and coordinate systems
- Supported by QGIS

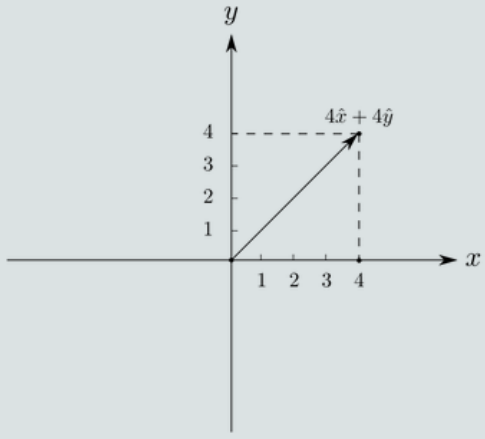
Microsoft SQL Server/ Oracle spatial

- Feature rich
- Optimized and standard compliant
- Closed source, limited flexibility

Outline

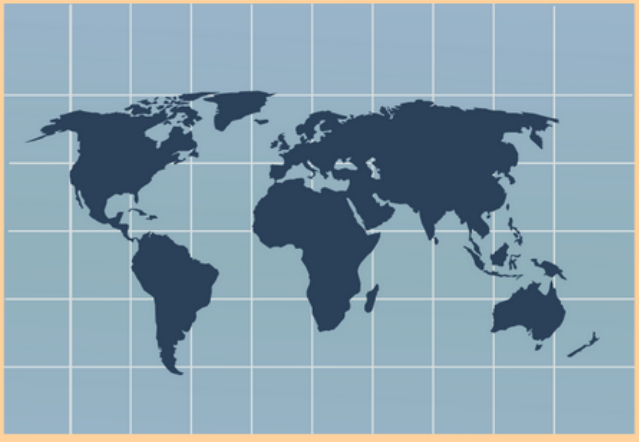
Spatial Reference Systems (SRS)

5.7



SRID 0

8.0



Projected SRS

Geographic SRS

Cartesian SRS

Spatial Reference Systems

Basic properties

- Each SRS has a unique spatial reference system ID (SRID)
- Each geometry has an SRID property
- De facto standard: EPSG Dataset
- Examples: 4326 = WGS 84 ("GPS coordinates"), 3857 = WGS 84 / World Mercator ("Web Mercator" - Google maps)
- A property of each geometry value
- Mixing geometries in different SRIDs in one computation doesn't make sense and will raise an error

Spatial Reference Systems

Current status

- MySQL currently ships with **5107** predefined CRS definitions from EPSG dataset v9.2
- Breakdown: 4628 projected, 479 geographic
- You can `CREATE` / `DROP` spatial reference systems to add custom CRSs
- In 8.0, MySQL refuses to perform computations unless it knows the CRS definition

```
CREATE TABLE cities
(
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(20) NOT NULL,
  location POINT SRID 4326 NOT NULL
);
```

Outline

Geometric Data Types in MySQL

- Open Geospatial Consortium (OGC) Simple Features specification.
- All geometry types share a common base type: `GEOMETRY`.

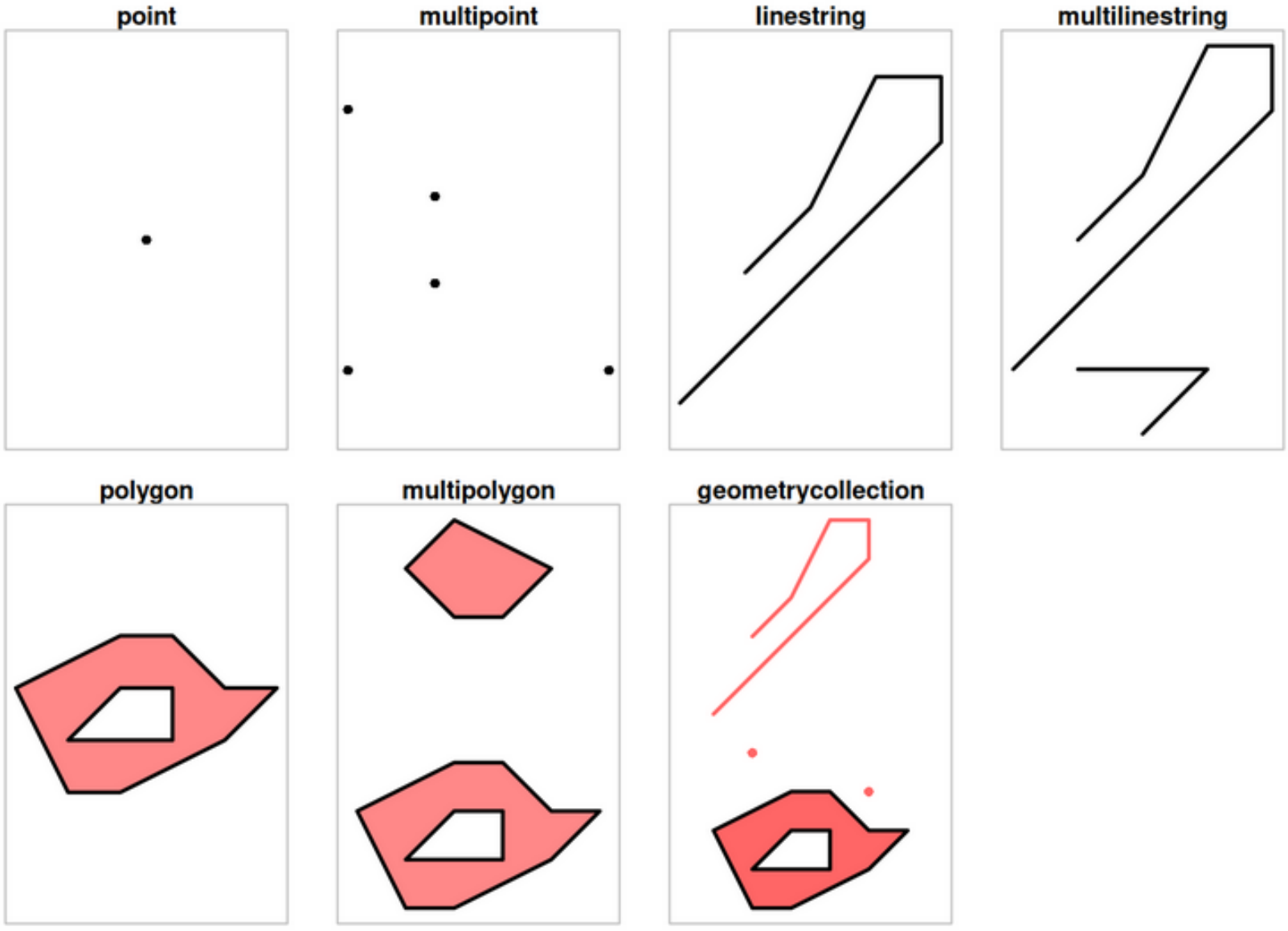
Main geometry classes:

- `POINT` – two coordinates (e.g., location of a city)
- `LINESTRING` – sequence of points forming a line or path
- `POLYGON` – closed shape with an outer ring and optional holes
- `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON` – collections of the above
- `GEOMETRYCOLLECTION` – heterogeneous set of geometries

Storage & SRID:

- Stored in binary (WKB) format with an optional SRID (spatial reference ID).
- Supports Cartesian (planar) and geographic (ellipsoidal) coordinate systems.

Geometric Data Types in MySQL



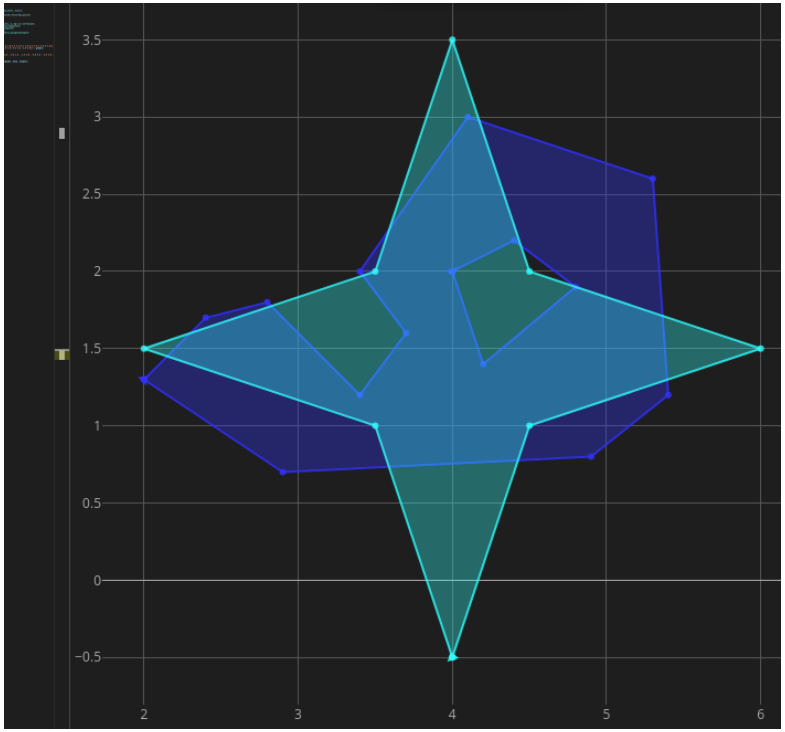
Spatial Functions in MySQL

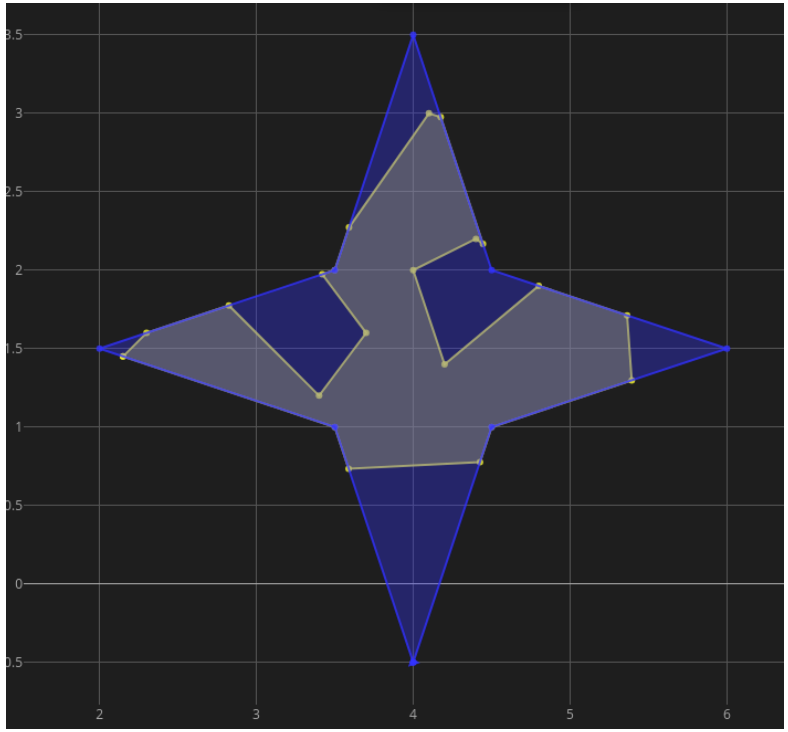
Categories of spatial functions

- Constructor functions
  - Create geometries from text or binary:
  - `ST_GeomFromText('POINT(10 20)', 4326)`
  - `ST_GeomFromGeoJSON({ "type": "Point", ... })`
- Conversion functions
  - Change between formats or extract metadata:
  - `ST_AsText(geom)`, `ST_AsGeoJSON(geom)`, `ST_SRID(geom)`
- Measurement functions
  - Compute distances, lengths, and areas:
  - `ST_Distance(a,b)`, `ST_Length(line)`, `ST_Area(poly)`
- Predicate functions
  - Boolean tests between geometries:
  - `ST_Intersects(a,b)`, `ST_Contains(a,b)`, `ST_Within(a,b)`
- Topological operations
  - `ST_Intersection(a,b)`, `ST_Union(a,b)`

Spatial Functions in MySQL

Coordinate-aware behavior





- For `SRID = 0`, MySQL performs planar (Cartesian) geometric computations.
- Otherwise, (e.g. 4326) functions perform ellipsoidal (geographic) computations.

Example

Compute the distance from Athens to Brussels (lat,lon)

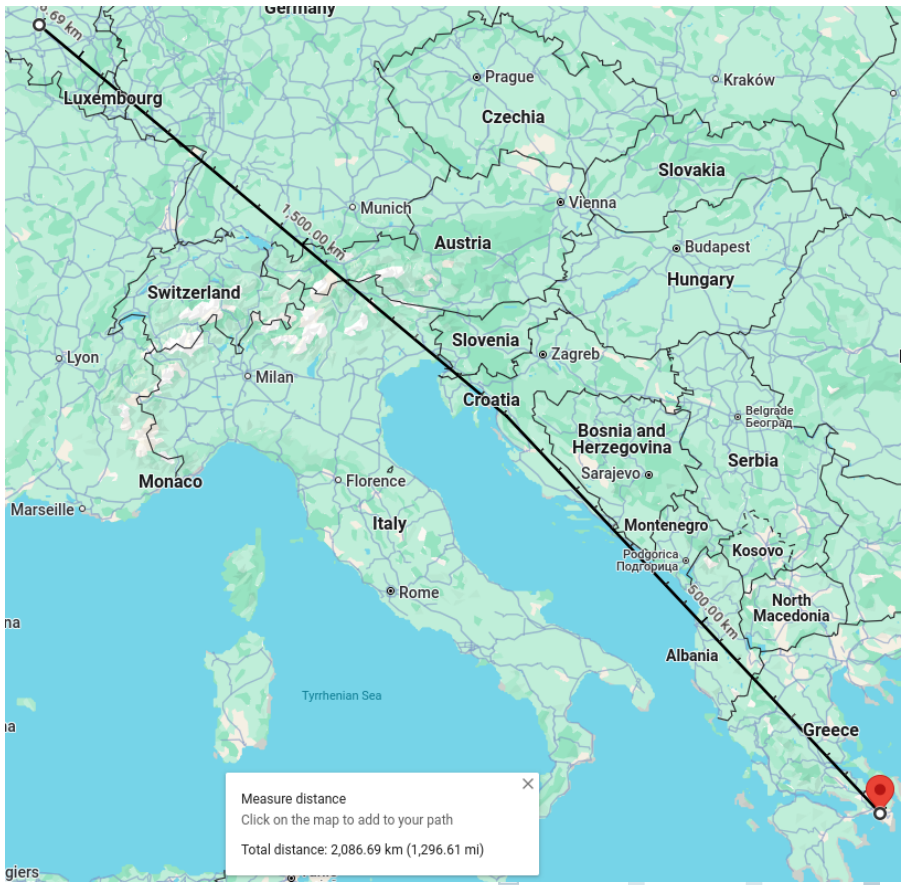
```
INSERT INTO cities (name, location)
VALUES
( 'Athens',
  ST_GeomFromText('POINT(37.971536 23.725750)', 4326)
);
```

```
INSERT INTO cities (name, location)
VALUES
( 'Brussels',
  ST_GeomFromText('POINT(50.8119483 4.3826169)', 4326)
);
```

Example

Compute the distance from Athens to Brussels

```
SELECT
  ST_DISTANCE(
    (SELECT location FROM cities WHERE name ='Athens'),
    (SELECT location FROM cities WHERE name ='Brussels')
  ) AS distance_meters;
```

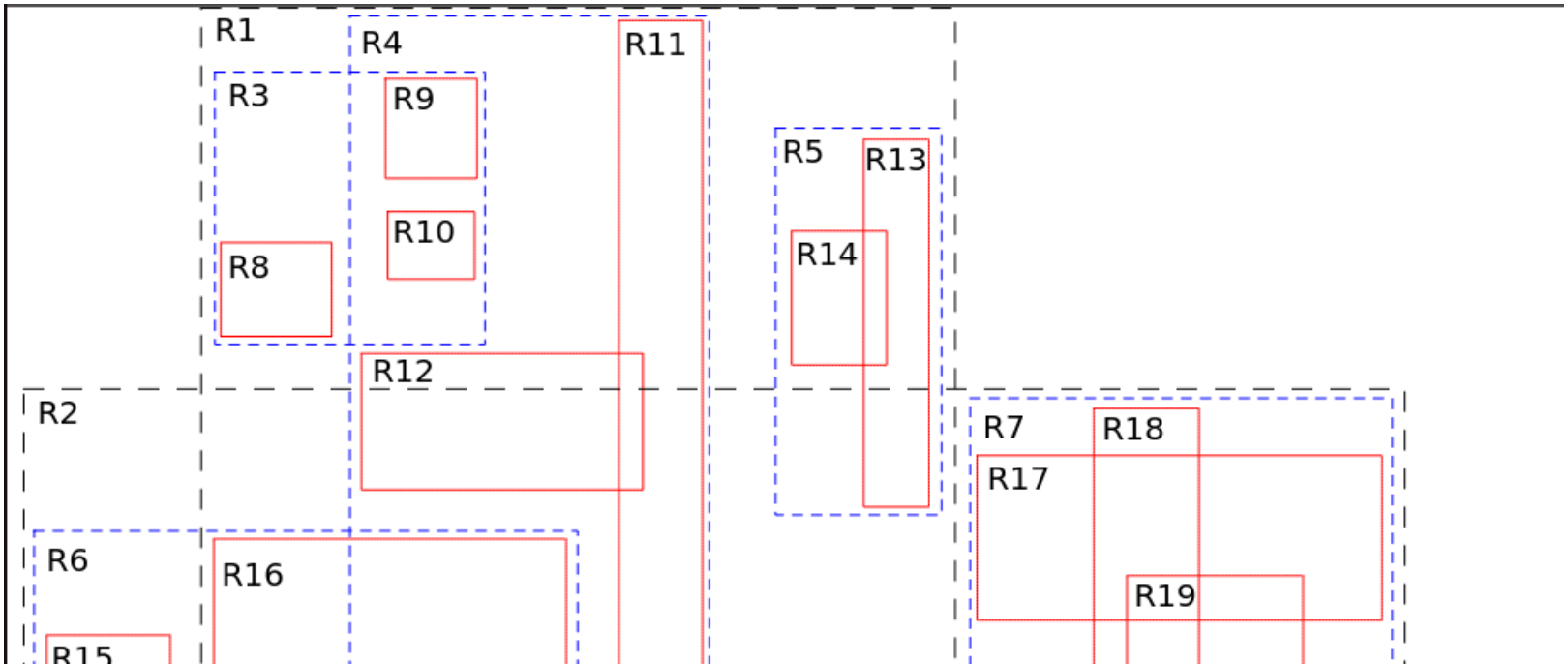


result: 2088389.0786590837 m

Outline

MySQL/InnoDB Spatial Index

- R-tree index on the *MBR* (minimum bounding rectangle) of geometries.
- Supported on InnoDB for `GEOMETRY`/subtypes (`POINT`, `POLYGON`, ...).
- Accelerates *MBR-sargable* predicates (e.g., `MBRIntersects()`, `MBRWithin()`, `ST_Intersects()`, ...
- *coarse pre-filter* before exact geometry checks.
- Column should be `NOT NULL` and consistent `SRID` (e.g., 4326).



Use of Spatial Index

Find all cities in Europe.

```
ALTER TABLE cities
ADD SPATIAL INDEX(location);
```

```
SELECT name
FROM cities
WHERE ST_Contains(GeomFromText('POLYGON((...))'),
                  location);
```

- Use index for large datasets
- Geographic aware if SRID $\neq 0$ is used

Outline

Transformations between coordinate systems

Scenario:

- A dataset of cities uses GPS coordinates in the WGS 84 system (`SRID 4326`) — a *geographic* coordinate system on a sphere-like Earth.
- A hiking trail map uses the British National Grid (`SRID 27700`) — a *projected* coordinate system on a flat surface.

The problem:

- Display cities on the hiking trail map.
- Same location has different numeric coordinates in each system.

Solution: project 4326 $\rightarrow$ 27700.

Transformations between coordinate systems

```
CREATE TABLE cities (
  name VARCHAR(45) PRIMARY KEY,
  location GEOMETRY NOT NULL SRID 4326
)ENGINE=InnoDB;

INSERT INTO cities (name, location)
VALUES ('London',
  ST_Geomfromtext('POINT(51.509865 -0.118092)', 4326)),
       ('Edinburgh',
  ST_Geomfromtext('POINT(55.953251 -3.188267)', 4326));

SELECT name, ST_AsText(ST_Transform(location, 27700))
   AS location_british_national_grid
FROM cities;
```

```
+-------------+-----------------------------------------+
| name        | location_british_national_grid          |
+-------------+-----------------------------------------+
| Edinburgh   | POINT(325899.1849114155 673995.7129374838)  |
| London      | POINT(530695.3868317431 180671.48838623578) |
+-------------+-----------------------------------------+
```

Projected Hiking trails

Hiking trails (orange) in Lake District (northwest England). Map data from OpenStreetMap; visualization with QGIS

Analysing trajectories

Scenario:

- A dataset of trajectories representing movements of walkers in a park.

The problem:

- Given a query trajectory return the set of the most similar trajectories.

Solution: Use trajectory similarity functions like frechet distance

Analyzing trajectories with MySQL

- Each walker's path represented as a `linestring` of GPS coordinates (lon, lat).
- Fréchet distance: captures similarity of two curves while respecting traversal order.
- Preprocess: simplify algorithm and R-tree
- No projections: same algorithm for geographic and cartesian CS

Motivation:

- identifying common travel patterns
- detecting anomalies in movement
- optimizing routes based on historical data
- spatial data cleansing: matching tracks created from noisy, or slightly inaccurate, GPS data to known roads or paths.

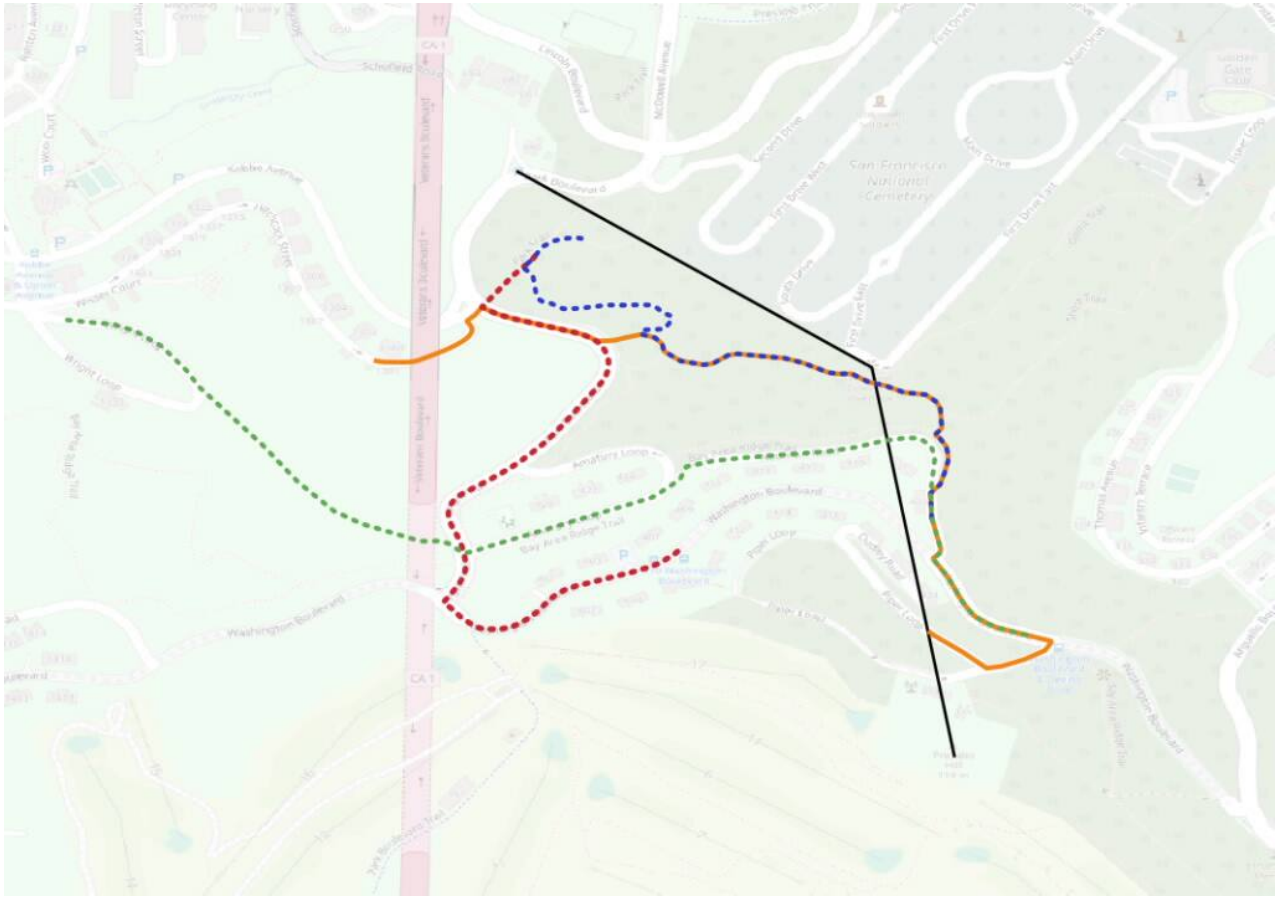
Walkers' trajectories example

```
CREATE TABLE walk_trajectories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    walker_name VARCHAR(100),
    trajectory LINESTRING NOT NULL SRID 4326
);

INSERT INTO walk_trajectories (walker_name, trajectory)
    VALUES
('Red Walker', ST_GeomFromText('LINESTRING
    (37.79562087383118 -122.46702694852982,37.7956012
    -122.4670506,...,37.79858158135988
    -122.46840682510111)', 4326));

SELECT walker_name, id,
  ST_FrechetDistance(trajectory,
                     (SELECT trajectory
                      FROM walk_trajectories
                      WHERE walker_name='Red Walker'))
                          as similarity
FROM walk_trajectories;
```

Walkers' trajectories example



453.59273618286403

241.65190882342154

261.68893003364565

429.19109867209164

Resources

- MySQL source code:

- MySQL documentation:

- MySQL spatial blogs:

Thank you — Questions?